Introduction to Programming I

Teacher's Manual

Version 1.3
June 2006
Requirements For the Laboratory Exercises

Supported Operating Systems

The NetBeans IDE 5.5 runs on operating systems that support the Java VM.
- Microsoft Windows XP Professional SP2 or newer
- Mac OS X 10.4.5 or newer
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System Update 1 (SPARC® and x86/x64 Platform Edition)

NetBeans Enterprise Pack is also known to run on the following platforms:
- Microsoft Windows 2000 Professional SP4
- Solaris™ 8 OS (SPARC and x86/x64 Platform Edition) and Solaris 9 OS (SPARC and x86/x64 Platform Edition)
- Various other Linux distributions

Minimum Hardware Configuration

**Note:** The NetBeans IDE’s minimum screen resolution is 1024x768 pixels.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
<th>Memory</th>
<th>Disk Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>500 MHz Intel Pentium III</td>
<td>512 MB</td>
<td>850 MB of free disk space</td>
</tr>
<tr>
<td>Linux</td>
<td>500 MHz Intel Pentium III workstation or equivalent</td>
<td>512 MB</td>
<td>450 MB of free disk space</td>
</tr>
<tr>
<td>Solaris OS (SPARC)</td>
<td>UltraSPARC II 450 MHz</td>
<td>512 MB</td>
<td>450 MB of free disk space</td>
</tr>
<tr>
<td>Solaris OS (x86/x64 Platform Edition)</td>
<td>AMD Opteron 100 Series 1.8 GHz</td>
<td>512 MB</td>
<td>450 MB of free disk space</td>
</tr>
<tr>
<td>Macintosh OS X operating system</td>
<td>PowerPC G4</td>
<td>512 MB</td>
<td>450 MB of free disk space</td>
</tr>
</tbody>
</table>

Recommended Hardware Configuration

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
<th>Memory</th>
<th>Disk Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>1.4 GHz Intel Pentium III workstation or equivalent</td>
<td>1 GB</td>
<td>1 GB of free disk space</td>
</tr>
<tr>
<td>Linux</td>
<td>1.4 GHz Intel Pentium III workstation or equivalent</td>
<td>1 GB</td>
<td>850 MB of free disk space</td>
</tr>
<tr>
<td>Solaris OS (SPARC)</td>
<td>UltraSPARC IIIi 1 GHz</td>
<td>1 GB</td>
<td>850 MB of free disk space</td>
</tr>
<tr>
<td>Solaris OS (x86/x64 Platform Edition)</td>
<td>AMD Opteron 100 Series 1.8 GHz</td>
<td>1 GB</td>
<td>850 MB of free disk space</td>
</tr>
<tr>
<td>Macintosh OS X operating system</td>
<td>PowerPC G5</td>
<td>1 GB</td>
<td>850 MB of free disk space</td>
</tr>
</tbody>
</table>

Required Software

NetBeans Enterprise Pack 5.5 Early Access runs on the Java 2 Platform Standard Edition Development Kit 5.0 Update 1 or higher (JDK 5.0, version 1.5.0_01 or higher), which consists of the Java Runtime Environment plus developer tools for compiling, debugging, and running applications written in the Java language. Sun Java System Application Server Platform Edition 9 has been tested with JDK 5.0 update 6.

- For **Solaris**, **Windows**, and **Linux**, you can download the JDK for your platform from [http://java.sun.com/j2se/1.5.0/download.html](http://java.sun.com/j2se/1.5.0/download.html)

For more information, please visit: [http://www.netbeans.org/community/releases/40/relnotes.html](http://www.netbeans.org/community/releases/40/relnotes.html)
# Table of Contents

1 Introduction to Computer Programming......................................................... 12
  1.1 Objectives.................................................................................................. 12
  1.2 Introduction............................................................................................... 12
  1.3 Basic Components of a Computer............................................................. 13
    1.3.1 Hardware.......................................................................................... 13
      1.3.1.1 The Central Processing Unit...................................................... 13
      1.3.1.2 Memory.................................................................................... 13
      1.3.1.3 Input and Output Devices...................................................... 14
    1.3.2 Software............................................................................................. 14
  1.4 Overview of Computer Programming Languages..................................... 15
    1.4.1 What is a Programming Language?................................................ 15
  1.5 The Program Development Life Cycle.................................................... 16
    1.5.1 Problem Definition......................................................................... 17
    1.5.2 Problem Analysis........................................................................... 17
    1.5.3 Algorithm design and representation.......................................... 18
      1.5.3.1 Flowcharting Symbols and their meanings.............................. 19
    1.5.4 Coding and Debugging................................................................. 20
  1.6 Number Systems and Conversions.......................................................... 21
    1.6.1 Decimal......................................................................................... 21
    1.6.2 Binary............................................................................................. 21
    1.6.3 Octal............................................................................................... 21
    1.6.4 Hexadecimal.................................................................................. 21
    1.6.5 Conversions.................................................................................... 22
      1.6.5.1 Decimal to Binary / Binary to Decimal................................... 22
      1.6.5.2 Decimal to Octal (or Hexadecimal)/Octal (or Hexadecimal) to Decimal.... 23
      1.6.5.3 Binary to Octal / Octal to Binary........................................... 24
      1.6.5.4 Binary to Hexadecimal / Hexadecimal to Binary..................... 25
  1.7 Exercises.................................................................................................. 26
    1.7.1 Writing Algorithms......................................................................... 26
    1.7.2 Number Conversions....................................................................... 26
  Introduction to Java...................................................................................... 27
  1.8 Objectives............................................................................................... 27
  1.9 Java Background..................................................................................... 27
    1.9.1 A little Bit of History ................................................................. 27
    1.9.2 What is Java Technology?............................................................ 27
      1.9.2.1 A programming language...................................................... 27
      1.9.2.2 A development environment.............................................. 27
      1.9.2.3 An application environment............................................... 27
      1.9.2.4 A deployment environment................................................... 28
    1.9.3 Some Features of Java..................................................................... 28
      1.9.3.1 The Java Virtual Machine..................................................... 28
      1.9.3.2 Garbage Collection................................................................ 28
      1.9.3.3 Code Security....................................................................... 29
    1.9.4 Phases of a Java Program............................................................... 30
  2 Getting to know your Programming Environment.................................... 31
    2.1 Objectives............................................................................................. 31
    2.2 Introduction.......................................................................................... 31
    2.3 My First Java Program....................................................................... 31
    2.4 Using a Text Editor and Console...................................................... 32
2.4.1 Errors ................................................................. 42
2.4.1.1 Syntax Errors................................................. 42
2.4.1.2 Run-time Errors............................................. 43
2.5 Using NetBeans..................................................... 44
2.6 Exercises............................................................. 56
   2.6.1 Hello World!.................................................. 56
   2.6.2 The Tree......................................................... 56
3 Programming Fundamentals......................................... 57
   3.1 Objectives......................................................... 57
   3.2 Dissecting my first Java program........................ 57
   3.3 Java Comments.................................................. 59
      3.3.1 C++-Style Comments..................................... 59
      3.3.2 C-Style Comments......................................... 59
      3.3.3 Special Javadoc Comments............................... 59
   3.4 Java Statements and blocks.................................. 60
   3.5 Java Identifiers.................................................. 61
   3.6 Java Keywords................................................... 62
   3.7 Java Literals....................................................... 63
      3.7.1 Integer Literals............................................. 63
      3.7.2 Floating-Point Literals.................................... 63
      3.7.3 Boolean Literals........................................... 63
      3.7.4 Character Literals......................................... 64
      3.7.5 String Literals.............................................. 64
   3.8 Primitive data types............................................. 65
      3.8.1 Logical - boolean......................................... 65
      3.8.2 Textual - char............................................. 65
      3.8.3 Integral - byte, short, int & long.................... 66
      3.8.4 Floating Point - float and double................... 67
   3.9 Variables........................................................ 68
      3.9.1 Declaring and Initializing Variables.................. 68
      3.9.2 Outputting Variable Data............................... 69
      3.9.3 System.out.println() vs. System.out.print()...... 69
      3.9.4 Reference Variables vs. Primitive Variables....... 70
   3.10 Operators......................................................... 71
      3.10.1 Arithmetic operators..................................... 71
      3.10.2 Increment and Decrement operators.................. 74
      3.10.3 Relational operators..................................... 76
      3.10.4 Logical operators....................................... 79
         3.10.4.1 && (logical AND) and & (boolean logical AND).... 80
         3.10.4.2 || (logical OR) and | (boolean logical inclusive OR)........ 82
         3.10.4.3 ^ (boolean logical exclusive OR)................ 84
         3.10.4.4 ! (logical NOT)...................................... 85
      3.10.5 Conditional Operator (?:).............................. 86
      3.10.6 Operator Precedence..................................... 88
   3.11 Exercises........................................................ 89
      3.11.1 Declaring and printing variables..................... 89
      3.11.2 Getting the average of three numbers................ 89
      3.11.3 Output greatest value.................................. 89
      3.11.4 Operator precedence.................................... 89
4 Getting Input from the Keyboard........................................... 90
   4.1 Objectives........................................................ 90
   4.2 Using BufferedReader to get input....................... 90
   4.3 Using JOptionPane to get input............................ 94
10.2 Inheritance ................................................................. 172
  10.2.1 Defining Superclasses and Subclasses ....................... 173
  10.2.2 The super keyword ............................................. 175
  10.2.3 Overriding Methods ........................................... 176
  10.2.4 Final Methods and Final Classes .............................. 177
10.3 Polymorphism ............................................................... 178
10.4 Abstract Classes .......................................................... 180
10.5 Interfaces ....................................................................... 182
  10.5.1 Why do we use Interfaces? .................................... 182
  10.5.2 Interface vs. Abstract Class ................................. 182
  10.5.3 Interface vs. Class ............................................. 183
  10.5.4 Creating Interfaces ............................................. 183
  10.5.5 Relationship of an Interface to a Class ..................... 185
  10.5.6 Inheritance among Interfaces ................................. 185
10.6 Exercises ................................................................. 186
  10.6.1 Extending StudentRecord .................................... 186
  10.6.2 The Shape abstract class .................................... 186
11 Basic Exception Handling .................................................. 187
  11.1 Objectives ............................................................. 187
  11.2 What are Exceptions? .............................................. 187
  11.3 Handling Exceptions ............................................... 187
  11.4 Exercises ............................................................ 190
    11.4.1 Catching Exceptions1 ....................................... 190
    11.4.2 Catching Exceptions2 ....................................... 190
Appendix A: Java and NetBeans Installation ............................. 191
  Installing Java in Ubuntu Gutsy ..................................... 192
  Installing Java in Windows ........................................... 197
  Installing NetBeans in Ubuntu Gutsy ............................... 202
  Installing NetBeans in Windows ..................................... 209
Appendix B: Getting to know your Programming Environment (Windows XP version) ........................................ 215
  My First Java Program .................................................. 215
  Using a Text Editor and Console .................................... 216
    Setting the Path ....................................................... 228
    Using NetBeans ....................................................... 229
Appendix C: Answers to Exercises ........................................... 239
Chapter 1 Exercises .......................................................... 239
  1.1 Writing Algorithms ................................................ 239
  1.2 Number Conversions .............................................. 242
Chapter 2 (No exercises) .................................................... 245
Chapter 3 Exercises .......................................................... 246
  3.1 Hello World! .......................................................... 246
  3.2 The Tree .............................................................. 246
Chapter 4 Exercises .......................................................... 247
  4.1 Declaring and printing variables ................................ 247
  4.2 Getting the average of three numbers ......................... 247
  4.3 Output greatest value ............................................. 248
  4.4 Operator precedence ............................................. 248
Chapter 5 Exercises .......................................................... 249
  5.1 Last 3 words (BufferedReader version) ......................... 249
  5.2 Last 3 words (JOptionPane version) ............................. 250
Chapter 6 Exercises .......................................................... 251
  6.1 Grades ............................................................... 251
  6.2 Number in words .................................................. 253
6.3 Hundred Times................................................................. 255
6.4 Powers............................................................................. 258
Chapter 7 Exercises............................................................. 261
7.1 Days of the Week.............................................................. 261
7.2 Greatest number.............................................................. 262
Chapter 8 Exercises............................................................. 263
8.1 Print Arguments................................................................. 263
Chapter 9 Exercises................................................................ 264
9.1 Defining terms................................................................. 264
9.2 Java Scavenger Hunt......................................................... 264
Chapter 10 Exercises........................................................... 265
10.1 Address Book Entry....................................................... 265
10.2 AddressBook................................................................. 267
Chapter 11 Exercises........................................................... 272
11.1 Extending StudentRecord.............................................. 272
11.2 Abstract Classes............................................................ 275
Chapter 12 Exercises........................................................... 277
12.1 Catching Exceptions 1.................................................... 277
12.2 Catching Exceptions 2.................................................... 277
Appendix D : Machine Problems........................................... 280
Machine Problem 1: Phone Book.......................................... 280
Machine Problem 2: Minesweeper......................................... 281
Machine Problem 3: Number Conversion............................. 282
Appendix E : Hands-on Laboratory....................................... 283
Note to the Teacher.............................................................. 283
Chapter 1 Hands-on............................................................. 283
Chapter 2 Hands-on............................................................. 283
Chapter 3 Hands-on............................................................. 284
3.1 Things to check before you start the lab.......................... 284
3.2 Write, Compile, and Run Hello Java Program.................. 285
3.3 Write, Compile, and Run Hello Java Program using NetBeans........................................................................... 286
Chapter 4 Hands-on............................................................. 287
4.1 Declaring, Initializing, Printing Variables....................... 287
4.2 Conditional Operator....................................................... 288
Chapter 5 Hands-on............................................................. 289
5.1 Getting Input From Keyboard via BufferedReader.......... 289
5.2 Getting Input From Keyboard via JOptionPane................. 290
Chapter 6 Hands-on............................................................. 291
6.1 For Loop......................................................................... 291
Chapter 7 Hands-on............................................................. 292
7.1 Arrays.............................................................................. 292
Chapter 8 Hands-on............................................................. 293
Chapter 9 Hands-on............................................................. 293
9.1 Pass-by-Value................................................................. 293
9.2 Pass-by-Reference........................................................... 294
9.3 Comparing Objects......................................................... 295
Chapter 10 Hands-on.......................................................... 296
10.1 Create your own class.................................................... 296
10.2 Overloading................................................................. 301
10.3 Packaging................................................................. 304
Chapter 11 Hands-on.......................................................... 310
11.1 Inheritance - Constructor.............................................. 310
11.2 Inheritance - Overriding............................................... 315
# Revision History

## For Version 1.4
April 2008

<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A and B, Chapter 3: Getting to know your programming environment</td>
<td>Switch to Netbeans 6.1 Beta Version</td>
</tr>
<tr>
<td>Appendix A, Chapter 3: Getting to know your programming environment</td>
<td>Switch from Ubuntu Dapper to Ubuntu Gutsy</td>
</tr>
</tbody>
</table>

## For Version 1.3
June 2006

<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A and B, Chapter 3: Getting to know your programming environment</td>
<td>Switch to Netbeans 5.5 Beta Version</td>
</tr>
<tr>
<td>Appendix A, Chapter 3: Getting to know your programming environment</td>
<td>Switch from Redhat Linux to Ubuntu Dapper</td>
</tr>
<tr>
<td>Appendix F: Additional Exercises</td>
<td>Added (Teacher's manual)-c/o JEDI member school teachers</td>
</tr>
</tbody>
</table>

## For Version 1.2
January 2006

<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Number</td>
<td>Change from 1.1 to 1.2</td>
</tr>
<tr>
<td>Chapter 3: Getting to know your programming environment Appendix A</td>
<td>Change Netbeans/netbeans to <strong>NetBeans</strong></td>
</tr>
<tr>
<td>Chapter 4: Programming Fundamentals</td>
<td>List of Java keywords</td>
</tr>
<tr>
<td>Chapter 10: Creating your own classes</td>
<td>Coding guidelines: filenames should have the same name as the <strong>public</strong> class name</td>
</tr>
<tr>
<td>Master Documents</td>
<td>Added to list of references</td>
</tr>
</tbody>
</table>

## For Version 1.1
August 2005

<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Number</td>
<td>Change from 1.0 to 1.1</td>
</tr>
<tr>
<td>Revision History</td>
<td>Added</td>
</tr>
<tr>
<td>Section</td>
<td>Details</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Appendix E: Hands-on Lab Exercises</td>
<td>Added (c/o Sang)</td>
</tr>
<tr>
<td>Chapter 10: Creating Your own classes</td>
<td>Added subsection on How to set classpath at packages section</td>
</tr>
</tbody>
</table>
| Chapter 11: Inheritance, Interfaces and Polymorphism | Polymorphism section  
  • Added example that uses another class whose method can receive a reference variable |
|                                             | Interface                                                              |
|                                             |  • Added sections                                                      |
|                                             |   • Why do we use Interfaces?                                          |
|                                             |   • Interface vs. Abstract Class                                       |
|                                             |   • Interface vs. Class                                                |
|                                             |   • Relationship of an Interface to a Class                            |
|                                             |   • Inheritance among Interfaces                                       |
1 Introduction to Computer Programming

1.1 Objectives

In this section, we will be discussing the basic components of a computer, both hardware and software. We will also be giving a brief overview of programming languages and the program development life cycle. Finally, different number systems and conversions from one type to another will be discussed.

At the end of the lesson, the student should be able to:

- Identify the different components of a computer
- Know about programming languages and their categories
- Understand the program development life cycle and apply it in problem solving
- Learn the different number systems and their conversions

1.2 Introduction

A computer is a machine that performs a variety of tasks according to specific instructions. It is a data processing machine which accepts data via an input device and its processor manipulates the data according to a program.

The computer has two major components. The first one is the Hardware which is the tangible part of the computer. It is composed of electronic and mechanical parts.

The second major component is the software which is the intangible part of a computer. It consists of data and the computer programs.
1.3 Basic Components of a Computer

1.3.1 Hardware

1.3.1.1 The Central Processing Unit
The processor is the “brain” of the computer. It contains millions of extremely tiny electrical parts. It does the fundamental computing within the system. Examples of processors are Pentium, Athlon and SPARC.

1.3.1.2 Memory
The memory is where data and instructions needed by the CPU to do its appointed tasks can be found. It is divided into several storage locations which have corresponding addresses. The CPU accesses the memory with the use of these addresses.

1. Main Memory
The main memory is very closely connected to the processor. It is used to hold programs and data, that the processor is actively working with. It is not used for long-term storage. It is sometimes called the RAM (Random Access Memory).

The computer's main memory is considered as volatile storage. This means that once the computer is turned off, all information residing in the main memory is erased.

2. The Secondary Memory
The secondary memory is connected to main memory. It is used to hold programs and data for long term use. Examples of secondary memory are hard disks and cd-rom.

Secondary memory is considered as non-volatile storage. This means that information residing in secondary memory is not erased after the computer is turned off.

<table>
<thead>
<tr>
<th>Main Memory</th>
<th>Secondary Memory</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>Slow</td>
<td>Speed</td>
</tr>
<tr>
<td>Expensive</td>
<td>Cheap</td>
<td>Price</td>
</tr>
<tr>
<td>Low</td>
<td>High</td>
<td>Capacity</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Volatile</td>
</tr>
</tbody>
</table>

Table 1: Comparison between main memory and secondary memory
1.3.1.3 Input and Output Devices

Input and output devices allow a computer system to interact with the outside world by moving data into and out of the system.

Examples of input devices are keyboards, mice and microphones. Examples of output devices are monitors, printers and speakers.

1.3.2 Software

A software is the program that a computer uses in order to function. It is kept on some hardware device like a hard disk, but it itself is intangible. The data that the computer uses can be anything that a program needs. Programs acts like instructions for the processor.

Some Types of Computer Programs:

1. Systems Programs

   - Programs that are needed to keep all the hardware and software systems running together smoothly
   - Examples:
     - Operating Systems like Linux, Windows, Unix, Solaris, MacOS

2. Application Programs

   - Programs that people use to get their work done
   - Examples:
     - Word Processor
     - Game programs
     - Spreadsheets

3. Compilers

   - The computer understands only one language: machine language. Machine language is in the form of ones and zeros. Since it is highly impractical for people to create programs out of zeros and ones, there must be a way of translating or converting a language which we understand into machine language, for this purpose, there exists compilers.
1.4 Overview of Computer Programming Languages

1.4.1 What is a Programming Language?

A programming language is a standardized communication technique for expressing instructions to a computer. Like human languages, each language has its own syntax and grammar.

Programming languages enable a programmer to precisely specify what data a computer will act upon, how these data will be stored/transmitted, and precisely what actions to take under various circumstances.

There are different types of programming languages that can be used to create programs, but regardless of what language you use, these instructions are translated into machine language that can be understood by computers.

1.4.2 Categories of Programming Languages

1. High-level Programming Languages
   - A high-level programming language is a programming language that is more user-friendly, to some extent platform-independent, and abstract from low-level computer processor operations such as memory accesses. A programming statement may be translated into one or several machine instructions by a compiler.
     - Examples are Java, C, C++, Basic, Fortran

2. Low-level Assembly Language
   - Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers. Assembly languages are available for each CPU family, and each assembly instruction is translated into one machine instruction by an assembler program.

Note: The terms "high-level" and "low-level" are inherently relative. Originally, assembly language was considered low-level and COBOL, C, etc. were considered high-level. Many programmers today might refer to these latter languages as low-level.
1.5 The Program Development Life Cycle

Programmers do not sit down and start writing code right away when trying to make a computer program. Instead, they follow an organized plan or methodology, that breaks the process into a series of tasks.

Here are the basic steps in trying to solve a problem on the computer:

1. Problem Definition
2. Problem Analysis
3. Algorithm design and representation (Pseudocode or flowchart)
4. Coding and debugging

In order to understand the basic steps in solving a problem on a computer, let us define a single problem that we will solve step-by-step as we discuss the problem solving methodologies in detail. The problem we will solve will be defined in the next section.
1.5.1 Problem Definition

A programmer is usually given a task in the form of a problem. Before a program can be designed to solve a particular problem, the problem must be well and clearly defined first in terms of its input and output requirements.

A clearly defined problem is already half the solution. Computer programming requires us to define the problem first before we even try to create a solution.

Let us now define our example problem:

“Create a program that will determine the number of times a name occurs in a list.”

1.5.2 Problem Analysis

After the problem has been adequately defined, the simplest and yet the most efficient and effective approach to solve the problem must be formulated.

Usually, this step involves breaking up the problem into smaller and simpler sub-problems.

Example Problem:
Determine the number of times a name occurs in a list

Input to the program:
list of names, name to look for

Output of the program:
the number of times the name occurs in a list
1.5.3 Algorithm design and representation

Once our problem is clearly defined, we can now set to finding a solution. In computer programming, it is normally required to express our solution in a step-by-step manner.

An Algorithm is a clear and unambiguous specification of the steps needed to solve a problem. It may be expressed in either Human language (English, Tagalog), through a graphical representation like a flowchart or through a pseudocode, which is a cross between human language and a programming language.

Now given the problem defined in the previous sections, how do we express our general solution in such a way that it is simple yet understandable?

**Expressing our solution through Human language:**
1. Get the list of names
2. Get the name to look for, let's call this the keyname
3. Compare the keyname to each of the names in the list
4. If the keyname is the same with a name in the list, add 1 to the count
5. If all the names have been compared, output the result

**Expressing our solution through a flowchart:**

![Flowchart](image)

*Figure 1.1: Example of a flow chart*
Expressing our solution through pseudocode:

Let nameList = List of Names  
Let keyName = the name to be sought  
Let Count = 0  
For each name in NameList do the following  
if name == keyName  
Count = Count + 1  
Display Count

Figure 1.2: Example of a pseudocode

1.5.3.1 Flowcharting Symbols and their meanings

A flowchart is a design tool used to graphically represent the logic in a solution. Flowcharts typically do not display programming language commands. Rather, they state the concept in English or mathematical notation.

Here are some guidelines for commonly used symbols in creating flowcharts. You can use any symbols in creating your flowcharts, as long as you are consistent in using them.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Process Symbol</strong></td>
<td>Represents the process of executing a defined operation or groups of operations that results in a change in value, form, or location of information. Also functions as the default symbol when no other symbol is available.</td>
</tr>
<tr>
<td></td>
<td><strong>Input/Output (I/O) Symbol</strong></td>
<td>Represents an I/O function, which makes data available for processing (input) or displaying (output) of processed information.</td>
</tr>
<tr>
<td></td>
<td><strong>Flowline Symbol</strong></td>
<td>Represents the sequence of available information and executable operations. The lines connect other symbols, and the arrowheads are mandatory only for right-to-left and bottom-to-top flow. Represents the addition of descriptive information, comments, or explanatory notes as clarification. The vertical line and the broken line may be placed on the left, as shown, or on the right.</td>
</tr>
<tr>
<td></td>
<td><strong>Annotation Symbol</strong></td>
<td>Represents a decision that determines which of a number of alternative paths is to be followed.</td>
</tr>
<tr>
<td></td>
<td><strong>Decision Symbol</strong></td>
<td>Represents the beginning, the end, or a point of interruption or delay in a program.</td>
</tr>
</tbody>
</table>

Terminal Symbol
1.5.4 Coding and Debugging

After constructing the algorithm, it is now possible to create the source code. Using the algorithm as basis, the source code can now be written using the chosen programming language.

Most of the time, after the programmer has written the program, the program isn't 100% working right away. The programmer has to add some fixes to the program in case of errors (also called bugs) that occurs in the program. This process of is called debugging.

There are two types of errors that a programmer will encounter along the way. The first one is compile-time error, and the other is runtime error.

**Compile-Time Errors** occur if there is a syntax error in the code. The compiler will detect the error and the program won't even compile. At this point, the programmer is unable to form an executable that a user can run until the error is fixed.

Forgetting a semi-colon at the end of a statement or misspelling a certain command, for example, is a compile-time error. It's something the compiler can detect as an error.

Compilers aren't perfect and so can't catch all errors at compile time. This is especially true for logic errors such as infinite loops. This type of error is called **runtime error**.

For example, the actual syntax of the code looks okay. But when you follow the code's logic, the same piece of code keeps executing over and over again infinitely so that it loops. In such a case, compilers aren't really smart enough to catch all of these types of errors at compile-time, and therefore, the program compiles fine into an executable file. However, and unfortunately, when the end-user runs the program, the program (or even their whole computer) freezes up due to an infinite loop. Other types of run-time errors are when an incorrect value is computed, the wrong thing happens, etc.
### 1.6 Number Systems and Conversions

Numbers can be represented in a variety of ways. The representation depends on what is called the **BASE**. The following are the four most common representations.

#### 1.6.1 Decimal

We normally represent numbers in their decimal form. Numbers in decimal form are in base 10. This means that the only digits that appear are 0-9. Here are examples of numbers written in decimal form:

- \( 126_{10} \) (normally written as just 126)
- \( 11_{10} \) (normally written as just 11)

#### 1.6.2 Binary

Numbers in binary form are in base 2. This means that the only legal digits are 0 and 1. We need to write the subscript \( _2 \) to indicate that the number is a binary number. Here are examples of numbers written in binary form:

- \( 1111110_2 \)
- \( 1011_2 \)

#### 1.6.3 Octal

Numbers in octal form are in base 8. This means that the only legal digits are 0-7. We need to write the subscript \( _8 \) to indicate that the number is an octal number. Here are examples of numbers written in octal form:

- \( 176_8 \)
- \( 13_8 \)

#### 1.6.4 Hexadecimal

Numbers in hexadecimal form are in base 16. This means that the only legal digits are 0-9 and the letters A-F (or a-f, lowercase or uppercase does not matter). We need to write the subscript \( _{16} \) to indicate that the number is a hexadecimal number. Here are examples of numbers written in hexadecimal form:

- \( 7E_{16} \)
- \( B_{16} \)

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal Equivalent</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

*Table 3: Hexadecimal Numbers and their Equivalence to decimal numbers*

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 126_{10} )</td>
<td>( 1111110_2 )</td>
<td>( 176_8 )</td>
<td>( 7E_{16} )</td>
</tr>
<tr>
<td>( 11_{10} )</td>
<td>( 1011_2 )</td>
<td>( 13_8 )</td>
<td>( B_{16} )</td>
</tr>
</tbody>
</table>

*Table 4: Summary of Examples*
1.6.5 Conversions

1.6.5.1 Decimal to Binary / Binary to Decimal

To convert a decimal number to binary, continuously divide the number by 2 and get the remainder (which is either 0 or 1), and get that number as a digit of the binary form of the number. Get the quotient and divide that number again by 2 and repeat the whole process until the quotient reaches 0 or 1. We then get all the remainders starting from the last remainder, and the result is the binary form of the number.

NOTE: For the last digit which is already less than the divisor (which is 2) just copy the value to the remainder portion.

For Example:

\[126_{10} = ?_{2}\]

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>126 / 2 =</td>
<td>63</td>
</tr>
<tr>
<td>63 / 2 =</td>
<td>31</td>
</tr>
<tr>
<td>31 / 2 =</td>
<td>15</td>
</tr>
<tr>
<td>15 / 2 =</td>
<td>7</td>
</tr>
<tr>
<td>7 / 2 =</td>
<td>3</td>
</tr>
<tr>
<td>3 / 2 =</td>
<td>1</td>
</tr>
<tr>
<td>1 / 2 =</td>
<td>0</td>
</tr>
</tbody>
</table>

Write it this way

So, writing the remainders from the bottom up, we get the binary number \(1111110_2\).

To convert a binary number to decimal, we multiply the binary digit to \(2\) raised to the position of the binary number\(^\text{\textit{\footnotesize{\textup{\textit{}}}\text{\textit{\footnotesize{\textup{\textit{}}}}}}\text{\textit{\footnotesize{\textup{\textit{}}}}}}\). We then add all the products to get the resulting decimal number.

For Example:

\[1111110_2 = ?_{10}\]

<table>
<thead>
<tr>
<th>Position</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 x 2^0 =</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^1 =</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^2 =</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^3 =</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^4 =</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^5 =</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x 2^6 =</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.6.5.2 Decimal to Octal (or Hexadecimal)/Octal (or Hexadecimal) to Decimal

Converting decimal numbers to Octal or hexadecimal is basically the same as converting decimal to binary. However, instead of having 2 as the divisor, you replace it with 8 (for octal) or 16 (for hexadecimal).

For Example (Octal):

\[ 126_{10} = \_\_\_\_8 \]

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>126 / 8</td>
<td>15</td>
</tr>
<tr>
<td>15 / 8</td>
<td>1</td>
</tr>
<tr>
<td>1 / 8</td>
<td>1</td>
</tr>
</tbody>
</table>

Write it this way

So, writing the remainders from the bottom up, we get the octal number \(176_8\).

For Example (Hexadecimal):

\[ 126_{10} = \_\_\_\_16 \]

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>126 / 16</td>
<td>7</td>
</tr>
<tr>
<td>7 / 16</td>
<td>14 (equal to hex digit E)</td>
</tr>
</tbody>
</table>

Write it this way

So, writing the remainders from the bottom up, we get the hexadecimal number \(7E_{16}\).

* * *

Converting octal or hexadecimal numbers is also the same as converting binary numbers to decimal. To do that, we will just replace the base number 2 with 8 for Octal and 16 for hexadecimal.

For Example (Octal):

\[ 176_8 = \_\_\_\_10 \]

<table>
<thead>
<tr>
<th>Position</th>
<th>Octal Digits</th>
<th>Octal Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>6 \times 8^0 = 6</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7 \times 8^1 = 56</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 \times 8^2 = 64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOTAL: 126</td>
<td></td>
</tr>
</tbody>
</table>

Total: 126
For Example (Hexadecimal):
7E_{16} = ?_{10}

\[
\begin{array}{c|c}
\text{Position} & \text{Hex Digits} \\
1 & 7 \\
0 & E \\
\end{array}
\]

\[
\begin{align*}
14 \times 16^0 &= 14 \\
7 \times 16^1 &= 112 \\
\text{TOTAL:}& \ 126
\end{align*}
\]

1.6.5.3 Binary to Octal / Octal to Binary

To convert from binary numbers to octal, we partition the binary number into groups of 3 digits (from right to left), and pad it with zeros if the number of digits is not divisible by 3. We then convert each partition into its corresponding octal digit. The following is a table showing the binary representation of each octal digit.

<table>
<thead>
<tr>
<th>Octal Digit</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

Table 5: Octal Digits and their corresponding binary representation

For Example:
111110_{2} = ?_{8}

001 111 100
1 7 6

Equivalent octal number

Converting octal numbers to binary is just the opposite of what is given above. Simply convert each octal digit into its binary representation (given the table) and concatenate them. The result is the binary representation.
1.6.5.4 Binary to Hexadecimal / Hexadecimal to Binary

To convert from binary numbers to hexadecimal, we partition the binary number into groups of 4 digits (from right to left), and pad it with zeros if the number of digits is not divisible by 4. We then convert each partition into its corresponding hexadecimal digit. The following is a table showing the binary representation of each hexadecimal digit.

<table>
<thead>
<tr>
<th>Hexadecimal Digit</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

Table 6: Hexadecimal Digits and their corresponding binary representation

For Example:
1111110_2 = ?_16

Converting hexadecimal numbers to binary is just the opposite of what is given above. Simply convert each hexadecimal digit into its binary representation (given the table) and concatenate them. The result is the binary representation.
1.7 Exercises

1.7.1 Writing Algorithms
Given the following set of tasks, create an algorithm to accomplish the following tasks. You may write your algorithms using pseudocodes or you can use flowcharts.
1. Baking Bread
2. Logging into your laboratory's computer
3. Getting the average of three numbers

1.7.2 Number Conversions
Convert the following numbers:
1. 1980\textsubscript{10} to binary, hexadecimal and octal
2. 1001001101\textsubscript{2} to decimal, hexadecimal and octal
3. 76\textsubscript{8} to binary, hexadecimal and decimal
4. 43F\textsubscript{16} to binary, decimal and octal
Introduction to Java

1.8 Objectives

In this section, we will be discussing a little bit of Java history and what is Java Technology. We will also discuss the phases that a Java program undergoes.

At the end of the lesson, the student should be able to:

- Describe the features of Java technology such as the Java virtual machine, garbage collection and code security
- Describe the different phases of a Java program

1.9 Java Background

1.9.1 A little Bit of History

Java was created in 1991 by James Gosling et al. of Sun Microsystems. Initially called Oak, in honor of the tree outside Gosling’s window, its name was changed to Java because there was already a language called Oak.

The original motivation for Java was the need for platform independent language that could be embedded in various consumer electronic products like toasters and refrigerators. One of the first projects developed using Java was a personal hand-held remote control named Star 7.

At about the same time, the World Wide Web and the Internet were gaining popularity. Gosling et. al. realized that Java could be used for Internet programming.

1.9.2 What is Java Technology?

1.9.2.1 A programming language

As a programming language, Java can create all kinds of applications that you could create using any conventional programming language.

1.9.2.2 A development environment

As a development environment, Java technology provides you with a large suite of tools: a compiler, an interpreter, a documentation generator, a class file packaging tool, and so on.

1.9.2.3 An application environment

Java technology applications are typically general-purpose programs that run on any machine where the Java runtime environment (JRE) is installed.
1.9.2.4 A deployment environment

There are two main deployment environments: First, the JRE supplied by the Java 2 Software Development Kit (SDK) contains the complete set of class files for all the Java technology packages, which includes basic language classes, GUI component classes, and so on. The other main deployment environment is on your web browser. Most commercial browsers supply a Java technology interpreter and runtime environment.

1.9.3 Some Features of Java

1.9.3.1 The Java Virtual Machine

The Java Virtual Machine is an imaginary machine that is implemented by emulating software on a real machine. The JVM provides the hardware platform specifications to which you compile all Java technology code. This specification enables the Java software to be platform-independent because the compilation is done for a generic machine known as the JVM.

A bytecode is a special machine language that can be understood by the Java Virtual Machine (JVM). The bytecode is independent of any particular computer hardware, so any computer with a Java interpreter can execute the compiled Java program, no matter what type of computer the program was compiled on.

1.9.3.2 Garbage Collection

Many programming languages allows a programmer to allocate memory during runtime. However, after using that allocated memory, there should be a way to deallocate that memory block in order for other programs to use it again. In C, C++ and other languages the programmer is responsible for this. This can be difficult at times since there can be instances wherein the programmers forget to deallocate memory and therefor result to what we call memory leaks.

In Java, the programmer is freed from the burden of having to deallocate that memory themselves by having what we call the garbage collection thread. The garbage collection thread is responsible for freeing any memory that can be freed. This happens automatically during the lifetime of the Java program.
1.9.3.3 Code Security

Code security is attained in Java through the implementation of its **Java Runtime Environment (JRE)**. The JRE runs code compiled for a JVM and performs class loading (through the class loader), code verification (through the bytecode verifier) and finally code execution.

The **Class Loader** is responsible for loading all classes needed for the Java program. It adds security by separating the namespaces for the classes of the local file system from those that are imported from network sources. This limits any Trojan horse applications since local classes are always loaded first. After loading all the classes, the memory layout of the executable is then determined. This adds protection against unauthorized access to restricted areas of the code since the memory layout is determined during runtime.

After loading the class andlayouting of memory, the **bytecode verifier** then tests the format of the code fragments and checks the code fragments for illegal code that can violate access rights to objects.

After all of these have been done, the code is then finally executed.
1.9.4 Phases of a Java Program

The following figure describes the process of compiling and executing a Java program.

The first step in creating a Java program is by writing your programs in a text editor. Examples of text editors you can use are notepad, vi, emacs, etc. This file is stored in a disk file with the extension `.java`.

After creating and saving your Java program, compile the program by using the Java Compiler. The output of this process is a file of Java bytecodes with the file extension `.class`.

The `.class` file is then interpreted by the Java interpreter that converts the bytecodes into the machine language of the particular computer you are using.

<table>
<thead>
<tr>
<th>Task</th>
<th>Tool to use</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write the program</td>
<td>Any text editor</td>
<td>File with <code>.java</code> extension</td>
</tr>
<tr>
<td>Compile the program</td>
<td>Java Compiler</td>
<td>File with <code>.class</code> extension (Java bytecodes)</td>
</tr>
<tr>
<td>Run the program</td>
<td>Java Interpreter</td>
<td>Program Output</td>
</tr>
</tbody>
</table>

*Table 7: Summary of Phases of a Java Program*
2 Getting to know your Programming Environment

2.1 Objectives

In this section, we will be discussing on how to write, compile and run Java programs. There are two ways of doing this, the first one is by using a console and a text editor. The second one is by using NetBeans which is an Integrated Development Environment or IDE.

At the end of the lesson, the student should be able to:

- Create a Java program using text editor and console in the Linux (Ubuntu Dapper) environment
- Differentiate between syntax-errors and runtime errors
- Create a Java program using NetBeans

2.2 Introduction

An IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger.

This tutorial uses Ubuntu Dapper as the operating system. Make sure that before you do this tutorial, you have installed Java and NetBeans in your system. For instructions on how to install Java and NetBeans, please refer to Appendix A. For the Windows XP version of this section, please refer to Appendix B.

Before going into details, let us first take a look at the first Java program you will be writing.

2.3 My First Java Program

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

Before we try to explain what the program means, let's first try to write this program in a file and try to run it.
2.4 Using a Text Editor and Console

For this example, we will be using a text editor to edit the Java source code. You will also need to open the Terminal window to compile and execute your Java programs.

**Step 1: Start the Text Editor**
To start the Text Editor in Linux, click on Applications->Accessories->Text Editor.

![Image of Text Editor Application in Linux](image1)

*Figure 2.1: Text Editor Application in Linux*

**Step 2: Open Terminal**
To open Terminal in Linux, click on Applications-> Accessories-> Terminal.

![Image of Terminal in Linux](image2)

*Figure 2.2: Terminal in Linux*
Step 3: Write your the source code of your Java program in the Text Editor

```java
public class Hello {
    /*
    * My first java program
    */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

Figure 2.3: Writing the Source Code with the Text Editor
**Step 4: Save your Java Program**

We will save our program on a file named "Hello.java", and we will be saving it inside a folder named `MYJAVAPROGRAMS`.

To open the **Save** dialog box, click on the File menu found on the menubar and then click on Save.

After doing the procedure described above, a dialog box will appear as shown in Figure below.

![Save As Dialog](image)

*Figure 2.4: Save As Dialog*
Click on the browse button, and then click on the Create Folder button.
Name the new folder MYJAPROGRAMS. Now, click on the MYJAPROGRAMS folder in order to get inside that folder. You will see a similar figure as shown below after you clicked on MYJAPROGRAMS. The folder should be empty for now since it's a newly created folder and we haven't saved anything in it yet.

Now, in the Selection textbox, type in the filename of your program, which is "Hello.java", and then click on the SAVE button.
Now that you've saved your file, notice how the title of the frame changes from "Untitled Document 1 (modified) – gedit" to "Hello.java (~/MYJAVAPROGRAMS) - gedit". Take note that if you want to make changes in your file, you can just edit it, and then save it again by clicking on File -> Save.

![Figure 2.5: New Window After Saving](image)

J.E.D.I

Introduction to Programming I

37
Step 5: Compiling your program
Now, the next step is to compile your program. Go to the Terminal window we just opened a while ago.

Typically, when you open the terminal window, it opens up and takes you directly to what is called your home folder. To see what is inside that home folder, type `ls` and then press ENTER. What you will see is a list of files and folders inside your home folder.

Now, you can see here that there is a folder named "MYJAVAPROGRAMS" which we have created a while ago, and where we saved our Hello.java program. Now let's go inside that directory.

To go inside a directory, you type in the command: `cd [directory name]`. The "cd" command stands for, change directory. In this case, since the name of our directory is MYJAVAPROGRAMS, you type in: `cd MYJAVAPROGRAMS`

![Figure 2.6: Changing the Directory](image)
Once inside the folder where your Java programs are, let us now start compiling your Java program. Take note that, you should make sure that the file is inside the folder where you are in. In order to do that, execute the "ls" command again to see if your file is inside that folder.

```
$ cd ~/MYJAVAPROGRAMS/
$ ls
Hello.java
```

*Figure 2.7: List of Files Inside the New Directory*
To compile a Java program, we type in the command: `javac [filename]`. So in this case, type in: `javac Hello.java`.

During compilation, `javac` adds a file to the disk called `[filename].class`, or in this case, `Hello.class`, which is the actual bytecode.
Step 6: Running the Program

Now, assuming that there are no problems during compilation (we'll explore more of the problems encountered during compilation in the next section), we are now ready to run your program.

To run your Java program, type in the command: `java [filename without the extension]`, so in the case of our example, type in: `java Hello`

You can see on the screen that you have just run your first Java program that prints the message, "Hello world!".

![Figure 2.9: Running Class File](image)
2.4.1 Errors

What we've shown so far is a Java program wherein we didn't encounter any problems in compiling and running. However, this is not always the case. As what we have discussed in the first part of this course, we usually encounter errors along the way.

As discussed before, there are two types of errors. The first one is a compile-time error or also called as syntax error. The second one is the runtime error.

2.4.1.1 Syntax Errors

Syntax errors are usually typing errors. You may have misspelled a command in Java or forgot to write a semi-colon at the end of a statement. Java attempts to isolate the error by displaying the line of code and pointing to the first incorrect character in that line. However, the problem may not be at the exact point.

Other common mistakes are in capitalization, spelling, the use of incorrect special characters, and omission of correct punctuation.

Let's take for example, our Hello.java program wherein we intentionally omit the semicolon at one statement and we try to type the incorrect spelling of a command.

```
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        /*prints the string “Hello world” on screen
        System.out.println(“Hello world! ”)
    }
}
```

*Figure 2.10: Source Code With Errors*
See the error messages generated after compiling the program. The first error message suggests that there is an error in line 6 of your program. It pointed to the next word after the `static`, which should be spelled as static.

The second error message suggests that there is a missing semicolon after your statement.

As a rule of thumb, if you encounter a lot of error messages, try to correct the first mistake in a long list, and try to compile the program again. Doing so may reduce the total number of errors dramatically.

### 2.4.1.2 Run-time Errors

Run-time errors are errors that will not display until you run or execute your program. Even programs that compile successfully may display wrong answers if the programmer has not thought through the logical processes and structures of the program.
2.5 Using NetBeans

Now that we’ve tried doing our programs the complicated way, let's now see how to do all the processes we’ve described in the previous sections by using just one application.

In this part of the lesson, we will be using NetBeans, which is an Integrated Development Environment or IDE. An IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger.

**Step 1: Run NetBeans**

To Run NetBeans, is by clicking on the shortcut icon found on your Desktop.

![Figure 2.12: Running NetBeans using shortcut icon on desktop](image)
After you've open NetBeans IDE, you will see a graphical user interface (GUI) similar to what is shown below.

![Figure 2.13: Window After Opening NetBeans](image)

*Figure 2.13: Window After Opening NetBeans*
**Step 2: Make a project**
Now, let's first make a project. Click on File-> New Project. After doing this, a New Project dialog will appear. Now click on Java Application and click on the NEXT button.
Now, a New Application dialog will appear.

![New Java Application dialog](image)

**Figure 2.15: Setting the Project Information**
Now try to change the Application Location, by clicking on the BROWSE button. A Project Location dialog will then appear. Double-click on your home folder.

The contents of the root folder is then displayed. Now double-click on the MYJAVAPROGRAMS folder and click on the OPEN button.
See now that the Project Location and Project Folder is changed to 
/home/<user>/MYJAVAPROGRAMS.

Finally, on the Create Main Class textfield, type in Hello as the main class' name, and then click on the FINISH button.

Figure 2.17: Window after Setting the Project Location to MYJAVAPROGRAMS/Setting the Main Class of the Project to Hello
**Step 3: Type in your program**

Before typing in your program, let us first describe the main window after creating the project.

As shown below, NetBeans automatically creates the basic code for your Java program. You can just add your own statements to the generated code. On the left side of the window, you can see a list of folders and files that NetBeans generated after creating the project. This can all be found in your MYJAVAPROGRAMS folder, where you set the Project location.

![Figure 2.18: View of the Created Project](image-url)
Now, try to modify the code generated by NetBeans. Ignore the other parts of the program for now, as we will explain the details of the code later. Insert the code:

```java
System.out.println("Hello world!");
```

after the statement, //TODO code application logic here.

Figure 2.19: Inserting the Code
**Step 4: Compile your program**

Now, to compile your program, just click on Build -> Build Main Project. Or, you could also use the shortcut button to compile your code.
If there are no errors in your program, you will see a build successful message on the output window.

![Figure 2.20: View after a Successful Compilation](image)

**Figure 2.20: View after a Successful Compilation**
Step 5: Run your program
To run your program, click on Run-> Run Main Project. Or you could also use the shortcut button to run your program.

Figure 2.21: Running with NetBeans
The output of your program is displayed in the output window.

![Figure 2.22: View after a Successful Run](image)

```java
public class Hello {
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Hello World!");
    }
}
```
2.6 Exercises

2.6.1 Hello World!
Using NetBeans, create a class named: [YourName]. The program should output on the screen:

Welcome to Java Programming [YourName]!!!

2.6.2 The Tree
Using NetBeans, create a class named: TheTree. The program should output the following lines on the screen:

I think that I shall never see,
a poem as lovely as a tree.
A tree whose hungry mouth is pressed
Against the Earth’s sweet flowing breast.
3 Programming Fundamentals

3.1 Objectives

In this section, we will be discussing the basic parts of a Java program. We will start by trying to explain the basic parts of the Hello.java program introduced in the previous section. We will also be discussing some coding guidelines or code conventions along the way to help in effectively writing readable programs.

At the end of the lesson, the student should be able to:

• Identify the basic parts of a Java program
• Differentiate among Java literals, primitive data types, variable types, identifiers and operators
• Develop a simple valid Java program using the concepts learned in this chapter

3.2 Dissecting my first Java program

Now, we'll try to dissect your first Java program:

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

The first line of the code,

```java
public class Hello
```

indicates the name of the class which is `Hello`. In Java, all code should be placed inside a class declaration. We do this by using the `class` keyword. In addition, the class uses an access specifier `public`, which indicates that our class is accessible to other classes from other packages (packages are a collection of classes). We will be covering packages and access specifiers later.

The next line which contains a curly brace `{` indicates the start of a block. In this code, we placed the curly brace at the next line after the class declaration, however, we can also place this next to the first line of our code. So, we could actually write our code as:

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

or

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```
The next three lines indicates a Java comment. A comment is something used to
document a part of a code. It is not part of the program itself, but used for
documentation purposes. It is good programming practice to add comments to your
code.

    /**
     * My first java program
     */

A comment is indicated by the delimiters "/*" and "/". Anything within these delimiters
are ignored by the Java compiler, and are treated as comments.
The next line,

    public static void main(String[] args) {

or can also be written as,

    public static void main(String[] args) {

indicates the name of one method in Hello which is the main method. The main method
is the starting point of a Java program. All programs except Applets written in Java start
with the main method. Make sure to follow the exact signature.

The next line is also a Java comment,

    //prints the string "Hello world" on screen

Now, we learned two ways of creating comments. The first one is by placing the
comment inside /* and */, and the other one is by writing // at the start of the
comment.

The next line,

    System.out.println("Hello world!");

prints the text “Hello World!” on screen. The command System.out.println(), prints the
text enclosed by quotation on the screen.

The last two lines which contains the two curly braces is used to close the main method
and class respectively.

Coding Guidelines:
1. Your Java programs should always end with the .java extension.
2. Filenames should match the name of your public class. So for example, if the name
   of your public class is Hello, you should save it in a file called Hello.java.
3. You should write comments in your code explaining what a certain class does, or
   what a certain method do.
3.3 **Java Comments**

Comments are notes written to a code for documentation purposes. Those text are not part of the program and does not affect the flow of the program.

Java supports three types of comments: C++-style single line comments, C-style multiline comments and special javadoc comments.

### 3.3.1 C++-Style Comments

C++ Style comments starts with `//`. All the text after `//` are treated as comments. For example,

```
// This is a C++ style or single line comments
```

### 3.3.2 C-Style Comments

C-style comments or also called multiline comments starts with a `/*` and ends with a `*/`. All text in between the two delimiters are treated as comments. Unlike C++ style comments, it can span multiple lines. For example,

```
/* this is an example of a
   C style or multiline comments */
```

### 3.3.3 Special Javadoc Comments

Special Javadoc comments are used for generating an HTML documentation for your Java programs. You can create javadoc comments by starting the line with `/**` and ending it with `*/`. Like C-style comments, it can also span lines. It can also contain certain tags to add more information to your comments. For example,

```
/**
   * This is an example of special java doc comments used for generating an html documentation. It uses tags like:
   * @author Florence Balagtas
   * @version 1.2
   */
```
3.4 Java Statements and blocks

A statement is one or more lines of code terminated by a semicolon. An example of a single statement is,

```
System.out.println("Hello world");
```

A block is one or more statements bounded by an opening and closing curly braces that groups the statements as one unit. Block statements can be nested indefinitely. Any amount of white space is allowed. An example of a block is,

```
public static void main( String[] args ){
    System.out.println("Hello");
    System.out.println("world");
}
```

**Coding Guidelines:**

1. In creating blocks, you can place the opening curly brace in line with the statement, like for example,

```
public static void main( String[] args ){
```

or you can place the curly brace on the next line, like,

```
public static void main( String[] args )
{
```

2. You should indent the next statements after the start of a block, for example,

```
public static void main( String[] args ){
    System.out.println("Hello");
    System.out.println("world");
}
```
3.5 Java Identifiers

Identifiers are tokens that represent names of variables, methods, classes, etc. Examples of identifiers are: Hello, main, System, out.

Java identifiers are case-sensitive. This means that the identifier: Hello is not the same as hello. Identifiers must begin with either a letter, an underscore “_”, or a dollar sign “$”. Letters may be lower or upper case. Subsequent characters may use numbers 0 to 9.

Identifiers cannot use Java keywords like class, public, void, etc. We will discuss more about Java keywords later.

**Coding Guidelines:**

1. For names of classes, capitalize the first letter of the class name. For names of methods and variables, the first letter of the word should start with a small letter. For example:
   
   ```
   ThisIsAnExampleOfClassName
   thisIsAnExampleOfMethodName
   ```

2. In case of multi-word identifiers, use capital letters to indicate the start of the word except the first word. For example, charArray, fileNumber, ClassName.

3. Avoid using underscores at the start of the identifier such as _read or _write.
3.6 Java Keywords

Keywords are predefined identifiers reserved by Java for a specific purpose. You cannot use keywords as names for your variables, classes, methods ...etc. Here is a list of the Java Keywords.

We will try to discuss all the meanings of these keywords and how they are used in our Java programs as we go along the way.

**Note:** true, false, and null are not keywords but they are reserved words, so you cannot use them as names in your programs either.
3.7 Java Literals

Literals are tokens that do not change or are constant. The different types of literals in Java are: Integer Literals, Floating-Point Literals, Boolean Literals, Character Literals and String Literals.

3.7.1 Integer Literals

Integer literals come in different formats: decimal (base 10), hexadecimal (base 16), and octal (base 8). In using integer literals in our program, we have to follow some special notations.

For decimal numbers, we have no special notations. We just write a decimal number as it is. For hexadecimal numbers, it should be preceded by “0x” or “0X”. For octals, they are preceeded by “0”.

For example, consider the number 12. Its decimal representation is 12, while in hexadecimal, it is 0xC, and in octal, it is equivalent to 014.

Integer literals default to the data type int. An int is a signed 32-bit value. In some cases, you may wish to force integer literal to the data type long by appending the “l” or “L” character. A long is a signed 64-bit value. We will cover more on data types later.

3.7.2 Floating-Point Literals

Floating point literals represent decimals with fractional parts. An example is 3.1415. Floating point literals can be expressed in standard or scientific notations. For example, 583.45 is in standard notation, while 5.8345e2 is in scientific notation.

Floating point literals default to the data type double which is a 64-bit value. To use a smaller precision (32-bit) float, just append the “f” or “F” character.

3.7.3 Boolean Literals

Boolean literals have only two values, true or false.
3.7.4 Character Literals

Character Literals represent single Unicode characters. A Unicode character is a 16-bit character set that replaces the 8-bit ASCII character set. Unicode allows the inclusion of symbols and special characters from other languages.

To use a character literal, enclose the character in single quote delimiters. For example, the letter a, is represented as ‘a’.

To use special characters such as a newline character, a backslash is used followed by the character code. For example, ‘\n’ for the newline character, ‘\r’ for the carriage return, ‘\b’ for backspace.

3.7.5 String Literals

String literals represent multiple characters and are enclosed by double quotes. An example of a string literal is, “Hello World”.
3.8 Primitive data types

The Java programming language defines eight primitive data types. The following are, boolean (for logical), char (for textual), byte, short, int, long (integral), double and float (floating point).

3.8.1 Logical - boolean
A boolean data type represents two states: true and false. An example is,

```java
boolean result = true;
```

The example shown above, declares a variable named result as boolean type and assigns it a value of true.

3.8.2 Textual – char
A character data type (char), represents a single Unicode character. It must have its literal enclosed in single quotes(‘ ’). For example,

```java
'a'  //The letter a
'	'  //A tab
```

To represent special characters like ' (single quotes) or " (double quotes), use the escape character \. For example,

```java
'\'  //for single quotes
'"'  //for double quotes
```

Although, String is not a primitive data type (it is a Class), we will just introduce String in this section. A String represents a data type that contains multiple characters. It is not a primitive data type, it is a class. It has it’s literal enclosed in double quotes(“”).

For example,

```java
String message="Hello world!"
```
3.8.3 Integral – byte, short, int & long

Integral data types in Java uses three forms – decimal, octal or hexadecimal. Examples are,

2    //The decimal value 2
077  //The leading 0 indicates an octal value
0xBACC  //The leading 0x indicates a hexadecimal value

Integral types has int as default data type. You can define its long value by appending the letter l or L. Integral data type have the following ranges:

<table>
<thead>
<tr>
<th>Integer Length</th>
<th>Name or Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>byte</td>
<td>-2^7 to 2^7-1</td>
</tr>
<tr>
<td>16 bits</td>
<td>short</td>
<td>-2^15 to 2^15-1</td>
</tr>
<tr>
<td>32 bits</td>
<td>int</td>
<td>-2^31 to 2^31-1</td>
</tr>
<tr>
<td>64 bits</td>
<td>long</td>
<td>-2^63 to 2^63-1</td>
</tr>
</tbody>
</table>

Table 8: Integral types and their ranges

**Coding Guidelines:**

In defining a long value, a lowercase L is not recommended because it is hard to distinguish from the digit 1.
3.8.4 Floating Point – float and double

Floating point types has `double` as default data type. Floating-point literal includes either a decimal point or one of the following,

- E or e //(add exponential value)
- F or f //(float)
- D or d //(double)

Examples are,

- 3.14 //A simple floating-point value (a double)
- 6.02E23 //A large floating-point value
- 2.718F //A simple float size value
- 123.4E+306D //A large double value with redundant D

In the example shown above, the 23 after the E in the second example is implicitly positive. That example is equivalent to 6.02E+23. Floating-point data types have the following ranges:

<table>
<thead>
<tr>
<th>Float Length</th>
<th>Name or Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bits</td>
<td>float</td>
<td>$-2^{31}$ to $2^{31} - 1$</td>
</tr>
<tr>
<td>64 bits</td>
<td>double</td>
<td>$-2^{63}$ to $2^{63} - 1$</td>
</tr>
</tbody>
</table>

*Table 9: Floating point types and their ranges*
3.9 Variables
A variable is an item of data used to store state of objects.

A variable has a data type and a name. The data type indicates the type of value that the variable can hold. The variable name must follow rules for identifiers.

3.9.1 Declaring and Initializing Variables
To declare a variable is as follows,

<data type> <name> [=initial value];

Note: Values enclosed in <> are required values, while those values enclosed in [] are optional.

Here is a sample program that declares and initializes some variables,

```java
public class VariableSamples {
    public static void main( String[] args ){
        // declare a data type with variable name
        // result and boolean data type
        boolean result;
        // declare a data type with variable name
        // option and char data type
        char option;
        option = 'C'; // assign 'C' to option
        // declare a data type with variable name
        // grade, double data type and initialized
        // to 0.0
        double grade = 0.0;
    }
}
```

Coding Guidelines:
1. It always good to initialize your variables as you declare them.
2. Use descriptive names for your variables. Like for example, if you want to have a variable that contains a grade for a student, name it as, grade and not just some random letters you choose.
3. Declare one variable per line of code. For example, the variable declarations,
   ```java
double exam=0;
double quiz=10;
double grade = 0;
```
   is preferred over the declaration,
   ```java
double exam=0, quiz=10, grade=0;
```
3.9.2 Outputting Variable Data

In order to output the value of a certain variable, we can use the following commands,

```java
System.out.println()
System.out.print()
```

Here's a sample program,

```java
public class OutputVariable {
    public static void main(String[] args) {
        int value = 10;
        char x;
        x = 'A';

        System.out.println(value);
        System.out.println("The value of x=\" + x + ");
    }
}
```

The program will output the following text on screen,

```
10
The value of x=A
```

3.9.3 System.out.println() vs. System.out.print()

What is the difference between the commands `System.out.println()` and `System.out.print()`? The first one appends a newline at the end of the data to output, while the latter doesn't.

Consider the statements,

```java
System.out.print("Hello ");
System.out.print("world!" );
```

These statements will output the following on the screen,

```
Hello world!
```

Now consider the following statements,

```java
System.out.println("Hello ");
System.out.println("world!" );
```

These statements will output the following on the screen,

```
Hello
world!
```
3.9.4 Reference Variables vs. Primitive Variables

We will now differentiate the two types of variables that Java programs have. These are reference variables and primitive variables.

**Primitive variables** are variables with primitive data types. They store data in the actual memory location of where the variable is.

**Reference variables** are variables that stores the address in the memory location. It points to another memory location of where the actual data is. When you declare a variable of a certain class, you are actually declaring a reference variable to the object with that certain class.

For example, suppose we have two variables with data types int and String.

```java
int num = 10;
String name = "Hello"
```

Suppose, the illustration shown below is the actual memory of your computer, wherein you have the address of the memory cells, the variable name and the data they hold.

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Variable Name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>num</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1563</td>
<td>name</td>
<td>Address(2000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td>&quot;Hello&quot;</td>
</tr>
</tbody>
</table>

As you can see, for the primitive variable num, the data is on the actual location of where the variable is. For the reference variable name, the variable just holds the address of where the actual data is.
### 3.10 Operators

In Java, there are different types of operators. There are arithmetic operators, relational operators, logical operators and conditional operators. These operators follow a certain kind of precedence so that the compiler will know which operator to evaluate first in case multiple operators are used in one statement.

#### 3.10.1 Arithmetic operators

Here are the basic arithmetic operators that can be used in creating your Java programs,

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>op1 + op2</td>
<td>Adds op1 and op2</td>
</tr>
<tr>
<td>*</td>
<td>op1 * op2</td>
<td>Multiplies op1 by op2</td>
</tr>
<tr>
<td>/</td>
<td>op1 / op2</td>
<td>Divides op1 by op2</td>
</tr>
<tr>
<td>%</td>
<td>op1 % op2</td>
<td>Computes the remainder of dividing op1 by op2</td>
</tr>
<tr>
<td>-</td>
<td>op1 - op2</td>
<td>Subtracts op2 from op1</td>
</tr>
</tbody>
</table>

*Table 10: Arithmetic operations and their functions*
Here's a sample program in the usage of these operators:

```java
public class ArithmeticDemo {
    public static void main(String[] args) {

        // a few numbers
        int i = 37;
        int j = 42;
        double x = 27.475;
        double y = 7.22;
        System.out.println("Variable values...");
        System.out.println(" i = " + i);
        System.out.println(" j = " + j);
        System.out.println(" x = " + x);
        System.out.println(" y = " + y); // adding numbers
        System.out.println("Adding...");
        System.out.println(" i + j = " + (i + j));
        System.out.println(" x + y = " + (x + y));

        // subtracting numbers
        System.out.println("Subtracting...");
        System.out.println(" i - j = " + (i - j));
        System.out.println(" x - y = " + (x - y));

        // multiplying numbers
        System.out.println("Multiplying...");
        System.out.println(" i * j = " + (i * j));
        System.out.println(" x * y = " + (x * y));

        // dividing numbers
        System.out.println("Dividing...");
        System.out.println(" i / j = " + (i / j));
        System.out.println(" x / y = " + (x / y));

        // computing the remainder resulting from dividing numbers
        System.out.println("Computing the remainder...");
        System.out.println(" i % j = " + (i % j));
        System.out.println(" x % y = " + (x % y));

        // mixing types
        System.out.println("Mixing types...");
        System.out.println(" j + y = " + (j + y));
        System.out.println(" i * x = " + (i * x));
    }
}
```
Here is the output of the program,

Variable values...
  i = 37
  j = 42
  x = 27.475
  y = 7.22
Adding...
  i + j = 79
  x + y = 34.695
Subtracting...
  i - j = -5
  x - y = 20.255
Multiplying...
  i * j = 1554
  x * y = 198.37
Dividing...
  i / j = 0
  x / y = 3.8054
Computing the remainder...
  i % j = 37
  x % y = 5.815
Mixing types...
  j + y = 49.22
  i * x = 1016.58

Note: When an integer and a floating-point number are used as operands to a single arithmetic operation, the result is a floating point. The integer is implicitly converted to a floating-point number before the operation takes place.
3.10.2 Increment and Decrement operators

Aside from the basic arithmetic operators, Java also includes a unary increment operator (++) and unary decrement operator (--). Increment and decrement operators increase and decrease a value stored in a number variable by 1.

For example, the expression,

\[
\text{count} = \text{count} + 1; \quad // \text{increment the value of count by 1}
\]

is equivalent to,

\[
\text{count}++; \
\]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>op++</td>
<td>Increments op by 1; evaluates to the value of op before it was incremented.</td>
</tr>
<tr>
<td>++</td>
<td>++op</td>
<td>Increments op by 1; evaluates to the value of op after it was incremented.</td>
</tr>
<tr>
<td>--</td>
<td>op--</td>
<td>Decrements op by 1; evaluates to the value of op before it was decremented.</td>
</tr>
<tr>
<td>--</td>
<td>--op</td>
<td>Decrements op by 1; evaluates to the value of op after it was decremented.</td>
</tr>
</tbody>
</table>

Table 11: Increment and Decrement operators

The increment and decrement operators can be placed before or after an operand. When used before an operand, it causes the variable to be incremented or decremented by 1, and then the new value is used in the expression in which it appears. For example,

\[
\begin{align*}
\text{int } i &= 10, \\
\text{int } j &= 3; \\
\text{int } k &= 0; \\
\text{k} &= ++j + i; \quad // \text{will result to } k = 4+10 = 14
\end{align*}
\]
When the increment and decrement operators are placed after the operand, the old value of the variable will be used in the expression where it appears. For example,

```c
int i = 10,
int j = 3;
int k = 0;

k = j++ + i; //will result to k = 3+10 = 13
```

**Coding Guideline:**

Always keep expressions containing increment and decrement operators simple and easy to understand.
3.10.3 Relational operators

Relational operators compare two values and determines the relationship between those values. The output of evaluation are the boolean values true or false.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>op1 &gt; op2</td>
<td>op1 is greater than op2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>op1 &gt;= op2</td>
<td>op1 is greater than or equal to op2</td>
</tr>
<tr>
<td>&lt;</td>
<td>op1 &lt; op2</td>
<td>op1 is less than op2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>op1 &lt;= op2</td>
<td>op1 is less than or equal to op2</td>
</tr>
<tr>
<td>==</td>
<td>op1 == op2</td>
<td>op1 and op2 are equal</td>
</tr>
<tr>
<td>!=</td>
<td>op1 != op2</td>
<td>op1 and op2 are not equal</td>
</tr>
</tbody>
</table>

Table 12: Relational Operators
Here's a sample program that uses relational operators,

```java
public class RelationalDemo {
    public static void main(String[] args) {
        // a few numbers
        int i = 37;
        int j = 42;
        int k = 42;
        System.out.println("Variable values... ");
        System.out.println(" i = " + i);
        System.out.println(" j = " + j);
        System.out.println(" k = " + k);

        // greater than
        System.out.println(" Greater than...");
        System.out.println(" i > j = " + (i > j)); // false
        System.out.println(" j > i = " + (j > i)); // true
        System.out.println(" k > j = " + (k > j)); // false

        // greater than or equal to
        System.out.println(" Greater than or equal to...");
        System.out.println(" i >= j = " + (i >= j)); // false
        System.out.println(" j >= i = " + (j >= i)); // true
        System.out.println(" k >= j = " + (k >= j)); // true

        // less than
        System.out.println(" Less than...");
        System.out.println(" i < j = " + (i < j)); // true
        System.out.println(" j < i = " + (j < i)); // false
        System.out.println(" k < j = " + (k < j)); // false

        // less than or equal to
        System.out.println(" Less than or equal to...");
        System.out.println(" i <= j = " + (i <= j)); // true
        System.out.println(" j <= i = " + (j <= i)); // false
        System.out.println(" k <= j = " + (k <= j)); // true

        // equal to
        System.out.println(" Equal to...");
        System.out.println(" i == j = " + (i == j)); // false
        System.out.println(" k == j = " + (k == j)); // true

        // not equal to
        System.out.println(" Not equal to...");
        System.out.println(" i != j = " + (i != j)); // true
        System.out.println(" k != j = " + (k != j)); // false
    }
}
```
Here’s the output from this program:

Variable values...
   i = 37
   j = 42
   k = 42
Greater than...
   i > j = false
   j > i = true
   k > j = false
Greater than or equal to...
   i >= j = false
   j >= i = true
   k >= j = true
Less than...
   i < j = true
   j < i = false
   k < j = false
Less than or equal to...
   i <= j = true
   j <= i = false
   k <= j = true
Equal to...
   i == j = false
   k == j = true
Not equal to...
   i != j = true
   k != j = true
3.10.4 Logical operators

Logical operators have one or two boolean operands that yield a boolean result. There are six logical operators: && (logical AND), & (boolean logical AND), || (logical OR), | (boolean logical inclusive OR), ^ (boolean logical exclusive OR), and ! (logical NOT).

The basic expression for a logical operation is,

```
x1 op x2
```

where x1, x2 can be boolean expressions, variables or constants, and op is either &&, &, ||, | or ^ operator. The truth tables that will be shown next, summarize the result of each operation for all possible combinations of x1 and x2.
3.10.4.1  && (logical AND) and & (boolean logical AND)

Here is the truth table for && and &,

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Table 13: Truth table for & and &&

The basic difference between && and & operators is that && supports short-circuit evaluations (or partial evaluations), while & doesn't. What does this mean?

Given an expression,

```java
exp1 && exp2
```

&& will evaluate the expression exp1, and immediately return a false value is exp1 is false. If exp1 is false, the operator never evaluates exp2 because the result of the operator will be false regardless of the value of exp2. In contrast, the & operator always evaluates both exp1 and exp2 before returning an answer.

Here's a sample source code that uses logical and boolean AND,

```java
public class TestAND {
    public static void main( String[] args ){
        int i = 0;
        int j = 10;
        boolean test= false;

        //demonstrate &&
        test = (i > 10) && (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);

        //demonstrate &
        test = (i > 10) & (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);
    }
}
```
The output of the program is,

```
0
10
false
0
11
false
```

Note, that the j++ on the line containing the && operator is not evaluated since the first expression (i>10) is already equal to false.
### 3.10.4.2 \ || (logical OR) and | (boolean logical inclusive OR) \\

Here is the truth table for || and |,

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

*Table 14: Truth table for | and ||*

The basic difference between || and | operators is that || supports short-circuit evaluations (or partial evaluations), while | doesn't. What does this mean?

Given an expression,

\[ \text{exp1} \ || \ \text{exp2} \]

|| will evaluate the expression exp1, and immediately return a true value is exp1 is true. If exp1 is true, the operator never evaluates exp2 because the result of the operator will be true regardless of the value of exp2. In contrast, the | operator always evaluates both exp1 and exp2 before returning an answer.

Here's a sample source code that uses logical and boolean OR,

```java
public class TestOR {
    public static void main( String[] args ){
        int i = 0;
        int j = 10;
        boolean test= false;

        //demonstrate ||
        test = (i < 10) || (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);

        //demonstrate |
        test = (i < 10) | (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);
    }
}
```
The output of the program is,

0
10
true
0
11
true

Note, that the j++ on the line containing the || operator is not evaluated since the first expression (i<10) is already equal to true.
3.10.4.3 ^ (boolean logical exclusive OR)

Here is the truth table for ^,

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x2</td>
<td>Result</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Table 15: Truth table for ^

The result of an exclusive OR operation is TRUE, if and only if one operand is true and the other is false. Note that both operands must always be evaluated in order to calculate the result of an exclusive OR.

Here's a sample source code that uses the logical exclusive OR operator,

```java
public class TestXOR {
    public static void main( String[] args ){
        boolean val1 = true;
        boolean val2 = true;
        System.out.println(val1 ^ val2);

        val1 = false;
        val2 = true;
        System.out.println(val1 ^ val2);

        val1 = false;
        val2 = false;
        System.out.println(val1 ^ val2);

        val1 = true;
        val2 = false;
        System.out.println(val1 ^ val2);
    }
}
```

The output of the program is,

false
true
false
true
3.10.4.4 ! (logical NOT)
The logical NOT takes in one argument, wherein that argument can be an expression, variable or constant. Here is the truth table for !,

<table>
<thead>
<tr>
<th>x1</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

*Table 16: Truth table for !*

Here's a sample source code that uses the logical NOT operator,

```java
public class TestNOT {
    public static void main( String[] args ){
        boolean val1 = true;
        boolean val2 = false;
        System.out.println(!val1);
        System.out.println(!val2);
    }
}
```

The output of the program is,

```java
false
true
```
3.10.5 **Conditional Operator (?:)**

The conditional operator `?:` is a ternary operator. This means that it takes in three arguments that together form a conditional expression. The structure of an expression using a conditional operator is,

```
exp1?exp2:exp3
```

wherein `exp1` is a boolean expression whose result must either be true or false.

If `exp1` is true, `exp2` is the value returned. If it is false, then `exp3` is returned.

For example, given the code,

```java
class ConditionalOperator
{
    public static void main( String[] args )
    {
        String status = "";
        int grade = 80;
        //get status of the student
        status = (grade >= 60)?"Passed":"Fail";
        //print status
        System.out.println( status );
    }
}
```

The output of this program will be,

```
Passed
```
Here is the flowchart of how `?:` works,

```
grade = 80
CONDITION is true
grade >= 60
print Passed
CONDITION is false
print Fail
```

**Figure 3.2: Flowchart using the `?:` operator**

Here is another program that uses the `?:` operator,

```java
class ConditionalOperator {
    public static void main(String[] args) {
        int score = 0;
        char answer = 'a';

        score = (answer == 'a') ? 10 : 0;
        System.out.println("Score = " + score);
    }
}
```

The output of the program is,

```
Score = 10
```
3.10.6 Operator Precedence

Operator precedence defines the compiler’s order of evaluation of operators so as to come up with an unambiguous result.

Given a complicated expression,

\[ 6 \% 2 * 5 + 4 / 2 + 88 - 10 \]

we can re-write the expression and place some parenthesis base on operator precedence,

\[ ((6 \% 2) * 5) + (4 / 2) + 88 - 10; \]

**Coding Guidelines**

*To avoid confusion in evaluating mathematical operations, keep your expressions simple and use parenthesis.*
## 3.11 Exercises

### 3.11.1 Declaring and printing variables

Given the table below, declare the following variables with the corresponding data types and initialization values. Output to the screen the variable names together with the values.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Data Type</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>integer</td>
<td>10</td>
</tr>
<tr>
<td>letter</td>
<td>character</td>
<td>a</td>
</tr>
<tr>
<td>result</td>
<td>boolean</td>
<td>true</td>
</tr>
<tr>
<td>str</td>
<td>String</td>
<td>hello</td>
</tr>
</tbody>
</table>

The following should be the expected screen output,

    Number = 10
    letter = a
    result = true
    str = hello

### 3.11.2 Getting the average of three numbers

Create a program that outputs the average of three numbers. Let the values of the three numbers be, 10, 20 and 45. The expected screen output is,

    number 1 = 10
    number 2 = 20
    number 3 = 45
    Average is = 25

### 3.11.3 Output greatest value

Given three numbers, write a program that outputs the number with the greatest value among the three. Use the conditional ?: operator that we have studied so far (**HINT**: You will need to use two sets of ?: to solve this). For example, given the numbers 10, 23 and 5, your program should output,

    number 1 = 10
    number 2 = 23
    number 3 = 5
    The highest number is = 23

### 3.11.4 Operator precedence

Given the following expressions, re-write them by writing some parenthesis based on the sequence on how they will be evaluated.

1. \(a / b ^ c ^ d - e + f - g * h + i\)
2. \(3 * 10 *2 / 15 - 2 + 4 ^ 2 ^ 2\)
3. \(r ^ s * t / u - v + w ^ x - y++\)
4 Getting Input from the Keyboard

4.1 Objectives

Now that we've studied some basic concepts in Java and we've written some simple programs, let's make our programs more interactive by getting some input from the user. In this section, we'll be discussing two methods of getting input, the first one is through the use of the BufferedReader class and the other one involves a graphical user interface by using JOptionPane.

At the end of the lesson, the student should be able to:

- Create an interactive Java program that gets input from the keyboard
- Use the BufferedReader class to get input from the keyboard using a console
- Use the JOptionPane class to get input from the keyboard using a graphical user interface

4.2 Using BufferedReader to get input

In this section, we will use the BufferedReader class found in the java.io package in order to get input from the keyboard.

Here are the steps to get input from the keyboard:

1. Add this at the top of your code:

   import java.io.*;

2. Add this statement:

   BufferedReader dataIn = new BufferedReader(
   new InputStreamReader( System.in) );

3. Declare a temporary String variable to get the input, and invoke the readLine() method to get input from the keyboard. You have to type it inside a try-catch block.

   try{
     String temp = dataIn.readLine();
   }
   catch( IOException e ){
     System.out.println(“Error in getting input”);
   }
Here is the complete source code:

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class GetInputFromKeyboard {
    public static void main(String[] args) {
        BufferedReader dataIn = new BufferedReader(new InputStreamReader(System.in));
        String name = "";
        System.out.print("Please Enter Your Name:");
        try{
            name = dataIn.readLine();
        }catch( IOException e ){
            System.out.println("Error!");
        }
        System.out.println("Hello " + name +"!");
    }
}
```

Now let's try to explain each line of code:

The statements,

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

indicate that we want to use the classes `BufferedReader`, `InputStreamReader` and `IOException` which is inside the `java.io` package. The Java Application Programming Interface (API) contains hundreds of predefined classes that you can use in your programs. These classes are organized into what we call packages.

**Packages** contain classes that have related purpose. Just like in our example, the `java.io` package contains classes that allow programs to input and output data. The statements can also be rewritten as,

```java
import java.io.*;
```

which will load all the classes found in the package, and then we can use those classes inside our program.
The next two statements,

```java
public class GetInputFromKeyboard {
    public static void main(String[] args) {
```

were already discussed in the previous lesson. This means we declare a class named GetInputFromKeyboard and we declare the main method.

In the statement,

```java
BufferedReader dataIn = new BufferedReader(new InputStreamReader(System.in));
```

we are declaring a variable named dataIn with the class type BufferedReader. Don't worry about what the syntax means for now. We will cover more about this later in the course.

Now, we are declaring a String variable with the identifier name,

```java
String name = "";
```

This is where we will store the input of the user. The variable name is initialized to an empty String "". It is always good to initialize your variables as you declare them.

The next line just outputs a String on the screen asking for the user's name.

```java
System.out.print("Please Enter Your Name:");
```

Now, the following block defines a try-catch block,

```java
try{
    name = dataIn.readLine();
}catch( IOException e ){
    System.out.println("Error!");
}
```

This assures that the possible exceptions that could occur in the statement

```java
name = dataIn.readLine();
```

will be caught. We will cover more about exception handling in the latter part of this course, but for now, just take note that you need to add this code in order to use the readLine() method of BufferedReader to get input from the user.
Now going back to the statement,

```java
name = dataIn.readLine();
```

the method call, `dataIn.readLine()`, gets input from the user and will return a String value. This value will then be saved to our `name` variable, which we will use in our final statement to greet the user,

```java
System.out.println("Hello " + name + ":");
```
4.3 Using JOptionPane to get input

Another way to get input from the user is by using the JOptionPane class which is found in the javax.swing package. JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.

Given the following code,

```java
import javax.swing.JOptionPane;

public class GetInputFromKeyboard {
    public static void main(String[] args) {
        String name = "";
        name = JOptionPane.showInputDialog("Please enter your name");

        String msg = "Hello " + name + "!";
        JOptionPane.showMessageDialog(null, msg);
    }
}
```

This will output,

```
Input

Please enter your name

OK Cancel
```

Figure 4.1: Getting Input Using JOptionPane

```
Input

Please enter your name

Florence

OK Cancel
```

Figure 4.2: Input florence on the JOptionPane

```
Message

Hello florence!

OK
```

Figure 4.3: Showing Message Using JOptionPane
The first statement,

```java
import javax.swing.JOptionPane;
```

indicates that we want to import the class `JOptionPane` from the `javax.swing` package.

We can also write this as,

```java
import javax.swing.*;
```

The statement,

```java
name = JOptionPane.showInputDialog("Please enter your name");
```

creates a `JOptionPane` input dialog, which will display a dialog with a message, a textfield and an OK button as shown in the figure. This returns a String which we will save in the `name` variable.

Now we create the welcome message, which we will store in the `msg` variable,

```java
String msg = "Hello " + name + "!";
```

The next line displays a dialog which contains a message and an OK button.

```java
JOptionPane.showMessageDialog(null, msg);
```
4.4 Exercises

4.4.1 Last 3 words (BufferedReader version)

Using BufferedReader, ask for three words from the user and output those three words on the screen. For example,

Enter word1: Goodbye
Enter word2: and
Enter word3: Hello

Goodbye and Hello

4.4.2 Last 3 words (JOptionPane version)

Using JOptionPane, ask for three words from the user and output those three words on the screen. For example,
5 Control Structures

5.1 Objectives

In the previous sections, we have given examples of sequential programs, wherein statements are executed one after another in a fixed order. In this section, we will be discussing control structures, which allows us to change the ordering of how the statements in our programs are executed.

At the end of the lesson, the student should be able to:

- Use decision control structures (if, else, switch) which allows selection of specific sections of code to be executed
- Use repetition control structures (while, do-while, for) which allow executing specific sections of code a number of times
- Use branching statements (break, continue, return) which allows redirection of program flow

5.2 Decision Control Structures

Decision control structures are Java statements that allows us to select and execute specific blocks of code while skipping other sections.

5.2.1 if statement

The if-statement specifies that a statement (or block of code) will be executed if and only if a certain boolean statement is true.

The if-statement has the form,

```java
if( boolean_expression )
    statement;
```

or

```java
if( boolean_expression ){
    statement1;
    statement2;
    ...
}
```

where, boolean_expression is either a boolean expression or boolean variable.
For example, given the code snippet,

```java
int grade = 68;
if( grade > 60 ) System.out.println("Congratulations!");
```

or

```java
int grade = 68;
if( grade > 60 ){
    System.out.println("Congratulations!");
    System.out.println("You passed!");
}
```

**Coding Guidelines:**

1. The `boolean_expression` part of a statement should evaluate to a boolean value. That means that the execution of the condition should either result to a value of `true` or a `false`.
2. Indent the statements inside the if-block. For example,
   ```java
   if( boolean_expression ){
       //statement1;
       //statement2;
   }
   ```

---

*Figure 5.1: Flowchart of If-Statement*
The if-else statement is used when we want to execute a certain statement if a condition is true, and a different statement if the condition is false.

The if-else statement has the form,

```
if( boolean_expression )
    statement;
else
    statement;
```

or can also be written as,

```
if( boolean_expression ){
    statement1;
    statement2;
    . . .
}
else{
    statement1;
    statement2;
    . . .
}
```

For example, given the code snippet,

```
int grade = 68;
if( grade > 60 )   System.out.println("Congratulations!");
else    System.out.println("Sorry you failed");
```

or

```
int grade = 68;
if( grade > 60 ){
    System.out.println("Congratulations!");
    System.out.println("You passed!");
}
else{
    System.out.println("Sorry you failed");
}
```
Coding Guidelines:

1. To avoid confusion, always place the statement or statements of an if or if-else block inside brackets {}.
2. You can have nested if-else blocks. This means that you can have other if-else blocks inside another if-else block. For example,
   ```
   if( boolean_expression ){
       if( boolean_expression ){
           . . .
       }
       else{
       }
   }
   ```

Figure 5.2: Flowchart of If-Else Statement
5.2.3 if-else-if statement

The statement in the else-clause of an if-else block can be another if-else structures. This cascading of structures allows us to make more complex selections.

The if-else if statement has the form,

```c
if( boolean_expression1 )
    statement1;
else if( boolean_expression2 )
    statement2;
else
    statement3;
```

Take note that you can have many else-if blocks after an if-statement. The else-block is optional and can be omitted. In the example shown above, if boolean_expression1 is true, then the program executes statement1 and skips the other statements. If boolean_expression2 is true, then the program executes statement 2 and skips to the statements following statement3.

![Flowchart of If-Else-If Statement](image)

*Figure 5.3: Flowchart of If-Else-If Statement*
For example, given the code snippet,

```java
int grade = 68;
if (grade > 90)
    System.out.println("Very good!");
else if (grade > 60)
    System.out.println("Very good!");
else
    System.out.println("Sorry you failed");
```

### 5.2.4 Common Errors when using the if-else statements:

1. The condition inside the if-statement does not evaluate to a boolean value. For example,

   ```java
   //WRONG
   int number = 0;
   if (number)
       //some statements here
   ```

   The variable number does not hold a Boolean value.

2. Using `=` instead of `==` for comparison. For example,

   ```java
   //WRONG
   int number = 0;
   if (number = 0)
       //some statements here
   ```

   This should be written as,

   ```java
   //CORRECT
   int number = 0;
   if (number == 0)
       //some statements here
   ```

3. Writing `elseif` instead of `else if`.


5.2.5 Example for if-else-else if

```java
public class Grade {
    public static void main(String[] args) {
        double grade = 92.0;

        if (grade >= 90) {
            System.out.println("Excellent! ");
        } else if (grade < 90 && grade >= 80) {
            System.out.println("Good job! ");
        } else if (grade < 80 && grade >= 60) {
            System.out.println("Study harder! ");
        } else {
            System.out.println("Sorry, you failed.");
        }
    }
}
```
5.2.6 switch statement

Another way to indicate a branch is through the `switch` keyword. The switch construct allows branching on multiple outcomes.

The switch statement has the form,

```java
switch( switch_expression ){
    case case_selector1:
        statement1; // block 1
        . . . //
        break;
    case case_selector2:
        statement1; // block 2
        . . . //
        break;
    . . .
    default:
        statement1; // block n
        . . . //
        break;
}
```

where, switch_expression is an `integer` or `character` expression and, case_selector1, case_selector2 and so on, are unique integer or character constants.

When a switch is encountered, Java first evaluates the switch_expression, and jumps to the case whose selector matches the value of the expression. The program executes the statements in order from that point on until a break statement is encountered, skipping then to the first statement after the end of the switch structure.

If none of the cases are satisfied, the default block is executed. Take note however, that the default part is optional. A switch statement can have no default block.

NOTES:
- Unlike with the `if` statement, the multiple statements are executed in the switch statement without needing the curly braces.
- When a case in a switch statement has been matched, all the statements associated with that case are executed. Not only that, the statements associated with the succeeding cases are also executed.
- To prevent the program from executing statements in the subsequent cases, we use a `break` statement as our last statement.
Coding Guidelines:

1. Deciding whether to use an if statement or a switch statement is a judgment call. You can decide which to use, based on readability and other factors.
2. An if statement can be used to make decisions based on ranges of values or conditions, whereas a switch statement can make decisions based only on a single integer or character value. Also, the value provided to each case statement must be unique.

Figure 5.4: Flowchart of Switch Statements
5.2.7 Example for switch

```java
public class Grade {
    public static void main( String[] args ) {
        int grade = 92;
        switch(grade){
            case 100:
                System.out.println( "Excellent!" );
                break;
            case 90:
                System.out.println("Good job!");
                break;
            case 80:
                System.out.println("Study harder!");
                break;
            default:
                System.out.println("Sorry, you failed.");
        }
    }
}
```
5.3 Repetition Control Structures

Repetition control structures are Java statements that allows us to execute specific blocks of code a number of times. There are three types of repetition control structures, the while, do-while and for loops.

5.3.1 while loop

The while loop is a statement or block of statements that is repeated as long as some condition is satisfied.

The while statement has the form,

\[
\text{while( boolean_expression )}{  
\text{statement1;  }  
\text{statement2;  }  
\ldots
\}
\]

The statements inside the while loop are executed as long as the boolean_expression evaluates to true.

For example, given the code snippet,

```java
int i = 4;  
while ( i > 0 ){  
    System.out.print(i);  
    i--;  
}
```

The sample code shown will print 4321 on the screen. Take note that if the line containing the statement i--; is removed, this will result to an infinite loop, or a loop that does not terminate. Therefore, when using while loops or any kind of repetition control structures, make sure that you add some statements that will allow your loop to terminate at some point.
The following are other examples of while loops,

**Example 1:**

```java
int x = 0;
while (x<10)
{
    System.out.println(x);
    x++;
}
```

**Example 2:**

```java
//infinite loop
while(true)
    System.out.println("hello");
```

**Example 3:**

```java
//no loops
// statement is not even executed
while (false)
    System.out.println("hello");
```
5.3.2 do-while loop

The do-while loop is similar to the while-loop. The statements inside a do-while loop are executed several times as long as the condition is satisfied.

The main difference between a while and do-while loop is that, the statements inside a do-while loop are executed at least once.

The do-while statement has the form,

```
do{
    statement1;
    statement2;
    . . .
}while( boolean_expression );
```

The statements inside the do-while loop are first executed, and then the condition in the boolean_expression part is evaluated. If this evaluates to true, the statements inside the do-while loop are executed again.

Here are a few examples that uses the do-while loop:

**Example 1:**

```
int x = 0;
do{
    System.out.println(x);
x++;
}while (x<10);
```

This example will output 0123456789 on the screen.

**Example 2:**

```
//infinite loop
do{
    System.out.println("hello");
} while (true);
```

This example will result to an infinite loop, that prints hello on screen.

**Example 3:**

```
//one loop
// statement is executed once
do
    System.out.println("hello");
while (false);
```

This example will output hello on the screen.
Coding Guidelines:

1. Common programming mistakes when using the do-while loop is forgetting to write the semi-colon after the while expression.
   ```java
   do{
      ...
   }while(boolean_expression) //WRONG->forgot semicolon ;
   ```

2. Just like in while loops, make sure that your do-while loops will terminate at some point.

5.3.3 for loop

The for loop, like the previous loops, allows execution of the same code a number of times.

The for loop has the form,

```java
for (InitializationExpression; LoopCondition; StepExpression){
   statement1;
   statement2;
   . . .
}
```

where,

- **InitializationExpression** - initializes the loop variable.
- **LoopCondition** - compares the loop variable to some limit value.
- **StepExpression** - updates the loop variable.

A simple example of the for loop is,

```java
int i;
for( i = 0; i < 10; i++ ){
   System.out.print(i);
}
```

In this example, the statement i=0, first initializes our variable. After that, the condition expression i<10 is evaluated. If this evaluates to true, then the statement inside the for loop is executed. Next, the expression i++ is executed, and then the condition expression is again evaluated. This goes on and on, until the condition expression evaluates to false.

This example, is equivalent to the while loop shown below,

```java
int i = 0;
while( i < 10 ){
   System.out.print(i);
   i++;
}
```
5.4 Branching Statements

Branching statements allows us to redirect the flow of program execution. Java offers three branching statements: break, continue and return.

5.4.1 break statement

The break statement has two forms: unlabeled (we saw its unlabeled form in the switch statement) and labeled.

5.4.1.1 Unlabeled break statement

The unlabeled break terminates the enclosing switch statement, and flow of control transfers to the statement immediately following the switch. You can also use the unlabeled form of the break statement to terminate a for, while, or do-while loop.

For example,

```java
String names[] = {"Beah", "Bianca", "Lance", "Belle",
                 "Nico", "Yza", "Gem", "Ethan"};

String searchName = "Yza";
boolean foundName = false;

for( int i=0; i< names.length; i++ ){
    if( names[i].equals( searchName ) ){
        foundName = true;
        break;
    }
}

if( foundName ){
    System.out.println( searchName + " found!" );
} else{
    System.out.println( searchName + " not found." );
}
```

In this example, if the search string "Yza" is found, the for loop will stop and flow of control transfers to the statement following the for loop.
5.4.1.2 Labeled break statement

The labeled form of a break statement terminates an outer statement, which is identified by the label specified in the break statement. The following program searches for a value in a two-dimensional array. Two nested for loops traverse the array. When the value is found, a labeled break terminates the statement labeled search, which is the outer for loop.

```java
int[][] numbers = {{1, 2, 3},
                   {4, 5, 6},
                   {7, 8, 9}};

int searchNum = 5;
boolean foundNum = false;

searchLabel:
for( int i=0; i<numbers.length; i++ ){
    for( int j=0; j<numbers[i].length; j++ ){
        if( searchNum == numbers[i][j] ){
            foundNum = true;
            break searchLabel;
        }
    }
}

if( foundNum ){
    System.out.println( searchNum + " found!" );
} else{
    System.out.println( searchNum + " not found!" );
}
```

The break statement terminates the labeled statement; it does not transfer the flow of control to the label. The flow of control transfers to the statement immediately following the labeled (terminated) statement.
5.4.2 continue statement

The continue statement has two forms: unlabeled and labeled. You can use the continue statement to skip the current iteration of a for, while or do-while loop.

5.4.2.1 Unlabeled continue statement

The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop, basically skipping the remainder of this iteration of the loop.

The following example counts the number of "Beah"s in the array.

```java
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int count = 0;
for( int i=0; i<names.length; i++ ){
    if( !names[i].equals("Beah") ){
        continue; //skip next statement
    }
    count++;
}
System.out.println("There are " + count + " Beahs in the list");
```

5.4.2.2 Labeled continue statement

The labeled form of the continue statement skips the current iteration of an outer loop marked with the given label.

```java
outerLoop:
    for( int i=0; i<5; i++ ){
        for( int j=0; j<5; j++ ){
            System.out.println("Inside for(j) loop"); //message1
            if( j == 2 ) continue outerLoop;
        }
        System.out.println("Inside for(i) loop"); //message2
    }
```

In this example, message 2 never gets printed since we have the statement continue outerloop which skips the iteration.
5.4.3 return statement

The return statement is used to exit from the current method. The flow of control returns to the statement that follows the original method call. The return statement has two forms: one that returns a value and one that doesn't.

To return a value, simply put the value (or an expression that calculates the value) after the return keyword. For example,

```
return ++count;
```

or

```
return "Hello";
```

The data type of the value returned by return must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value. For example,

```
return;
```

We will cover more about return statements later when we discuss about methods.
5.5 Exercises

5.5.1 Grades
Get three exam grades from the user and compute the average of the grades. Output the average of the three exams. Together with the average, also include a smiley face in the output if the average is greater than or equal to 60, otherwise output :-(. 
1. Use BufferedReader to get input from the user, and System.out to output the result. 
2. Use JOptionPane to get input from the user and to output the result.

5.5.2 Number in words
Get a number as input from the user, and output the equivalent of the number in words. The number inputted should range from 1-10. If the user inputs a number that is not in the range, output, "Invalid number". 
1. Use an if-else statement to solve this problem 
2. Use a switch statement to solve this problem 

5.5.3 Hundred Times
Create a program that prints your name a hundred times. Do three versions of this program using a while loop, a do-while loop and a for-loop.

5.5.4 Powers
Compute the power of a number given the base and exponent. Do three versions of this program using a while loop, a do-while loop and a for-loop.
6 Java Arrays

6.1 Objectives

In this section, we will be discussing about Java Arrays. First, we are going to define what arrays are, and then we are going to discuss on how to declare and use them.

At the end of the lesson, the student should be able to:

- Declare and create arrays
- Access array elements
- Determine the number of elements in an array
- Declare and create multidimensional arrays

6.2 Introduction to arrays

In the previous sections, we have discussed on how to declare different variables using the primitive data types. In declaring variables, we often use a unique identifier or name and a datatype. In order to use the variable, we call it by its identifier name.

For example, we have here three variables of type int with different identifiers for each variable.

```java
int number1;
int number2;
int number3;

number1 = 1;
number2 = 2;
number3 = 3;
```

As you can see, it seems like a tedious task in order to just initialize and use the variables especially if they are used for the same purpose. In Java and other programming languages, there is one capability wherein we can use one variable to store a list of data and manipulate them more efficiently. This type of variable is called an array.

An array stores multiple data items of the same datatype, in a contiguous block of memory, divided into a number of slots. Think of an array as a stretched variable – a location that still has one identifier name, but can hold more than one value.
6.3 Declaring Arrays

Arrays must be declared like all variables. When declaring an array, you list the data type, followed by a set of square brackets[], followed by the identifier name. For example,

```java
int [] ages;
```

or you can place the brackets after the identifier. For example,

```java
int ages[];
```

After declaring, we must create the array and specify its length with a constructor statement. This process in Java is called instantiation (the Java word for creates). In order to instantiate an object, we need to use a constructor for this. We will cover more about instantiating objects and constructors later. Take note, that the size of an array cannot be changed once you’ve initialized it. For example,

```java
//declaration
int ages[];

//instantiate object
ages = new int[100];
```

or, can also be written as,

```java
//declare and instantiate object
int ages[] = new int[100];
```

In the example, the declaration tells the Java Compiler that the identifier `ages` will be used as the name of an array containing integers, and to create or instantiate a new array containing 100 elements.

Instead of using the new keyword to instantiate an array, you can also automatically declare, construct and assign values at once.

```
Figure 6.2: Instantiating Arrays
```
Examples are,

```java
//creates an array of boolean variables with identifier results. This array contains 4 elements that are initialized to values {true, false, true, false}
boolean results[] = { true, false, true, false };

//creates an array of 4 double variables initialized to the values {100, 90, 80, 75};
double[] grades = {100, 90, 80, 75};

//creates an array of Strings with identifier days and initialized. This array contains 7 elements
String days[] = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };```
6.4 Accessing an array element

To access an array element, or a part of the array, you use a number called an index or a subscript.

An index number or subscript is assigned to each member of the array, allowing the program and the programmer to access individual values when necessary. Index numbers are always integers. They begin with zero and progress sequentially by whole numbers to the end of the array. Take note that the elements inside your array is from 0 to (sizeOfArray-1).

For example, given the array we declared a while ago, we have

```java
//assigns 10 to the first element in the array
ages[0] = 10;

//prints the last element in the array
System.out.print(ages[99]);
```

Take note that once an array is declared and constructed, the stored value of each member of the array will be initialized to zero for number data. However, reference data types such as Strings are not initialized to blanks or an empty string "". Therefore, you must populate the String arrays explicitly.

The following is a sample code on how to print all the elements in the array. This uses a for loop, so our code is shorter.

```java
public class ArraySample{
    public static void main( String[] args ){
        int[] ages = new int[100];
        for( int i=0; i<100; i++ ){
            System.out.print( ages[i] );
        }
    }
}
```

**Coding Guidelines:**

1. It is usually better to initialize or instantiate the array right away after you declare it. For example, the declaration,
   ```java
   int []arr = new int[100];
   ```
   is preferred over,
   ```java
   int []arr;
   arr = new int[100];
   ```
2. The elements of an n-element array have indexes from 0 to n-1. Note that there is no array element arr[n]! This will result in an array-index-out-of-bounds exception.
3. You cannot resize an array.
6.5 Array length

In order to get the number of elements in an array, you can use the length field of an array. The length field of an array returns the size of the array. It can be used by writing,

\[ \text{arrayName}.length \]

For example, given the previous example, we can re-write it as,

```java
public class ArraySample {
    public static void main( String[] args ){
        int[] ages = new int[100];
        for( int i=0; i<ages.length; i++ ){
            System.out.print( ages[i] );
        }
    }
}
```

**Coding Guidelines:**

1. When creating for loops to process the elements of an array, use the array object’s length field in the condition statement of the for loop. This will allow the loop to adjust automatically for different-sized arrays.
2. Declare the sizes of arrays in a Java program using named constants to make them easy to change. For example,

   ```java
   final int ARRAY_SIZE = 1000; //declare a constant
   ...
   int[] ages = new int[ARRAY_SIZE];
   ```
6.6 Multidimensional Arrays

Multidimensional arrays are implemented as arrays of arrays. Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name. For example,

```
// integer array 512 x 128 elements
int[][] twoD = new int[512][128];

// character array 8 x 16 x 24
char[][][] threeD = new char[8][16][24];

// String array 4 rows x 2 columns
String[][] dogs = {{ "terry", "brown" },
                   { "Kristin", "white" },
                   { "toby", "gray" },
                   { "fido", "black" }};
```

To access an element in a multidimensional array is just the same as accessing the elements in a one dimensional array. For example, to access the first element in the first row of the array `dogs`, we write,

```
System.out.print( dogs[0][0] );
```

This will print the String "terry" on the screen.
6.7 Exercises

6.7.1 Days of the Week
Create an array of Strings which are initialized to the 7 days of the week. For Example,

```java
String days[] = {"Monday", "Tuesday"...};
```

Using a while-loop, print all the contents of the array. (do the same for do-while and for-loop)

6.7.2 Greatest number
Using BufferedReader or JOptionPane, ask for 10 numbers from the user. Use an array to store the values of these 10 numbers. Output on the screen the number with the greatest value.

6.7.3 Addressbook Entries
Given the following multidimensional array that contains addressbook entries:

```java
String entry = {"Florence", "735-1234", "Manila"},
    {"Joyce", "983-3333", "Quezon City"},
    {"Becca", "456-3322", "Manila"};
```

Print the following entries on screen in the following format:

Name : Florence
Tel. # : 735-1234
Address : Manila

Name : Joyce
Tel. # : 983-3333
Address : Quezon City

Name : Becca
Tel. # : 456-3322
Address : Manila
7 Command-line Arguments

7.1 Objectives
In this section, we will study on how to process input from the command-line by using arguments pass onto a Java program.

At the end of the lesson, the student should be able to:
- Know and explain what a command-line argument is
- Get input from the user using command-line arguments
- Learn how to pass arguments to your programs in NetBeans

7.2 Command-line arguments
A Java application can accept any number of arguments from the command-line. Command-line arguments allow the user to affect the operation of an application for one invocation. The user enters command-line arguments when invoking the application and specifies them after the name of the class to run.

For example, suppose you have a Java application, called Sort, that sorts five numbers, you run it like this:

```
$ java Sort 5 4 3 2 1
```

Take note that the arguments are separated by spaces.
In the Java language, when you invoke an application, the runtime system passes the command-line arguments to the application's main method via an array of Strings. Each String in the array contains one of the command-line arguments. Remember the declaration for the main method,

```java
public static void main( String[] args )
```

The arguments that are passed to your program are saved into an array of String with the args identifier.

In the previous example, the command-line arguments passed to the Sort application is an array that contains five strings which are: "5", "4", "3", "2" and "1". You can derive the number of command-line arguments with the array's length attribute.

For example,

```java
int numberOfArgs = args.length;
```

If your program needs to support a numeric command-line argument, it must convert a String argument that represents a number, such as "34", to a number. Here's a code snippet that converts a command-line argument to an integer,

```java
int firstArg = 0;
if (args.length > 0){
    firstArg = Integer.parseInt(args[0]);
}
```

parseInt throws a NumberFormatException (ERROR) if the format of args[0] isn't valid (not a number).

**Coding Guidelines:**

*Before using command-line arguments, always check if the number of arguments before accessing the array elements so that there will be no exception generated.*
7.3 Command-line arguments in NetBeans

To illustrate on how to pass some arguments to your programs in NetBeans, let us create a Java program that will print the number of arguments and the first argument passed to it.

```java
public class CommandLineExample
{
    public static void main( String[] args ){
        System.out.println("Number of arguments=" + args.length);
        System.out.println("First Argument="+ args[0]);
    }
}
```

Now, run netbeans and create a new project and name this CommandLineExample. Copy the code shown above and compile the code. Now, follow these steps to pass arguments to your program using NetBeans.

Click on Projects (encircled below).

![Image of NetBeans opening project file](image.png)

*Figure 7.2: Opening Project File*
Right-click on the CommandLineExample icon, and a popup menu will appear. Click on Properties.

The Project Properties dialog will then appear.
Now, click on Run

On the Arguments textbox, type the arguments you want to pass to your program. In this case we typed in the arguments 5 4 3 2 1. Then, click on the OK button.
Now try to RUN your program.

As you can see here, the output to your program is the number of arguments which is 5, and the first argument which is 5.
7.4 Exercises

7.4.1 Print arguments
Get input from the user using command-line arguments and print all the arguments to the screen. For example, if the user entered,

```java
java Hello world that is all
```

your program should print

```
Hello
world
that
is
all
```

7.4.2 Arithmetic Operations
Get two numbers from the user using command-line arguments and print sum, difference, product and quotient of the two numbers. For example, if the user entered,

```java
java ArithmeticOperation 20 4
```

your program should print

```
sum = 24
difference = 16
product = 80
quotient = 5
```
8 Working with the Java Class Library

8.1 Objectives

In this section, we will introduce some basic concepts of object-oriented programming. Later on, we will discuss the concept of classes and objects, and how to use these classes and their members. Comparison, conversion and casting of objects will also be covered. For now, we will focus on using classes that are already defined in the Java class library, we will discuss later on how to create your own classes.

At the end of the lesson, the student should be able to:

- Explain object-oriented programming and some of its concepts
- Differentiate between classes and objects
- Differentiate between instance variables/methods and class (static) variables/methods
- Explain what methods are and how to call and pass parameters to methods
- Identify the scope of a variable
- Cast primitive data types and objects
- Compare objects and determine the class of an objects

8.2 Introduction to Object-Oriented Programming

Object-Oriented programming or OOP revolves around the concept of objects as the basic elements of your programs. When we compare this to the physical world, we can find many objects around us, such as cars, lion, people and so on. These objects are characterized by their properties (or attributes) and behaviors.

For example, a car object has the properties, type of transmission, manufacturer and color. Its behaviors are turning, braking and accelerating. Similarly, we can define different properties and behavior of a lion. Please refer to the table below for the examples.

<table>
<thead>
<tr>
<th>Object</th>
<th>Properties</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>type of transmission</td>
<td>turning</td>
</tr>
<tr>
<td></td>
<td>manufacturer</td>
<td>braking</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>accelerating</td>
</tr>
<tr>
<td>Lion</td>
<td>Weight</td>
<td>roaring</td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>sleeping</td>
</tr>
<tr>
<td></td>
<td>hungry or not hungry</td>
<td>hunting</td>
</tr>
<tr>
<td></td>
<td>tamed or wild</td>
<td></td>
</tr>
</tbody>
</table>

Table 17: Example of Real-life Objects
With these descriptions, the objects in the physical world can easily be modeled as software objects using the properties as data and the behaviors as methods. These data and methods could even be used in programming games or interactive software to simulate the real-world objects! An example would be a car software object in a racing game or a lion software object in an educational interactive software zoo for kids.

8.3 Classes and Objects

8.3.1 Difference Between Classes and Objects

In the software world, an object is a software component whose structure is similar to objects in the real world. Each object is composed of a set of data (properties/attributes) which are variables describing the essential characteristics of the object, and it also consists of a set of methods (behavior) that describes how an object behaves. Thus, an object is a software bundle of variables and related methods. The variables and methods in a Java object are formally known as instance variables and instance methods to distinguish them from class variables and class methods, which will be discussed later.

The class is the fundamental structure in object-oriented programming. It can be thought of as a template, a prototype or a blueprint of an object. It consists of two types of members which are called fields (properties or attributes) and methods. Fields specify the data types defined by the class, while methods specify the operations. An object is an instance of the class.

To differentiate between classes and objects, let us discuss an example. What we have here is a Car Class which can be used to define several Car Objects. In the table shown below, Car A and Car B are objects of the Car class. The class has fields plate number, color, manufacturer, and current speed which are filled-up with corresponding values in objects Car A and Car B. The Car has also some methods Accelerate, Turn and Brake.

<table>
<thead>
<tr>
<th>Car Class</th>
<th>Object Car A</th>
<th>Object Car B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plate Number</td>
<td>ABC 111</td>
<td>XYZ 123</td>
</tr>
<tr>
<td>Color</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Mitsubishi</td>
<td>Toyota</td>
</tr>
<tr>
<td>Current Speed</td>
<td>50 km/h</td>
<td>100 km/h</td>
</tr>
<tr>
<td><strong>Instance Methods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accelerate Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brake Method</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 18: Example of Car class and its objects

When instantiated, each object gets a fresh set of state variables. However, the method implementations are shared among objects of the same class.

Classes provide the benefit of reusability. Software programmers can use a class over and over again to create many objects.
8.3.2 Encapsulation

Encapsulation is the method of hiding certain elements of the implementation of a certain class. By placing a boundary around the properties and methods of our objects, we can prevent our programs from having side effects wherein programs have their variables changed in unexpected ways.

We can prevent access to our object’s data by declaring them in a certain way such that we can control access to them. We will learn more about how Java implements encapsulation as we discuss more about classes.

8.3.3 Class Variables and Methods

In addition to the instance variables, it is also possible to define class variables, which are variables that belong to the whole class. This means that it has the same value for all the objects in the same class. They are also called static member variables.

To clearly describe class variables, let’s go back to our Car class example. Suppose that our Car class has one class variable called Count. If we change the value of Count to 2, all of the objects of the Car class will have the value 2 for their Count variable.

<table>
<thead>
<tr>
<th>Car Class</th>
<th>Object Car A</th>
<th>Object Car B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plate Number</td>
<td>ABC 111</td>
<td>XYZ 123</td>
</tr>
<tr>
<td>Color</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Mitsubishi</td>
<td>Toyota</td>
</tr>
<tr>
<td>Current Speed</td>
<td>50 km/h</td>
<td>100 km/h</td>
</tr>
<tr>
<td>Class Variable</td>
<td>Count = 2</td>
<td></td>
</tr>
<tr>
<td>Instance Methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accelerate Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turn Method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brake Method</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 19: Car class’ methods and variables*
8.3.4 Class Instantiation

To create an object or an instance of a class, we use the `new` operator. For example, if you want to create an instance of the class String, we write the following code,

```java
String str2 = new String(“Hello world!”);
```

or also equivalent to,

```java
String str2 = “Hello”;
```

The new operator allocates a memory for that object and returns a reference of that memory location to you. When you create an object, you actually invoke the class' constructor. The constructor is a method where you place all the initializations, it has the same name as the class.
8.4 Methods

8.4.1 What are Methods and Why Use Methods?

In the examples we discussed before, we only have one method, and that is the main() method. In Java, we can define many methods which we can call from different methods.

A method is a separate piece of code that can be called by a main program or any other method to perform some specific function.

The following are characteristics of methods:
• It can return one or no values
• It may accept as many parameters it needs or no parameter at all. Parameters are also called function arguments.
• After the method has finished execution, it goes back to the method that called it.

Now, why do we need to create methods? Why don't we just place all the code inside one big method? The heart of effective problem solving is in problem decomposition. We can do this in Java by creating methods to solve a specific part of the problem. Taking a problem and breaking it into small, manageable pieces is critical to writing large programs.
8.4.2 Calling Instance Methods and Passing Variables

Now, to illustrate how to call methods, let’s use the String class as an example. You can use the Java API documentation to see all the available methods in the String class. Later on, we will create our own methods, but for now, let us use what is available.

To call an **instance method**, we write the following,

```
nameOfObject.nameOfMethod( parameters );
```

Let’s take two sample methods found in the class String,

<table>
<thead>
<tr>
<th>Method declaration</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>public char charAt(int index)</td>
<td>Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.</td>
</tr>
<tr>
<td>public boolean equalsIgnoreCase(String anotherString)</td>
<td>Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.</td>
</tr>
</tbody>
</table>

Using the methods,

```java
String str1 = "Hello";
char x = str2.charAt(0); //will return the character H
//and store it to variable x

String str2 = "hello";

//this will return a boolean value true
boolean result = str1.equalsIgnoreCase(str1);
```
8.4.3 Passing Variables in Methods

In our examples, we already tried passing variables to methods. However, we haven't differentiated between the different types of variable passing in Java. There are two types of passing data to methods, the first one is pass-by-value and then, pass-by-reference.

8.4.3.1 Pass-by-value

When a pass-by-value occurs, the method makes a copy of the value of the variable passed to the method. The method cannot accidentally modify the original argument even if it modifies the parameters during calculations.

For example,

```java
public class TestPassByValue {
    public static void main( String[] args ){
        int i = 10;
        //print the value of i
        System.out.println( i );

        //call method test
        //and pass i to method test
        test( i );

        //print the value of i. i not changed
        System.out.println( i );
    }

    public static void test( int j ){
        //change value of parameter j
        j = 33;
    }
}
```

In the given example, we called the method test and passed the value of i as parameter. The value of i is copied to the variable of the method j. Since j is the variable changed in the test method, it will not affect the variable value if i in main since it is a different copy of the variable.

By default, all primitive data types when passed to a method are pass-by-value.
8.4.3.2 Pass-by-reference

When a pass-by-reference occurs, the reference to an object is passed to the calling method. This means that, the method makes a copy of the reference of the variable passed to the method. However, unlike in pass-by-value, the method can modify the actual object that the reference is pointing to, since, although different references are used in the methods, the location of the data they are pointing to is the same.

For example,

```java
class TestPassByReference {
    public static void main( String[] args ) {
        // create an array of integers
        int[] ages = {10, 11, 12};

        // print array values
        for( int i=0; i<ages.length; i++ ) {
            System.out.println( ages[i] );
        }

        // call test and pass reference to array
        test( ages );

        // print array values again
        for( int i=0; i<ages.length; i++ ) {
            System.out.println( ages[i] );
        }
    }

    public static void test( int[] arr ) {
        // change values of array
        for( int i=0; i<arr.length; i++ ) {
            arr[i] = i + 50;
        }
    }
}
```

Pass ages as parameter which is copied to variable arr

Pass-by-reference occurs, the reference to an object is passed to the calling method.
**Coding Guidelines:**

A common misconception about pass-by-reference in Java is when creating a swap method using Java references. Take note that Java manipulates objects 'by reference,' but it passes object references to methods 'by value.'” As a result, you cannot write a standard swap method to swap objects.

### 8.4.4 Calling Static Methods

Static methods are methods that can be invoked without instantiating a class (means without invoking the new keyword). Static methods belongs to the class as a whole and not to a certain instance (or object) of a class. Static methods are distinguished from instance methods in a class definition by the keyword static.

To call a static method, just type,

```java
Classname.staticMethodName(params);
```

Examples of static methods, we've used so far in our examples are,

```java
//prints data to screen
System.out.println("Hello world");

//converts the String 10, to an integer
int i = Integer.parseInt("10");

//Returns a String representation of the integer argument as an
//unsigned integer base 16
String hexEquivalent = Integer.toHexString( 10 );
```
8.4.5 Scope of a variable

In addition to a variable's data type and name, a variable has scope. The **scope** determines where in the program the variable is accessible. The scope also determines the lifetime of a variable or how long the variable can exist in memory. The scope is determined by where the variable declaration is placed in the program.

To simplify things, just think of the scope as anything between the curly braces {...}. The outer curly braces is called the **outer** blocks, and the inner curly braces is called **inner** blocks.

If you declare variables in the outer block, they are visible (i.e. usable) by the program lines inside the inner blocks. However, if you declare variables in the inner block, you cannot expect the outer block to see it.

A variable's scope is inside the block where it is declared, starting from the point where it is declared, and in the inner blocks.

For example, given the following code snippet,

```java
public class ScopeExample
{
    public static void main( String[] args ){
        int i = 0;
        int j = 0;

        //... some code here
        {
            int k = 0;
            int m = 0;
            int n = 0;
        }
    }
}
```

The code we have here represents five scopes indicated by the lines and the letters representing the scope. Given the variables i, j, k, m and n, and the five scopes A, B, C, D and E, we have the following scopes for each variable:

- The scope of variable i is A.
- The scope of variable j is B.
- The scope of variable k is C.
- The scope of variable m is D.
- The scope of variable n is E.
Now, given the two methods main and test in our previous examples,

```java
class TestPassByReference {
    public static void main( String[] args ) {
        //create an array of integers
        int [] ages = {10, 11, 12};

        //print array values
        for ( int i=0; i<ages.length; i++ ) {
            System.out.println( ages[i] );
        }

        //call test and pass reference to array
        test( ages );

        //print array values again
        for ( int i=0; i<ages.length; i++ ) {
            System.out.println( ages[i] );
        }
    }

    public static void test( int[] arr ) {
        //change values of array
        for ( int i=0; i<arr.length; i++ ) {
            arr[i] = i + 50;
        }
    }
}
```

In the main method, the scope of the variables are,

- `ages[]` - scope A
- `i` in B - scope B
- `i` in C - scope C

In the test method, the scope of the variables are,

- `arr[]` - scope D
- `i` in E - scope E
When declaring variables, only one variable with a given identifier or name can be declared in a scope. That means that if you have the following declaration,

```
{
    int test = 10;
    int test = 20;
}
```

your compiler will generate an error since you should have unique names for your variables in one block. However, you can have two variables of the same name, if they are not declared in the same block. For example,

```
int test = 0;
System.out.print( test );
//...some code here
{
    int test = 20;
    System.out.print( test );
}
```

When the first System.out.print is invoke, it prints the value of the first test variable since it is the variable seen at that scope. For the second System.out.print, the value 20 is printed since it is the closest test variable seen at that scope.

**Coding Guidelines:**

Avoid having variables of the same name declared inside one method to avoid confusion.
8.5 Casting, Converting and Comparing Objects

In this section, we are going to learn how to do typecasting. Typecasting or casting is the process of converting a data of a certain data type to another data type. We will also learn how to convert primitive data types to objects and vice versa. And finally, we are going to learn how to compare objects.

8.5.1 Casting Primitive Types

Casting between primitive types enables you to convert the value of one data from one type to another primitive type. This commonly occurs between numeric types.

There is one primitive data type that we cannot do casting though, and that is the boolean data type.

An example of typecasting is when you want to store an integer data to a variable of data type double. For example,

```java
int numInt = 10;
double numDouble = numInt; //implicit cast
```

In this example, since the destination variable (double) holds a larger value than what we will place inside it, the data is implicitly casted to data type double.

Another example is when we want to typecast an int to a char value or vice versa. A character can be used as an int because each character has a corresponding numeric code that represents its position in the character set. If the variable i has the value 65, the cast (char)i produces the character value 'A'. The numeric code associated with a capital A is 65, according to the ASCII character set, and Java adopted this as part of its character support. For example,

```java
char valChar = 'A';
int valInt = valChar;
System.out.print(valInt); //explicit cast: output 65
```
When we convert a data that has a large type to a smaller type, we must use an **explicit cast**. Explicit casts take the following form:

```
(dataType)value
```

where,

- `dataType`, is the name of the data type you're converting to
- `value`, is an expression that results in the value of the source type.

For example,

```java
double   valDouble = 10.12;
int      valInt = (int)valDouble;  //convert valDouble to int type

double   x = 10.2;
int      y = 2;

int      result = (int)(x/y);  //typecast result of operation to int
```
Instances of classes also can be cast into instances of other classes, with one restriction: The source and destination classes must be related by inheritance; one class must be a subclass of the other. We’ll cover more about inheritance later.

Analogous to converting a primitive value to a larger type, some objects might not need to be cast explicitly. In particular, because a subclass contains all the same information as its superclass, you can use an instance of a subclass anywhere a superclass is expected.

For example, consider a method that takes two arguments, one of type Object and another of type Window. You can pass an instance of any class for the Object argument because all Java classes are subclasses of Object. For the Window argument, you can pass in its subclasses, such as Dialog, FileDialog, and Frame. This is true anywhere in a program, not just inside method calls. If you had a variable defined as class Window, you could assign objects of that class or any of its subclasses to that variable without casting.

There is a catch, however: Because subclasses contain more behavior than their superclasses, there’s a loss in precision involved. Those superclass objects might not have all the behavior needed to act in place of a subclass object. For example, if you have an operation that calls methods in objects of the class Integer, using an object of class Number won’t include many methods specified in Integer. Errors occur if you try to call methods that the destination object doesn’t have.

To use superclass objects where subclass objects are expected, you must cast them explicitly. You won’t lose any information in the cast, but you gain all the methods and variables that the subclass defines. To cast an object to another class, you use the same operation as for primitive types:

To cast,

\[(\text{classname})\text{object}\]

where,

- \text{classname}, is the name of the destination class
- \text{object}, is a reference to the source object.
• **Note:** that casting creates a reference to the old object of the type `classname`; the old object continues to exist as it did before.

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep;  // no cast needed for upward use
veep = (VicePresident)emp;  // must cast explicitly
```

The following example casts an instance of the class `VicePresident` to an instance of the class `Employee`; `VicePresident` is a subclass of `Employee` with more information, which here defines that the `VicePresident` has executive washroom privileges,

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep;  // no cast needed for upward use
veep = (VicePresident)emp;  // must cast explicitly
```
8.5.3 Converting Primitive Types to Objects and Vice Versa

One thing you can't do under any circumstance is cast from an object to a primitive data type, or vice versa. Primitive types and objects are very different things in Java, and you can't automatically cast between the two or use them interchangeably.

As an alternative, the java.lang package includes classes that correspond to each primitive data type: Float, Boolean, Byte, and so on. Most of these classes have the same names as the data types, except that the class names begin with a capital letter (Short instead of short, Double instead of double, and the like). Also, two classes have names that differ from the corresponding data type: Character is used for char variables and Integer for int variables. (Called Wrapper Classes)

Java treats the data types and their class versions very differently, and a program won't compile successfully if you use one when the other is expected.

Using the classes that correspond to each primitive type, you can create an object that holds the same value.

Examples:

```java
// The following statement creates an instance of the Integer class with the integer value 7801 (primitive -> Object)
Integer dataCount = new Integer(7801);

// The following statement converts an Integer object to its primitive data type int. The result is an int with value 7801
int newCount = dataCount.intValue();

// A common translation you need in programs
// is converting a String to a numeric type, such as an int
// Object->primitive
String pennsylvania = "65000";
int penn = Integer.parseInt(pennsylvania);
```

- **CAUTION:** The Void class represents nothing in Java, so there's no reason it would be used when translating between primitive values and objects. It's a placeholder for the void keyword, which is used in method definitions to indicate that the method does not return a value.
### 8.5.4 Comparing Objects

In our previous discussions, we learned about operators for comparing values—equal, not equal, less than, and so on. Most of these operators work only on primitive types, not on objects. If you try to use other values as operands, the Java compiler produces errors.

The exceptions to this rule are the operators for equality: `==` (equal) and `!=` (not equal). When applied to objects, these operators don’t do what you might first expect. Instead of checking whether one object has the same value as the other object, they determine whether both sides of the operator refer to the same object.

To compare instances of a class and have meaningful results, you must implement special methods in your class and call those methods. A good example of this is the String class.

It is possible to have two different String objects that contain the same values. If you were to employ the `==` operator to compare these objects, however, they would be considered unequal. Although their contents match, they are not the same object.

To see whether two String objects have matching values, a method of the class called `equals()` is used. The method tests each character in the string and returns true if the two strings have the same values.

The following code illustrates this,

```java
class EqualsTest {
    public static void main(String[] arguments) {
        String str1, str2;
        str1 = "Free the bound periodicals."
        str2 = str1;

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));

        str2 = new String(str1);

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));
        System.out.println("Same value? " + str1.equals(str2));
    }
}
```

This program's output is as follows,

**OUTPUT:**

String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? True
Now let's discuss the code.

```java
String str1, str2;
str1 = "Free the bound periodicals.";

str2 = new String(str1);
```

The first part of this program declares two variables (str1 and str2), assigns the literal "Free the bound periodicals." to str1, and then assigns that value to str2. As you learned earlier, str1 and str2 now point to the same object, and the equality test proves that.

In the second part of this program, you create a new String object with the same value as str1 and assign str2 to that new String object. Now you have two different string objects in str1 and str2, both with the same value. Testing them to see whether they're the same object by using the == operator returns the expected answer: false—they are not the same object in memory. Testing them using the equals() method also returns the expected answer: true—they have the same values.

- **NOTE:** Why can't you just use another literal when you change str2, rather than using new? String literals are optimized in Java; if you create a string using a literal and then use another literal with the same characters, Java knows enough to give you the first String object back. Both strings are the same objects; you have to go out of your way to create two separate objects.
8.5.5 Determining the Class of an Object

Want to find out what an object’s class is? Here's the way to do it for an object assigned to the variable key:

1. The `getClass()` method returns a Class object (where Class is itself a class) that has a method called `getName()`. In turn, `getName()` returns a string representing the name of the class.

   For Example,
   
   ```java
   String name = key.getClass().getName();
   ```

2. The `instanceOf` operator

   The `instanceOf` has two operands: a reference to an object on the left and a class name on the right. The expression returns true or false based on whether the object is an instance of the named class or any of that class’s subclasses.

   For Example,
   
   ```java
   boolean ex1 = "Texas" instanceof String; // true
   Object pt = new Point(10, 10);
   boolean ex2 = pt instanceof String; // false
   ```
8.6 Exercises

8.6.1 Defining terms

In your own words, define the following terms:
1. Class
2. Object
3. Instantiate
4. Instance Variable
5. Instance Method
6. Class Variables or static member variables
7. Constructor

8.6.2 Java Scavenger Hunt

Pipoy is a newbie in the Java programming language. He just heard that there are already ready-to-use APIs in Java that one could use in their programs, and he’s eager to try them out. The problem is, Pipoy does not have a copy of the Java Documentation, and he also doesn’t have an internet access, so there’s no way for him to view the Java APIs.

Your task is to help Pipoy look for the APIs (Application Programming Interface). You should state the class where the method belongs, the method declaration and a sample usage of the said method.

For example, if Pipoy wants to know the method that converts a String to integer, your answer should be:

Class: Integer
Method Declaration: public static int parseInt( String value )
Sample Usage:
String strValue = "100";
int value = Integer.parseInt( strValue );

Make sure that the snippet of code you write in your sample usage compiles and outputs the correct answer, so as not to confuse Pipoy. (Hint: All methods are in the java.lang package). In cases where you can find more methods that can accomplish the task, give only one.

Now let’s start the search!
1. Look for a method that checks if a certain String ends with a certain suffix. For example, if the given string is "Hello", the method should return true the suffix given is "lo", and false if the given suffix is "alp".
2. Look for the method that determines the character representation for a specific digit in the specified radix. For example, if the input digit is 15, and the radix is 16, the method would return the character F, since F is the hexadecimal representation for the number 15 (base 10).
3. Look for the method that terminates the currently running Java Virtual Machine
4. Look for the method that gets the floor of a double value. For example, if I input a 3.13, the method should return the value 3.
5. Look for the method that determines if a certain character is a digit. For example, if I input '3', it returns the value true.
9 Creating your own Classes

9.1 Objectives

Now that we've studied on how to use existing classes from the Java class library, we will now be studying on how to write our own classes. For this section, in order to easily understand how to create classes, we will make a sample class wherein we will add more data and functionality as we go along the way.

We will create a class that contains information of a Student and operations needed for a certain student record.

Things to take note of for the syntax defined in this section and for the other sections:

* - means that there may be 0 or more occurrences of the line where it was applied to.
<description> - indicates that you have to substitute an actual value for this part instead of typing it as it is.
[] - indicates that this part is optional

At the end of the lesson, the student should be able to:

- Create their own classes
- Declare attributes and methods for their classes
- Use the this reference to access instance data
- Create and call overloaded methods
- Import and create packages
- Use access modifiers to control access to class members
9.2 Defining your own classes

Before writing your class, think first on where you will be using your class and how your class will be used. Think of an appropriate name for the class, and list all the information or properties that you want your class to contain. Also list down the methods that you will be using for your class.

To define a class, we write,

```java
<modifier> class <name> {
    <attributeDeclaration>*
    <constructorDeclaration>*
    <methodDeclaration>*
}
```

where

- `<modifier>` is an access modifier, which may be combined with other types of modifier.

**Coding Guidelines:**

Remember that for a top-level class, the only valid access modifiers are public and package (i.e., if no access modifier prefixes the class keyword).

In this section, we will be creating a class that will contain a student record. Since we've already identified the purpose of our class, we can now name it. An appropriate name for our class would be `StudentRecord`.

Now, to define our class we write,

```java
public class StudentRecord {
    //we'll add more code here later
}
```

where,

- public - means that our class is accessible to other classes outside the package
- class - this is the keyword used to create a class in Java
- StudentRecord - a unique identifier that describes our class

**Coding Guidelines:**

1. Think of an appropriate name for your class. Don't just call your class XYZ or any random names you can think of.
2. Class names should start with a CAPITAL letter.
3. The filename of your class should have the SAME NAME as your public class name.
### 9.3 Declaring Attributes

To declare a certain attribute for our class, we write,

```
<modifier> <type> <name> [= <default_value>];
```

Now, let us write down the list of attributes that a student record can contain. For each information, also list what data types would be appropriate to use. For example, you don’t want to have a data type int for a student's name, or a String for a student's grade.

The following are some sample information we want to add to the student record.

```
name - String
address - String
age - int
math grade - double
english grade - double
science grade - double
average grade - double
```

You can add more information if you want to, it's all really up to you. But for this example, we will be using these information.

#### 9.3.1 Instance Variables

Now that we have a list of all the attributes we want to add to our class, let us now add them to our code. Since we want these attributes to be unique for each object (or for each student), we should declare them as instance variables.

For example,

```java
public class StudentRecord {
    private String name;
    private String address;
    private int age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;
    //we'll add more code here later
}
```

where,

- `private` here means that the variables are only accessible within the class. Other objects cannot access these variables directly. We will cover more about accessibility later.

**Coding Guidelines:**

1. Declare all your instance variables on the top of the class declaration.
2. Declare one variable for each line.
3. Instance variables, like any other variables should start with a SMALL letter.
4. Use an appropriate data type for each variable you declare.
5. Declare instance variables as private so that only class methods can access them directly.
9.3.2 Class Variables or Static Variables

Aside from instance variables, we can also declare class variables or variables that belong to the class as a whole. The value of these variables are the same for all the objects of the same class. Now suppose, we want to know the total number of student records we have for the whole class, we can declare one static variable that will hold this value. Let us call this as studentCount.

To declare a static variable,

```java
public class StudentRecord {
    //instance variables we have declared
    private static int studentCount;
    //we'll add more code here later
}
```

we use the keyword static to indicate that a variable is a static variable.

So far, our whole code now looks like this.

```java
public class StudentRecord {
    private String name;
    private String address;
    private int age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;
    private static int studentCount;
    //we'll add more code here later
}
```

9.4 Declaring Methods

Before we discuss what methods we want our class to have, let us first take a look at the general syntax for declaring methods.

To declare methods we write,

```java
<modifier> <returnType> <name>(<parameter>*) {
    <statement>*
}
```

where,
<modifier> can carry a number of different modifiers
$returnType$ can be any data type (including void)
$name$ can be any valid identifier
$parameter$ ::= $parameter_type$ <parameter_name>[,]
9.4.1 Accessor methods

In order to implement encapsulation, that is, we don't want any objects to just access our data anytime, we declare the fields or attributes of our classes as private. However, there are times wherein we want other objects to access private data. In order to do this, we create accessor methods.

**Accessor methods** are used to read values from class variables (instance/static). An accessor method usually starts with a `get<NameOfInstanceVariable>`. It also returns a value.

For our example, we want an accessor method that can read the name, address, english grade, math grade and science grade of the student.

Now let's take a look at one implementation of an accessor method,

```java
public class StudentRecord
{
    private String name;
    :
    :
    public String getName()
    {
        return name;
    }
}
```

where,

- **public** - means that the method can be called from objects outside the class
- **String** - is the return type of the method. This means that the method should return a value of type String
- **getName()** - the name of the method
- **()** - this means that our method does not have any parameters

The statement,

```
return name;
```

in our program signifies that it will return the value of the instance variable name to the calling method. Take note that the return type of the method should have the same data type as the data in the return statement. You usually encounter the following error if the two does not have the same data type,

```
StudentRecord.java:14: incompatible types
found    : int
required: java.lang.String
    return age;
^ 1 error
```
Another example of an accessor method is the getAverage method,

```java
public class StudentRecord {
    private String name;

    public double getAverage(){
        double result = 0;
        result = (mathGrade+englishGrade+scienceGrade)/3;
        return result;
    }
}
```

The getAverage method computes the average of the 3 grades and returns the result.

### 9.4.2 Mutator Methods

Now, what if we want other objects to alter our data? What we do is we provide methods that can write or change values of our class variables (instance/static). We call these methods, **mutator methods**. A mutator method is usually written as `set<NameOfInstanceVariable>`.

Now let's take a look at one implementation of a mutator method,

```java
public class StudentRecord {
    private String name;

    public void setName( String temp ){
        name = temp;
    }
}
```

where,

- `public` - means that the method can be called from objects outside the class
- `void` - means that the method does not return any value
- `setName` - the name of the method
- `(String temp)` - parameter that will be used inside our method

The statement,

```java
    name = temp;
```

assigns the value of temp to name and thus changes the data inside the instance variable name.

Take note that mutator methods don't return values. However, it contains some program argument or arguments that will be used inside the method.
9.4.3 Multiple Return statements

You can have multiple return statements for a method as long as they are not on the same block. You can also use constants to return values instead of variables.

For example, consider the method,

```java
public String getNumberInWords(int num) {
    String defaultNum = "zero";
    if (num == 1) {
        return "one";  // return a constant
    }
    else if (num == 2) {
        return "two";  // return a constant
    }
    // return a variable
    return defaultNum;
}
```

9.4.4 Static methods

For the static variable studentCount, we can create a static method to access its value.

```java
public class StudentRecord {
    private static int studentCount;
    public static int getStudentCount() {
        return studentCount;
    }
}
```

where,
- **public** - means that the method can be called from objects outside the class
- **static** - means that the method is static and should be called by typing,[ClassName].[methodName]. For example, in this case, we call the method `StudentRecord.getStudentCount()`
- **int** - is the return type of the method. This means that the method should return a value of type `int`
- **getStudentCount()** - the name of the method
- **()** - this means that our method does not have any parameters

For now, `getStudentCount` will always return the value zero since we haven’t done anything yet in our program in order to set its value. We will try to change the value of studentCount later on when we discuss constructors.

**Coding Guidelines:**

1. Method names should start with a SMALL letter.
2. Method names should be verbs
3. Always provide documentation before the declaration of the method. You can use javadocs style for this. Please see example.
9.4.5 Sample Source Code for StudentRecord class

Here is the code for our StudentRecord class,

```java
public class StudentRecord
{
    private String name;
    private String address;
    private int age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;

    private static int studentCount;

    /**
     * Returns the name of the student
     */
    public String getName()
    {
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName( String temp )
    {
        name = temp;
    }

    // other code here ....

    /**
     * Computes the average of the english, math and science
     * grades
     */
    public double getAverage()
    {
        double result = 0;
        result = ( mathGrade+englishGrade+scienceGrade )/3;
        return result;
    }

    /**
     * returns the number of instances of StudentRecords
     */
    public static int getStudentCount()
    {
        return studentCount;
    }
}
```
Now, here's a sample code of a class that uses our StudentRecord class.

```java
public class StudentRecordExample {
    public static void main( String[] args ){
        //create three objects for Student record
        StudentRecord annaRecord = new StudentRecord();
        StudentRecord beahRecord = new StudentRecord();
        StudentRecord crisRecord = new StudentRecord();

        //set the name of the students
        annaRecord.setName("Anna");
        beahRecord.setName("Beah");
        crisRecord.setName("Cris");

        //print anna's name
        System.out.println( annaRecord.getName() );

        //print number of students
        System.out.println("Count="+StudentRecord.getStudentCount());
    }
}
```

The output of this program is,

```
Anna
Student Count = 0
```
9.5 The this reference

The this reference is used to access the instance variables shadowed by the parameters. To understand this better, let's take for example the setAge method. Suppose we have the following declaration for setAge.

```java
public void setAge( int age ){
    age = age; //WRONG!!!
}
```

The parameter name in this declaration is age, which has the same name as the instance variable age. Since the parameter age is the closest declaration to the method, the value of the parameter age will be used. So in the statement,

```java
age = age;
```

we are just assigning the value of the parameter age to itself! This is not what we want to happen in our code. In order to correct this mistake, we use the this reference. To use the this reference, we type,

```java
this.<nameOfTheInstanceVariable>
```

So for example, we can now rewrite our code to,

```java
public void setAge( int age ){
    this.age = age;
}
```

This method will then assign the value of the parameter age to the instance variable of the object StudentRecord.

**NOTE: You can only use the this reference for instance variables and NOT static or class variables.**
9.6 Overloading Methods

In our classes, we want to sometimes create methods that has the same names but function differently depending on the parameters that are passed to them. This capability is possible in Java, and it is called **Method Overloading**.

**Method overloading** allows a method with the same name but different parameters, to have different implementations and return values of different types. Rather than invent new names all the time, method overloading can be used when the same operation has different implementations.

For example, in our StudentRecord class we want to have a method that prints information about the student. However, we want the print method to print things differently depending on the parameters we pass to it. For example, when we pass a String, we want the print method to print out the name, address and age of the student. When we pass 3 double values, we want the method to print the student's name and grades.

We have the following overloaded methods inside our StudentRecord class,

```java
public void print(String temp)
{
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}

public void print(double eGrade, double mGrade, double sGrade)
{
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
    System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

```
When we try to call this in the following main method,

```java
public static void main( String[] args )
{
    StudentRecord annaRecord = new StudentRecord();
    annaRecord.setName("Anna");
    annaRecord.setAddress("Philippines");
    annaRecord.setAge(15);
    annaRecord.setMathGrade(80);
    annaRecord.setEnglishGrade(95.5);
    annaRecord.setScienceGrade(100);
    //overloaded methods
    annaRecord.print( annaRecord.getName() );
    annaRecord.print( annaRecord.getEnglishGrade(),
                       annaRecord.getMathGrade(),
                       annaRecord.getScienceGrade());
}
```

we will have the output for the first call to print,

```
Name:Anna
Address:Philippines
Age:15
```

we will have the output for the second call to print,

```
Name:Anna
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```

Always remember that overloaded methods have the following properties,

- the same name
- **different parameters**
- return types can be different or the same
9.7 Declaring Constructors

We have discussed before the concept of constructors. Constructors are important in instantiating an object. It is a method where all the initializations are placed.

The following are the properties of a constructor:
1. Constructors have the same name as the class
2. A constructor is just like an ordinary method, however only the following information can be placed in the header of the constructor, scope or accessibility identifier (like public...), constructor's name and parameters if it has any.
3. Constructors does not have any return value
4. You cannot call a constructor directly, it can only be called by using the new operator during class instantiation.

To declare a constructor, we write,

```java
<modifier> <className> (<parameter>*) {
<statement>*
}
```

9.7.1 Default Constructor

Every class has a default constructor. The default constructor is the constructor without any parameters. If the class does not specify any constructors, then an implicit default constructor is created.

For example, in our StudentRecord class, the default constructor would look like this,

```java
public StudentRecord()
{
    //some code here
}
```

9.7.2 Overloading Constructors

As we have mentioned, constructors can also be overloaded, for example, we have here four overloaded constructors,

```java
public StudentRecord(){
    //some initialization code here
}

public StudentRecord(String temp){
    this.name = temp;
}

public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
}

public StudentRecord(double mGrade, double eGrade, 
                        double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
9.7.3 Using Constructors

To use these constructors, we have the following code,

```java
public static void main( String[] args )
{
    //create three objects for Student record
    StudentRecord annaRecord=new StudentRecord("Anna");
    StudentRecord beahRecord=new StudentRecord("Beah", "Philippines");
    StudentRecord crisRecord=new StudentRecord(80,90,100);
    //some code here
}
```

Now, before we move on, let us go back to the static variable studentCount we have declared a while ago. The purpose of the studentCount is to count the number of objects that are instantiated with the class StudentRecord. So, what we want to do here is, everytime an object of class StudentRecord is instantiated, we increment the value of studentCount. A good location to modify and increment the value of studentCount is in the constructors, because it is always called everytime an object is instantiated. For example,

```java
public StudentRecord(){
    //some initialization code here
    studentCount++; //add a student
}

public StudentRecord(String temp){
    this.name = temp;
    studentCount++; //add a student
}

public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
    studentCount++; //add a student
}

public StudentRecord(double mGrade, double eGrade,
                      double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
    studentCount++; //add a student
}
```
9.7.4 The this() Constructor Call

Constructor calls can be chained, meaning, you can call another constructor from inside another constructor. We use the **this** call for this. For example, given the following code,

```java
public StudentRecord()
    { this("some string"); }

public StudentRecord(String temp)
    { this.name = temp; }

public static void main( String[] args )
    { StudentRecord       annaRecord = new StudentRecord();
```

Given the code above, when the statement at line 13 is called, it will call the default constructor line 1. When statement in line 2 is executed, it will then call the constructor that has a String parameter (in line 6).

There are a few things to remember when using the **this** constructor call:

1. When using the this constructor call, **IT MUST OCCUR AS THE FIRST STATEMENT in a constructor**
2. It can **ONLY BE USED IN A CONSTRUCTOR DEFINITION**. The this call can then be followed by any other relevant statements.
9.8 Packages

Packages are Java’s means of grouping related classes and interfaces together in a single unit (interfaces will be discussed later). This powerful feature provides for a convenient mechanism for managing a large group of classes and interfaces while avoiding potential naming conflicts.

9.8.1 Importing Packages

To be able to use classes outside of the package you are currently working in, you need to import the package of those classes. By default, all your Java programs import the java.lang.* package, that is why you can use classes like String and Integers inside the program even though you haven't imported any packages.

The syntax for importing packages is as follows,

```
import <nameOfPackage>
```

For example, if you want to use the class Color inside package awt, you have to type the following,

```
import java.awt.Color;
import java.awt.*
```

The first statement imports the specific class Color while the other imports all of the classes in the java.awt package.

Another way to import classes from other packages is through explicit package referencing. This is done by using the package name to declare an object of a class.

```
java.awt.Color color;
```

9.8.2 Creating your own packages

To create your own package, we write,

```
package <packageName>;
```

Suppose we want to create a package where we will place our StudentRecord class, together with other related classes. We will call our package, schoolClasses.

The first thing you have to do is create a folder named schoolClasses. Copy all the classes that you want to belong to this package inside this folder. After copying, add the following code at the top of the class file. For example,

```
package schoolClasses;

public class StudentRecord
{
    private String name;
    private String address;
    private int age;
};
```

Packages can also be nested. In this case, the Java interpreter expects the directory structure containing the executable classes to match the package hierarchy.
9.8.3 Setting the CLASSPATH

Now, suppose we place the package `schoolClasses` under the C:\ directory. We need to set the classpath to point to that directory so that when we try to run it, the JVM will be able to see where our classes are stored.

Before we discuss how to set the classpath, let us take a look at an example on what will happen if we don’t set the classpath.

Suppose we compile and then run the `StudentRecord` class we wrote in the last section,

```
C:\schoolClasses> javac StudentRecord.java

C:\schoolClasses> java StudentRecord
Exception in thread "main" java.lang.NoClassDefFoundError:
  StudentRecord (wrong name: schoolClasses/StudentRecord)
  at java.lang.ClassLoader.defineClass1(Native Method)
  at java.lang.ClassLoader.defineClass(Unknown Source)
  at java.security.SecureClassLoader.defineClass(Unknown Source)
  at java.net.URLClassLoader.defineClass(Unknown Source)
  at java.net.URLClassLoader.access$100(Unknown Source)
  at java.net.URLClassLoader$1.run(Unknown Source)
  at java.security.AccessController.doPrivileged(Native Method)
  at java.net.URLClassLoader.findClass(Unknown Source)
  at java.lang.ClassLoader.loadClass(Unknown Source)
  at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
  at java.lang.ClassLoader.loadClass(Unknown Source)
  at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

We encounter a `NoClassDefFoundError` which means that Java did not know where to look for your class. The reason for this is that your class `StudentRecord` now belongs to a package named `studentClasses`. If we want to run our class, we have to tell Java about its full class name which is `schoolClasses.StudentRecord`. We also have to tell JVM where to look for our packages, which in this case is in location C:\. To do this, we must set the classpath.

To set the classpath in Windows, we type this at the command prompt,

```
C:\schoolClasses> set classpath=C:\
```

where C:\ is the directory in which we have placed the packages. After setting the classpath, we can now run our program anywhere by typing,

```
C:\schoolClasses> java schoolClasses.StudentRecord
```

For Unix base systems, suppose we have our classes in the directory `/usr/local/myClasses`, we write,

```
export classpath=/usr/local/myClasses
```

We encounter a `NoClassDefFoundError` which means that Java did not know where to look for your class. The reason for this is that your class `StudentRecord` now belongs to a package named `studentClasses`. If we want to run our class, we have to tell Java about its full class name which is `schoolClasses.StudentRecord`. We also have to tell JVM where to look for our packages, which in this case is in location C:\. To do this, we must set the classpath.

To set the classpath in Windows, we type this at the command prompt,

```
C:\schoolClasses> set classpath=C:\
```

where C:\ is the directory in which we have placed the packages. After setting the classpath, we can now run our program anywhere by typing,

```
C:\schoolClasses> java schoolClasses.StudentRecord
```

For Unix base systems, suppose we have our classes in the directory `/usr/local/myClasses`, we write,

```
export classpath=/usr/local/myClasses
```
Take note that you can set the classpath anywhere. You can also set more than one classpath, we just have to separate them by ; (for windows) and : (for Unix based systems). For example,

```
set classpath=C:\myClasses;D:\;E:\MyPrograms\Java
```

and for Unix based systems,

```
export classpath=/usr/local/java:/usr/myClasses
```
9.9 Access Modifiers

When creating our classes and defining the properties and methods in our class, we want to implement some kind of restriction to access these data. For example, if you want a certain attribute to be changed only by the methods inside the class, you may want to hide this from other objects using your class. In Java, we have what we call access modifiers in order to implement this.

There are four different types of member access modifiers in Java: public, private, protected and default. The first three access modifiers are explicitly written in the code to indicate the access type, for the fourth one which is default, no keyword is used.

9.9.1 default access (also called package accessibility)

This specifies that only classes in the same package can have access to the class' variables and methods. There are no actual keyword for the default modifier; it is applied in the absence of an access modifier. For example,

```java
public class StudentRecord {

    //default access to instance variable
    int name;

    //default access to method
    String getName(){
        return name;
    }
}
```

In this example, the instance variable name and the method getName() can be accessed from other objects, as long as the object belongs to the same package where the class StudentRecord belongs to.

9.9.2 public access

This specifies that class members are accessible to anyone, both inside and outside the class. Any object that interacts with the class can have access to the public members of the class. For example,

```java
public class StudentRecord {

    //default access to instance variable
    public int name;

    //default access to method
    public String getName(){
        return name;
    }
}
```

In this example, the instance variable name and the method getName() can be accessed from other objects.
9.9.3 protected access

This specifies that the class members are accessible only to methods in that class and the subclasses of the class. For example,

```java
public class StudentRecord
{
    //default access to instance variable
    protected int name;

    //default access to method
    protected String getName()
    {
        return name;
    }
}
```

In this example, the instance variable name and the method getName() can be accessed only from methods inside the class and from subclasses of StudentRecord. We will discuss about subclasses on the next chapter.

9.9.4 private access

This specifies that the class members are only accessible by the class they are defined in. For example,

```java
public class StudentRecord
{
    //default access to instance variable
    private int name;

    //default access to method
    private String getName()
    {
        return name;
    }
}
```

In this example, the instance variable name and the method getName() can be accessed only from methods inside the class.

**Coding Guidelines:**
The instance variables of a class should normally be declared private, and the class will just provide accessor and mutator methods to these variables.
9.10 Exercises

9.10.1 Address Book Entry

Your task is to create a class that contains an address book entry. The following table describes the information that an addressbook entry has.

<table>
<thead>
<tr>
<th>Attributes/Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the person in the addressbook</td>
</tr>
<tr>
<td>Address</td>
<td>Address of the person</td>
</tr>
<tr>
<td>Telephone Number</td>
<td>Telephone number of the person</td>
</tr>
<tr>
<td>Email Address</td>
<td>Person’s Email address</td>
</tr>
</tbody>
</table>

*Table 21: Attributes and Attributes Descriptions*

For the methods, create the following:
1. Provide the necessary accessor and mutator methods for all the attributes.
2. Constructors

9.10.2 AddressBook

Create a class address book that can contain 100 entries of AddressBookEntry objects (use the class you created in the first exercise). You should provide the following methods for the address book.

1. Add entry
2. Delete entry
3. View all entries
4. Update an entry


10 Inheritance, Polymorphism and Interfaces

10.1 Objectives

In this section, we will be discussing on how a class can inherit the properties of an existing class. A class that does this is called a subclass and its parent class is called the superclass. We will also be discussing a special property of Java wherein it can automatically apply the proper methods to each object regardless of what subclass the object came from. This property is known as polymorphism. Finally, we are going to discuss interfaces that helps reduce programming effort.

At the end of the lesson, the student should be able to:

- Define super classes and subclasses
- Override methods of superclasses
- Create final methods and final classes

10.2 Inheritance

In Java, all classes, including the classes that make up the Java API, are subclassed from the Object superclass. A sample class hierarchy is shown below.

Any class above a specific class in the class hierarchy is known as a superclass. While any class below a specific class in the class hierarchy is known as a subclass of that class.

Inheritance is a major advantage in object-oriented programming since once a behavior (method) is defined in a superclass, that behavior is automatically inherited by all subclasses. Thus, you can encode a method only once and they can be used by all subclasses. A subclass only need to implement the differences between itself and the parent.
Defining Superclasses and Subclasses

To derive a class, we use the `extends` keyword. In order to illustrate this, let's create a sample parent class. Suppose we have a parent class called Person.

```java
public class Person {
    protected String name;
    protected String address;

    /**
     * Default constructor
     */
    public Person() {
        System.out.println("Inside Person:Constructor");
        name = ";
        address = "
    }

    /**
     * Constructor with 2 parameters
     */
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    /**
     * Accessor methods
     */
    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String add) {
        this.address = add;
    }
}
```

Notice that, the attributes name and address are declared as `protected`. The reason we did this is that, we want these attributes to be accessible by the subclasses of the superclass. If we declare this as private, the subclasses won't be able to use them. Take note that all the properties of a superclass that are declared as `public, protected and default` can be accessed by its subclasses.
Now, we want to create another class named Student. Since a student is also a person, we decide to just extend the class Person, so that we can inherit all the properties and methods of the existing class Person. To do this, we write,

```java
public class Student extends Person {
    public Student() {
        System.out.println("Inside Student:Constructor");
        // some code here
    }
    // some code here
}
```

When a Student object is instantiated, the default constructor of its superclass is invoked implicitly to do the necessary initializations. After that, the statements inside the subclass are executed. To illustrate this, consider the following code,

```java
public static void main(String[] args) {
    Student anna = new Student();
}
```

In the code, we create an object of class Student. The output of the program is,

```
Inside Person:Constructor
Inside Student:Constructor
```

The program flow is shown below.

![Program Flow Diagram](Figure 10.2: Program Flow)
10.2.2 The super keyword

A subclass can also explicitly call a constructor of its immediate superclass. This is done by using the super constructor call. A super constructor call in the constructor of a subclass will result in the execution of relevant constructor from the superclass, based on the arguments passed.

For example, given our previous example classes Person and Student, we show an example of a super constructor call. Given the following code for Student,

```java
public Student(){
    super( "SomeName", "SomeAddress" );
    System.out.println("Inside Student:Constructor");
}
```

This code calls the second constructor of its immediate superclass (which is Person) and executes it. Another sample code shown below,

```java
public Student(){
    super();
    System.out.println("Inside Student:Constructor");
}
```

This code calls the default constructor of its immediate superclass (which is Person) and executes it.

There are a few things to remember when using the super constructor call:

1. The super() call MUST OCCUR THE FIRST STATEMENT IN A CONSTRUCTOR.
2. The super() call can only be used in a constructor definition.
3. This implies that the this() construct and the super() calls CANNOT BOTH OCCUR IN THE SAME CONSTRUCTOR.

Another use of super is to refer to members of the superclass (just like the this reference). For example,

```java
public Student()
{
    super.name = "somename";
    super.address = "some address";
}
```
10.2.3 Overriding Methods

If for some reason a derived class needs to have a different implementation of a certain method from that of the superclass, **overriding** methods could prove to be very useful. A subclass can override a method defined in its superclass by providing a new implementation for that method.

Suppose we have the following implementation for the getName method in the Person superclass,

```java
public class Person {
    :
    :
    public String getName(){
        System.out.println("Parent: getName");
        return name;
    }
    :
}
```

To override, the getName method in the subclass Student, we write,

```java
public class Student extends Person {
    :
    :
    public String getName(){
        System.out.println("Student: getName");
        return name;
    }
    :
}
```

So, when we invoke the getName method of an object of class Student, the overridden method would be called, and the output would be,

```
Student: getName
```
**10.2.4 Final Methods and Final Classes**

In Java, it is also possible to declare classes that can no longer be subclassed. These classes are called **final classes**. To declare a class to be final, we just add the final keyword in the class declaration. For example, if we want the class `Person` to be declared final, we write,

```java
public final class Person
{
    //some code here
}
```

Many of the classes in the Java API are declared final to ensure that their behavior cannot be overridden. Examples of these classes are `Integer`, `Double` and `String`.

It is also possible in Java to create methods that cannot be overridden. These methods are what we call **final methods**. To declare a method to be final, we just add the final keyword in the method declaration. For example, if we want the `getName` method in class `Person` to be declared final, we write,

```java
public final String getName(){
    return name;
}
```

Static methods are automatically final. This means that you cannot override them.
10.3 Polymorphism

Now, given the parent class Person and the subclass Student of our previous example, we add another subclass of Person which is Employee. Below is the class hierarchy for that,

![Class Hierarchy Diagram]

In Java, we can create a reference that is of type superclass to an object of its subclass. For example,

```java
public static main( String[] args )
{
    Person ref;
    Student studentObject = new Student();
    Employee employeeObject = new Employee();
    ref = studentObject;  //Person ref points to a
                         // Student object
    //some code here
}
```

Now suppose we have a getName method in our superclass Person, and we override this method in both the subclasses Student and Employee,

```java
public class Person
{
    public String getName()
    {
        System.out.println("Person Name:" + name);
        return name;
    }
}

public class Student extends Person
{
    public String getName()
    {
        System.out.println("Student Name:" + name);
        return name;
    }
}

public class Employee extends Person
{
    public String getName()
    {
        System.out.println("Employee Name:" + name);
        return name;
    }
}
```
Going back to our main method, when we try to call the getName method of the reference Person ref, the getName method of the Student object will be called. Now, if we assign ref to an Employee object, the getName method of Employee will be called.

```java
public static main( String[] args )
{
    Person    ref;
    Student    studentObject = new Student();
    Employee   employeeObject = new Employee();
    ref = studentObject; //Person reference points to a // Student object
    String temp = ref.getName(); //getName of Student //class is called
    System.out.println( temp );
    ref = employeeObject; //Person reference points to an // Employee object
    String temp = ref.getName(); //getName of Employee //class is called
    System.out.println( temp );
}
```

This ability of our reference to change behavior according to what object it is holding is called **polymorphism**. Polymorphism allows multiple objects of different subclasses to be treated as objects of a single superclass, while automatically selecting the proper methods to apply to a particular object based on the subclass it belongs to.

Another example that exhibits the property of polymorphism is when we try to pass a reference to methods. Suppose we have a static method `printInformation` that takes in a Person object as reference, we can actually pass a reference of type Employee and type Student to this method as long as it is a subclass of the class Person.

```java
public static main( String[] args )
{
    Student    studentObject = new Student();
    Employee   employeeObject = new Employee();

    printInformation( studentObject );
    printInformation( employeeObject );
}
```

```java
public static printInformation( Person p ){
    . . .
}
```
10.4 Abstract Classes

Now suppose we want to create a superclass wherein it has certain methods in it that contains some implementation, and some methods wherein we just want to be overridden by its subclasses.

For example, we want to create a superclass named LivingThing. This class has certain methods like breath, eat, sleep and walk. However, there are some methods in this superclass wherein we cannot generalize the behavior. Take for example, the walk method. Not all living things walk the same way. Take the humans for instance, we humans walk on two legs, while other living things like dogs walk on four legs. However, there are many characteristics that living things have in common, that is why we want to create a general superclass for this.

In order to do this, we can create a superclass that has some methods with implementations and others which do not. This kind of class is called an abstract class.

An abstract class is a class that cannot be instantiated. It often appears at the top of an object-oriented programming class hierarchy, defining the broad types of actions possible with objects of all subclasses of the class.

Those methods in the abstract classes that do not have implementation are called abstract methods. To create an abstract method, just write the method declaration without the body and use the abstract keyword. For example,

```java
public abstract void someMethod();
```
Now, let's create an example abstract class.

```java
public abstract class LivingThing {
    public void breath(){
        System.out.println("Living Thing breathing...");
    }
    public void eat(){
        System.out.println("Living Thing eating...");
    }

    /**
    * abstract method walk
    * We want this method to be overridden by subclasses of
    * LivingThing
    */
    public abstract void walk();
}
```

When a class extends the LivingThing abstract class, it is required to override the abstract method `walk()`, or else, that subclass will also become an abstract class, and therefore cannot be instantiated. For example,

```java
public class Human extends LivingThing {
    public void walk(){
        System.out.println("Human walks...");
    }
}
```

If the class `Human` does not override the `walk` method, we would encounter the following error message,

```
Human.java:1: Human is not abstract and does not override
abstract method walk() in LivingThing
public class Human extends LivingThing
^ 1 error
```

**Coding Guidelines:**

*Use abstract classes to define broad types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.*
10.5 Interfaces

An interface is a special kind of block containing method signatures (and possibly constants) only. Interfaces define the signatures of a set of methods without the body.

Interfaces define a standard and public way of specifying the behavior of classes. They allow classes, regardless of their location in the class hierarchy, to implement common behaviors. Note that interfaces exhibit polymorphism as well, since program may call an interface method and the proper version of that method will be executed depending on the type of object passed to the interface method call.

10.5.1 Why do we use Interfaces?

We need to use interfaces if we want unrelated classes to implement similar methods. Thru interfaces, we can actually capture similarities among unrelated classes without artificially forcing a class relationship.

Let's take as an example a class Line which contains methods that computes the length of the line and compares a Line object to objects of the same class. Now, suppose we have another class MyInteger which contains methods that compares a MyInteger object to objects of the same class. As we can see here, both of the classes have some similar methods which compares them from other objects of the same type, but they are not related whatsoever. In order to enforce a way to make sure that these two classes implement some methods with similar signatures, we can use an interface for this. We can create an interface class, let's say interface Relation which has some comparison method declarations. Our interface Relation can be declared as,

```java
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Another reason for using an object's programming interface is to reveal an object's programming interface without revealing its class. As we can see later on the section Interface vs. Classes, we can actually use an interface as data type.

Finally, we need to use interfaces to model multiple inheritance which allows a class to have more than one superclass. Multiple inheritance is not present in Java, but present in other object-oriented languages like C++.

10.5.2 Interface vs. Abstract Class

The following are the main differences between an interface and an abstract class: interface methods have no body, an interface can only define constants and an interface have no direct inherited relationship with any particular class, they are defined independently.
10.5.3 Interface vs. Class

One common characteristic of an interface and class is that they are both types. This means that an interface can be used in places where a class can be used. For example, given a class Person and an interface PersonInterface, the following declarations are valid:

```java
PersonInterface pi = new Person();
Person pc = new Person();
```

However, you cannot create an instance from an interface. An example of this is:

```java
PersonInterface pi = new PersonInterface(); //COMPILATION ERROR!!!
```

Another common characteristic is that both interface and class can define methods. However, an interface does not have an implementation code while the class have one.

10.5.4 Creating Interfaces

To create an interface, we write,

```java
public interface [InterfaceName]
{
    //some methods without the body
}
```

As an example, let’s create an interface that defines relationships between two objects according to the “natural order” of the objects.

```java
public interface Relation
{
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Now, to use the interface, we use the `implements` keyword. For example,

```java
/**
 * This class defines a line segment
 */
public class Line implements Relation
{
    private double x1;
    private double x2;
    private double y1;
    private double y2;

    public Line(double x1, double x2, double y1, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
    }

    public double getLength(){
```
double length = Math.sqrt((x2-x1)*(x2-x1) +
    (y2-y1)*(y2-y1));
return length;
}

public boolean isGreater( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen > bLen);
}

public boolean isLess( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen < bLen);
}

public boolean isEqual( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen == bLen);
}

When your class tries to implement an interface, always make sure that you implement all the methods of that interface, or else, you would encounter this error,

Line.java:4: Line is not abstract and does not override abstract method isGreater(java.lang.Object,java.lang.Object) in Relation
public class Line implements Relation

1 error

**Coding Guidelines:**

*Use interfaces to create the same standard method definitions in many different classes. Once a set of standard method definition is created, you can write a single method to manipulate all of the classes that implement the interface.*
10.5.5 Relationship of an Interface to a Class

As we have seen in the previous section, a class can implement an interface as long as it provides the implementation code for all the methods defined in the interface.

Another thing to note about the relationship of interfaces to classes is that, a class can only EXTEND ONE super class, but it can IMPLEMENT MANY interfaces. An example of a class that implements many interfaces is,

```java
public class Person implements PersonInterface,
                 LivingThing,
                 WhateverInterface {
    //some code here
}
```

Another example of a class that extends one super class and implements an interface is,

```java
public class ComputerScienceStudent extends Student
    implements PersonInterface,
               LivingThing {
    //some code here
}
```

Take note that an interface is not part of the class inheritance hierarchy. Unrelated classes can implement the same interface.

10.5.6 Inheritance among Interfaces

Interfaces are not part of the class hierarchy. However, interfaces can have inheritance relationship among themselves. For example, suppose we have two interfaces `StudentInterface` and `PersonInterface`. If `StudentInterface` extends `PersonInterface`, it will inherit all of the method declarations in `PersonInterface`.

```java
public interface PersonInterface {
    
}

public interface StudentInterface extends PersonInterface {
    
}
```
10.6 Exercises

10.6.1 Extending StudentRecord

In this exercise, we want to create a more specialized student record that contains additional information about a Computer Science student. Your task is to extend the StudentRecord class that was implemented in the previous lessons. Add some attributes and methods that you think are needed for a Computer Science student record. Try to override some existing methods in the superclass StudentRecord, if you really need to.

10.6.2 The Shape abstract class

Try to create an abstract class called Shape with abstract methods getArea() and getName(). Write two of its subclasses Circle and Square. You can add additional methods to its subclasses if you want to.
11 Basic Exception Handling

11.1 Objectives

In this section, we are going to study a technique used in Java to handle unusual conditions that interrupt the normal operation of the program. This technique is called exception handling.

At the end of the lesson, the student should be able to:

- Define exceptions
- Handle exceptions using a simple try-catch-finally block

11.2 What are Exceptions?

An exception is an event that interrupts the normal processing flow of a program. This event is usually some error of some sort. This causes our program to terminate abnormally.

Some examples of exceptions that you might have encountered in our previous exercises are: ArrayIndexOutOfBoundsException exceptions, which occurs if we try to access a non-existent array element, or maybe a NumberFormatException, which occurs when we try to pass as a parameter a non-number in the Integer.parseInt method.

11.3 Handling Exceptions

To handle exceptions in Java, we use a try-catch-finally block. What we do in our programs is that we place the statements that can possibly generate an exception inside this block.

The general form of a try-catch-finally block is,

```java
try{
    //write the statements that can generate an exception
    //in this block
} catch( <exceptionType1> <varName1> ){
    //write the action your program will do if an exception
    //of a certain type occurs
} ...
catch( <exceptionType2> <varName2> ){
    //write the action your program will do if an
    //exception of a certain type occurs
} finally{
    //add more cleanup code here
}
```
Exceptions thrown during execution of the try block can be caught and handled in a catch block. The code in the finally block is always executed.

The following are the key aspects about the syntax of the try-catch-finally construct:

- The block notation is mandatory.
- For each try block, there can be one or more catch blocks, but only one finally block.
- The catch blocks and finally blocks must always appear in conjunction with the try block, and in the above order.
- A try block must be followed by at least one catch block OR one finally block, or both.
- Each catch block defines an exception handle. The header of the catch block takes exactly one argument, which is the exception its block is willing to handle. The exception must be of the Throwable class or one of its subclasses.

Let's take for example a code that prints the second argument when we try to run the code using command-line arguments. Suppose, there is no checking inside your code for the number of arguments and we just access the second argument args[1] right away, we'll get the following exception.

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 1
    at ExceptionExample.main(ExceptionExample.java:5)
```
To prevent this from happening, we can place the code inside a try-catch block. The finally block is just optional. For this example, we won’t use the finally block.

```java
public class ExceptionExample {
    public static void main( String[] args ){
        try{
            System.out.println( args[1] );
        }catch( ArrayIndexOutOfBoundsException exp ){
            System.out.println("Exception caught!");
        }
    }
}
```

So when we try to run the program again without arguments, the output would be,

```
Exception caught!
```
11.4 Exercises

11.4.1 Catching Exceptions 1

Given the following code:

```java
public class TestExceptions{
    public static void main( String[] args ){
        for( int i=0; true; i++ ){
            System.out.println("args["+i+"]="+ args[i]);
        }
    }
}
```

Compile and run the TestExceptions program. The output should look like this:

```
javac TestExceptions  one  two  three
args[0]=one
args[1]=two
args[2]=three
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 3
    at TestExceptions.main(l.java:4)
```

Modify the TestExceptions program to handle the exception. The output of the program after catching the exception should look like this:

```
javac TestExceptions  one  two  three
args[0]=one
args[1]=two
args[2]=three
Exception caught:
    java.lang.ArrayIndexOutOfBoundsException: 3
    Quiting...
```

11.4.2 Catching Exceptions 2

Chances are very good that some programs you've written before have encountered exceptions. Since you didn't catch the exceptions, they simply halted the execution of your code. Go back to those programs and implement exception handling.
Appendix A: Java and NetBeans Installation

In this section, we will discuss on how to install Java and NetBeans in your system (Ubuntu Dapper/Windows XP). If you are not provided with the Java 6.0.05 and NetBeans 6.1 installers by your instructor, you can download a copy of the installers from the Sun Microsystems website (http://java.sun.com/) for Java and http://www.NetBeans.org/downloads/ for NetBeans). Before starting with the installation, copy the installers in your hard disk first.

For Ubuntu Gutsy:
Copy all the installers inside the /usr folder.

For Windows:
Just copy the installers in any temporary directory.
Installing Java in Ubuntu Gutsy

**Step 1:** Go to the folder where you have your installers

![Folder with installers](image)

**Step 2:** Before running your installer, make sure it is executable. To do this, right click on the installer, click on Properties. Next, Click on the Permissions tab, and then Click on the Check Box "Allow executing files as program." Close the window.
Step 3: Double click on the file jdk-6u5-linux-i586.bin. A dialog box will display, click on the button 'Run In Terminal'.

After pressing ENTER, you will see the license agreement displayed on the console.

Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA SE DEVELOPMENT KIT (JDK), VERSION 6

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.

1. DEFINITIONS. "Software" means the identified above in binary form, any other machine readable materials (including, but not limited to, libraries, source files, header files, and data files), any updates or error corrections provided by Sun, and any user manuals, programming guides and other documentation provided to you.
Just press enter, until you see the question: Do you agree to the above license terms? [yes or no]. Just type: yes and press ENTER. Just wait for the installer to finish unpacking all its contents and installing Java.
Step 4: Creating symbolic links

In order to run java commands anywhere, we need to create symbolic links for all the commands in JDK inside the /usr/local/bin directory. To do this, Open the terminal and go to the usr folder by typing cd usr/.

```
teng@teng-laptop:~$ cd /usr
```

Next, Move the installed directory to /opt/ folder, type:

```
sudo mv jdk1.6.0_05/ /opt/
```
To make the symbolic links to the commands, type:

```
sudo ln -s /opt/jdk1.6.0_05/bin/* /usr/bin/
```

Installing Java in Windows

Step 1: Using Windows Explorer, go to the folder where your Java installer is located.

Figure 11.2: Folder containing installers
Step 2: Run the installer

To run the installer, just double-click on the installer icon. Press ACCEPT.

![License Agreement](image)

**Figure 11.3: License agreement**

Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA SE DEVELOPMENT KIT (JDK), VERSION 6

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON.
Click on NEXT to continue installation.
Click on NEXT to continue installation.

![Java Setup - Custom](image)

Click on FINISH to complete installation.
Figure 11.6: Finish installation

Java(TM) SE Development Kit 6 Update 5 Successfully Installed

Product Registration is FREE and provides many benefits:
* Notification of new versions, patches, and updates
* Special offers on Sun products, services and training
* Access to early releases and documentation

The JDK product registration form will be presented after you click Finish.

For more information on what data Registration collects and how it is managed and used, see the Product Registration Information Page.

Product Registration Information

Finish
Installing NetBeans in Ubuntu Gutsy

**Step 1:** Go to the folder where you have your installers
Step 2: Before running your installer, make sure it is executable. To do this, right click on the installer, click on Properties. Click on the Permissions tab, and then Click on the execute box. Close the window.

Step 3: Double click on the file netbeans-5_5-beta-linux.bin. Click on Run in Terminal.
A Netbeans 5.5 dialog will then appear. Click on NEXT.
Click on the radio button that says "I accept the terms in the license agreement". And then click on NEXT.
Click on NEXT.

The next dialog just shows information about NetBeans that you will install. Just click again on NEXT.
Now, just wait for NetBeans to finish its installation. Click on FINISH to complete the installation.
Now, you can run NetBeans by double clicking the Netbeans icon on your desktop.
Installing NetBeans in Windows

Step 1: Using Windows Explorer, go to the folder where your NetBeans installer is located.
Step 2: Run the installer

To run the installer, just double-click on the installer icon. After clicking on the netbeans-6.1beta-javase-windows icon, the NetBeans installation wizard will appear. Click on NEXT to enter installation process.

![NetBeans IDE Installer](image)

*Figure 11.8: NetBeans installation*
The agreement page will appear. Choose to ACCEPT and click NEXT to continue.

Figure 11.9: License Agreement

NETBEANS IDE 6.1 - DEVELOPMENT

Please note this code is under development. Rapid change and occasional instabilities can be expected.

Please review the complete list of open-source licenses governing software included in the Product. They can be found in the THIRDPARTYLICENSE.txt file. Please review the list of libraries and licenses provided for use. The license file contains five distinct licenses.

Unless specified below, the use of NetBeans IDE 6.1 and components from the GlassFish runtime are governed by the terms of either the GNU General Public License Version 2 with Classpath Exception or the Common Development and Distribution License. The Product also contains components from OpenJDK, which are governed exclusively by the terms of the GPLv2 with Classpath Exception.

I accept the terms in the license agreement

---

Introduction to Programming I 211
Then you will be given the choice on which directory to place the NetBeans. You can click on BROWSE to choose a different directory. Next is choosing the Standard Edition JDKs from your machine. If you have finished installing Java, the jdk1.6.0_03 should appear from your choices. Click on NEXT to continue.

It will then inform you the location and size of NetBeans which will be installed to your machine. Click on NEXT to finish installation.
Figure 11.11: Installation Summary
You have installed NetBeans on your computer. Click on FINISH to complete installation.

**Figure 11.12: Successful installation**
Appendix B: Getting to know your Programming Environment (Windows XP version)

In this section, we will be discussing on how to write, compile and run Java programs. There are two ways of doing this, the first one is by using a console and a text editor. The second one is by using NetBeans which is an Integrated Development Environment or IDE.

An IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger.

Before going into details, let us first take a look at the first Java program you will be writing.

**My First Java Program**

```java
public class Hello
{
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

Before we try to explain what the program means, let's first try to write this program in a file and try to run it.
Using a Text Editor and Console

For this example, we will be using the text editor "Notepad" (for Windows) to edit the Java source code. You can use other text editors if you want to. You will also need to open the MS-DOS prompt window to compile and execute your Java programs.

Step 1: Start Notepad

To start Notepad in Windows, click on start-> All Programs-> Accessories-> Notepad.
Step 2: Open the Command Prompt window

To open the MSDOS command prompt in Windows, click on start-> All programs-> Accessories-> Command Prompt.
Step 3: Write your the source code of your Java program in Notepad

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```

Step 4: Save your Java Program

We will save our program on a file named "Hello.java", and we will be saving it inside a folder named `MYJAVAPROGRAMS`.

To open the **Save** dialog box, click on the File menu found on the menubar and then click on Save.

After doing the procedure described above, a dialog box will appear as shown in Figure below.
Click on the MY DOCUMENTS button to open the My Documents folder where we will be saving all your Java programs.
Now, we'll create a new folder inside the My Documents folder where we will save your programs. We shall name this folder MYJAVAPROGRAMS. Click on the button encircled in the figure below to create the folder.

![Image of Save As dialog box with folder named MYJAVAPROGRAMS]

*Figure 11.19: Clicking on the encircled button will create a New Folder.*

After the folder is created, you can type in the desired name for this folder. In this case, type in MYJAVAPROGRAMS, and then press ENTER.
Now that we've created the folder where we will save all the files, double click on that folder to open it. You will see a similar figure as shown below. The folder should be empty for now since it's a newly created folder and we haven't saved anything in it yet.
Now click on the drop down list box "Save as type", so that we can choose what kind of file we want to save. Click on the "All Files" option.

Now, in the Filename textbox, type in the filename of your program, which is "Hello.java", and then click on the SAVE button.
Now that you've saved your file, notice how the title of the frame changes from Untitled-Notepad to Hello.java-Notepad. Take note that if you want to make changes in your file, you can just edit it, and then save it again by clicking on File -> Save.

```java
public class Hello {
    /**
     * My first java program
     */
    public static void main(String[] args) {
        //prints the string "Hello world" on screen
        System.out.println("Hello world!");
    }
}
```
Step 5: Compiling your program

Now, the next step is to compile your program. Go to the MSDOS command prompt window we just opened a while ago.

Typically, when you open the command prompt window, it opens up and takes you directly to what is called your home folder. To see what is inside that home folder, type **DIR** or **dir** and then press ENTER. What you will see is a list of files and folders inside your home folder.

![Command Prompt](image)

Figure 11.20: List of files and folders shown after executing the command DIR.

Now, you can see here that there is a folder named "My Documents" where we created your MYJAVAPROGRAMS folder. Now let's go inside that directory.
To go inside a directory, you type in the command: `cd [directory name]`. The "cd" command stands for, change directory. In this case, since the name of our directory is My Documents, you type in: `cd My Documents`.

Now that you are inside the "My Documents" folder, try typing in the "dir" command again, and tell me what you see.

Now perform the same steps described before to go inside the MYJAVAPROGRAMS folder.
Once inside the folder where your Java programs are, let us now start compiling your Java program. Take note that, you should make sure that the file is inside the folder where you are in. In order to do that, execute the `dir` command again to see if your file is inside that folder.

To compile a Java program, we type in the command: `javac [filename]`. So in this case, type in: `javac Hello.java`.

During compilation, `javac` adds a file to the disk called `[filename].class`, or in this case, `Hello.class`, which is the actual bytecode.
Step 6: Running the Program

Now, assuming that there are no problems during compilation (we'll explore more of the problems encountered during compilation in the next section), we are now ready to run your program.

To run your Java program, type in the command: java [filename without the extension], so in the case of our example, type in: java Hello

You can see on the screen that you have just run your first Java program that prints the message, "Hello world!".
Setting the Path

Sometimes, when you try to invoke the `javac` or `java` command, you encounter the message: `{javac} is not recognized as an internal or external command, operable program or batch file`. This means that either you haven't installed Java in your system yet, or you have to configure the path on where the Java commands are installed so that your system will know where to find them.

If you are sure that you’ve already installed Java in your system, try setting the PATH variable to point to where the Java commands are installed. To do this, type in the command:

```
set PATH=C:\j2sdk1.4.2_04\bin
```

This will tell your system to look for the commands in the `C:\j2sdk1.4.2_04\bin` folder, which is usually the default location wherein your Java files are placed during installation. After doing this, you can now use the Java commands.
J.E.D.I

**Using NetBeans**

Now that we've tried doing our programs the complicated way, let's now see how to do all the processes we've described in the previous sections by using just one application.

In this part of the lesson, we will be using **NetBeans**, which is an **Integrated Development Environment or IDE**. An IDE is a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger.

**Step 1: Run NetBeans**

To run NetBeans, click on start-> All Programs-> NetBeans 5.5 Beta -> NetBeans IDE
After you've open NetBeans IDE, you will see a graphical user interface (GUI) similar to what is shown below.

Figure 11.28: NetBeans IDE
Step 2: Make a project

Now, let's first make a project. Click on File -> New Project.

After doing this, a New Project dialog will appear.
Now click on Java Application and click on the NEXT button.
Now, a New Application dialog will appear. Edit the Project Name part and type in "HelloApplication".

*Figure 11.29: Change Project Name*
Now try to change the Application Location, by clicking on the BROWSE button. Follow the steps described in the previous section to go to your MYJAVAPROGRAMS folder.

Finally, on the Create Main Class textfield, type in Hello as the main class' name, and then click on the FINISH button.
Step 3: Type in your program

Before typing in your program, let us first describe the main window after creating the project.

As shown below, NetBeans automatically creates the basic code for your Java program. You can just add your own statements to the generated code. On the left side of the window, you can see a list of folders and files that NetBeans generated after creating the project. This can all be found in your MYJAVAPROGRAMS folder, where you set the Project location.

Now, try to modify the code generated by NetBeans. Ignore the other parts of the program for now, as we will explain the details of the code later. Insert the code:
```
System.out.println("Hello world!");
```
after the statement, //TODO code application logic here here.
Step 4: Compile your program

Now, to compile your program, just click on Build -> Build Main Project. Or, you could also use the shortcut button to compile your code.

Figure 11.30: Shortcut button to compile code
If there are no errors in your program, you will see a build successful message on the output window.

```java
public static
    // TODO c
    System.out

BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 11.31: Output window just below the window where you type your source code
Step 5: Run your program

To run your program, click on Run -> Run Main Project. Or you could also use the shortcut button to run your program.

![Shortcut button to run program](image1)

The output of your program is displayed in the output window.

![Output of Hello.java](image2)
Appendix C : Answers to Exercises

Chapter 1 Exercises

1.1 Writing Algorithms

1. Baking Bread

**Pseudocode:**

```plaintext
prepare all ingredients
pour all ingredients in mixing bowl
while batter not smooth yet
    mix ingredients
pour into bread pan
place inside oven
while bread not yet done
    wait
remove from oven
```

**Flowchart:**

[Diagram of the flowchart for baking bread]

---

Introduction to Programming I 239
2. Logging into your laboratory's computer

**Pseudocode:**

Let power = computer's power button
Let in = status of user (initially false)

if power == off
    Press power button

Enter "boot" process

while in == false
    enter user name
    enter password
    if password and user name correct
        in = true
end while

**Flowchart:**

![Flowchart diagram]
3. Getting the average of three numbers

**Pseudocode:**

Let count = 0
Let sum = 0
Let average = 0

While count < 3
   Get number
   sum = sum + number
   count++

average = sum/3

Display average

**Flowchart:**
1.2 Number Conversions

1. 1980\(_{10}\) to binary, hexadecimal and octal

To Binary:

\[
\begin{align*}
1980/2 &= 990 & \text{0} \\
990/2 &= 495 & \text{0} \\
495/2 &= 247 & \text{1} \\
247/2 &= 123 & \text{1} \\
123/2 &= 61 & \text{1} \\
61/2 &= 30 & \text{1} \\
30/2 &= 15 & \text{0} \\
15/2 &= 7 & \text{1} \\
7/2 &= 3 & \text{1} \\
3/2 &= 1 & \text{1} \\
1/2 &= 0 & \text{1}
\end{align*}
\]

Binary = 11110111100

To Hexadecimal:

\[
\begin{array}{ccc}
0111 & 1011 & 1100 \\
7 & B & C \\
\end{array}
\]

Hexadecimal = 7BC

To Octal:

\[
\begin{array}{cccc}
011 & 110 & 111 & 100 \\
3 & 6 & 7 & 4 \\
\end{array}
\]

Octal = 3674
2. 1001001101₂ to decimal, hexadecimal and octal

To Decimal:

\[
\begin{align*}
1 \times 1 &= 1 \\
0 \times 2 &= 0 \\
1 \times 4 &= 4 \\
1 \times 8 &= 8 \\
0 \times 16 &= 0 \\
0 \times 32 &= 0 \\
1 \times 64 &= 64 \\
0 \times 128 &= 0 \\
0 \times 256 &= 0 \\
1 \times 512 &= 512 \\
\text{TOTAL} &= 589 \\
\end{align*}
\]

Decimal = 589

To Hexadecimal:

\[
\begin{align*}
0010 &\quad 0100 &\quad 1101 \\
2 &\quad 4 &\quad D \\
\end{align*}
\]

Hexadecimal = 24D

To Octal:

\[
\begin{align*}
001 &\quad 001 &\quad 001 &\quad 101 \\
1 &\quad 1 &\quad 1 &\quad 5 \\
\end{align*}
\]

Octal = 1115
3. 76₈ to binary, hexadecimal and decimal

To Binary:

111  110
7    6

Binary = 111110

To Hexadecimal:

0011  1110
3    E

Hexadecimal = 3E

To Decimal:

6 * 1 =  6
7 * 8 = 56
TOTAL = 62

Decimal = 62
4. $43F_{16}$ to binary, decimal and octal

To Binary:

\[
\begin{array}{c}
4 \\
3 \\
F \\
\end{array}
\begin{array}{c}
0100 \\
0011 \\
1111 \\
\end{array}
\]

Binary = 010000111111

To Decimal:

\[
\begin{align*}
F \times 1 &= 15 \\
3 \times 16 &= 48 \\
4 \times 256 &= 1024 \\
\text{TOTAL} &= 1087 \\
\end{align*}
\]

Decimal = 1087

To Octal:

\[
\begin{array}{c}
010 \\
000 \\
111 \\
111 \\
\end{array}
\begin{array}{c}
2 \\
0 \\
7 \\
7 \\
\end{array}
\]

Octal = 02077

**Chapter 2 (No exercises)**
Chapter 3 Exercises

3.1 Hello World!
/**
 * This class prints the line "Welcome to Java Programming
 * [YourName]!!!" on screen.
 */
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to Java Programming 
[YourName]!!!");
    }
}

3.2 The Tree
/**
 * A program that prints four lines on screen
 */
public class TheTree
{
    public static void main(String[] args)
    {
        System.out.println("I think I shall never see,");
        System.out.println("a poem as lovely as a tree.");
        System.out.println("A tree whose hungry mouth is 
pressed");
        System.out.println("Against the Earth's flowing
breast.");
    }
}
Chapter 4 Exercises

4.1 Declaring and printing variables

/// A program that declares different variables
/// then outputs the values of the variables
*/
public class VariableSample {
    public static void main(String[] args) {
        // declares integer number with 10 as initial value
        int number = 10;

        // declares character letter with 'a' as initial value
        char letter = 'a';

        // declares boolean result with true as initial value
        boolean result = true;

        // declares String str with "hello" as initial value
        String str = "hello";

        // prints the values of the variables on screen
        System.out.println("Number = "+number);
        System.out.println("letter = "+letter);
        System.out.println("result = "+result);
        System.out.println("str = "+str);
    }
}

4.2 Getting the average of three numbers

/// A program that solves for the average
/// of the three numbers: 10,20, and 45
/// then outputs the result on the screen
*/
public class AverageNumber {
    public static void main(String[] args) {
        // declares the three numbers
        int num1 = 10;
        int num2 = 20;
        int num3 = 45;

        // get the average of the three numbers
        // and saves it inside the ave variable
        int ave = (num1+num2+num3)/3;

        // prints the output on the screen
        System.out.println("number 1 = "+num1);
        System.out.println("number 2 = "+num2);
        System.out.println("number 3 = "+num3);
        System.out.println("Average is = "+ave);
    }
}
4.3 Output greatest value

```java
/**
 * A program that outputs the number with
 * the greatest value given three numbers
 */
public class GreatestValue {

    public static void main(String[] args){

        // declares the numbers
        int num1 = 10;
        int num2 = 23;
        int num3 = 5;
        int max = 0;

        // determines the highest number
        max = (num1>num2)?num1:num2;
        max = (max>num3)?max:num3;

        // prints the output on the screen
        System.out.println("number 1 = "+num1);
        System.out.println("number 2 = "+num2);
        System.out.println("number 3 = "+num3);
        System.out.println("The highest number is = "+max);
    }
}
```

4.4 Operator precedence

1. (((a/b)^c)^((d-e+f-(g*h))+i))
2. ((((((3*10)*2)/15)-2+4)^2)^2)
3. ((r^((((s*t)/u)-v)+w))^(x-(y++)))
5.1 Last 3 words (BufferedReader version)

```java
import java.io.*;
/**
 * A program that asks three words from the user
 * and then prints it on the screen as a phrase
 */
public class LastThreeWords {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String firstWord = "";
        String secondWord = "";
        String thirdWord = "";
        try {
            System.out.print("Enter word1: ");
            firstWord = reader.readLine(); // gets the 1st word
            System.out.print("Enter word2: ");
            secondWord = reader.readLine(); // gets the 2nd word
            System.out.print("Enter word3: ");
            thirdWord = reader.readLine(); // gets the 3rd word
        } catch (IOException e) {
            System.out.println("Error in getting input");
        }
        // prints the phrase
        System.out.println(firstWord + " " + secondWord + " " + thirdWord);
    }
}
```
5.2 Last 3 words (JOptionPane version)

import javax.swing.JOptionPane;
/**
 * A program that asks three words from the user using the
 * JOptionPane
 * and then displays these three words as a phrase on the
 * screen
 */
public class LastThreeWords {

    public static void main(String[] args) {

        // gets the first word from the user
        String firstWord = JOptionPane.showInputDialog
                             ("Enter word1");

        // gets the second word from the user
        String secondWord = JOptionPane.showInputDialog
                            ("Enter word2");

        // gets the third word from the user
        String thirdWord = JOptionPane.showInputDialog
                           ("Enter word3");

        // displays the message
        JOptionPane.showMessageDialog(null, firstWord +
                                       " "+secondWord +
                                       " "+thirdWord);
    }
}
Chapter 6 Exercises

6.1 Grades

Using BufferedReader:

```java
import java.io.*;
/**
 * Gets three number inputs from the user 
 * then displays the average on the screen 
 */
public class Grades {
    public static void main(String[] args){
        //declares the variable reader as the
        BufferedReader reader = new BufferedReader
            ( new InputStreamReader( System.in));
        int firstGrade = 0;
        int secondGrade = 0;
        int thirdGrade = 0;
        double average = 0;
        try{
            System.out.print("First grade: ");
            firstGrade = Integer.parseInt
                (reader.readLine());
            System.out.print("Second grade: ");
            secondGrade = Integer.parseInt
                (reader.readLine());
            System.out.print("Third grade: ");
            thirdGrade = Integer.parseInt
                (reader.readLine());
        }catch( Exception e){
            System.out.println("Input is invalid");
            System.exit(0);
        }
        //solves for the average
        average = (firstGrade+secondGrade+thirdGrade)/3;
        //prints the average of the three exams
        System.out.print("Average: "+average);
        if(average>=60) System.out.print(" ;-)");
        else System.out.print(" ;-(");
    }
}
```
Using JOptionPane:

```java
import javax.swing.JOptionPane;
/**
* Gets three number inputs from the user
* then displays the average on the screen
*/
public class Grades {
    public static void main(String[] args) {
        double firstGrade = 0;
        double secondGrade = 0;
        double thirdGrade = 0;
        double average = 0;
        try {
            firstGrade = Double.parseDouble(JOptionPane.showInputDialog("First grade"));
            secondGrade = Double.parseDouble(JOptionPane.showInputDialog("Second grade"));
            thirdGrade = Double.parseDouble(JOptionPane.showInputDialog("Third grade"));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Input is invalid");
            System.exit(0);
        }
        //solves for the average
        average = (firstGrade + secondGrade + thirdGrade) / 3;
        if (average >= 60) {
            JOptionPane.showMessageDialog(null, "Average: "+average+" ;-)");
        } else {
            JOptionPane.showMessageDialog(null, "Average: "+average+" ;-()");
        }
    }
}
```
6.2 Number in words

Using if-else statement:

```java
import javax.swing.JOptionPane;

/**
 * Transforms a number input from 1-10 to words
 * using if-else
 */
public class NumWords
{

    public static void main(String[] args)
    {
        String msg = "";
        int input = 0;

        //gets the input string
        input = Integer.parseInt(JOptionPane.showInputDialog
            ("Enter number"));

        //sets msg to the string equivalent of input
        if(input == 1) msg = "one";
        else if(input == 2) msg = "two";
        else if(input == 3) msg = "three";
        else if(input == 4) msg = "four";
        else if(input == 5) msg = "five";
        else if(input == 6) msg = "six";
        else if(input == 7) msg = "seven";
        else if(input == 8) msg = "eight";
        else if(input == 9) msg = "nine";
        else if(input == 10) msg = "ten";
        else msg = "Invalid number";

        //displays the number in words if with in range
        JOptionPane.showMessageDialog(null, msg);
    }
}
```
Using switch statement:

```java
import javax.swing.JOptionPane;
/**
 * Transforms a number input from 1-10 to words
 * using switch.
 */
public class NumWords {
    public static void main(String[] args){
        String msg = "";
        int input = 0;

        //gets the input string
        input = Integer.parseInt(JOptionPane.showInputDialog
                ("Enter number"));

        //sets msg to the string equivalent of input
        switch(input){
            case 1:
                msg = "one";
                break;
            case 2:
                msg = "two";
                break;
            case 3:
                msg = "three";
                break;
            case 4:
                msg = "four";
                break;
            case 5:
                msg = "five";
                break;
            case 6:
                msg = "six";
                break;
            case 7:
                msg = "seven";
                break;
            case 8:
                msg = "eight";
                break;
            case 9:
                msg = "nine";
                break;
            case 10:
                msg = "ten";
                break;
            default:
                msg = "Invalid number";
                break;
        }
        //displays the number in words if with in range
        JOptionPane.showMessageDialog(null,msg);
    }
}
```
6.3 Hundred Times

Using while-loop:

```java
import java.io.*;

/**
 * A program that prints a given name one hundred times
 * using while loop
 */

public class HundredNames{

    public static void main(String[] args){
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String name = "";
        int counter = 0;

        //gets the users' name
        try{
            System.out.print("Enter name: ");
            name = reader.readLine();
        }catch(Exception e){
            System.out.println("Invalid input");
            System.exit(0);
        }

        //while loop that prints the name one hundred times
        while(counter < 100){
            System.out.println(name);
            counter++;
        }
    }
}
```
Using do-while loop:

```java
import java.io.*;
/**
 * A program that prints a given name one hundred times
 * using do-while loop
 */
public class HundredNames {

    public static void main(String[] args){
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String name = "";
        int counter = 0;

        //gets the users' name
        try{
            System.out.print("Enter name: ");
            name = reader.readLine();
        }catch(Exception e){
            System.out.println("Invalid input");
            System.exit(0);
        }

        //do-while loop that prints the name one hundred times
        do{
            System.out.println(name);
            counter++;
        }while(counter < 100);
    }
}
```
Using for loop:

import java.io.*;
/**
 * A program that prints a given name one hundred times
 * using do-while loop
 */
public class HundredNames
{
    public static void main(String[] args)
    {
        BufferedReader reader = new BufferedReader
            (new InputStreamReader(System.in));
        String name = "";
        //gets the users' name
        try{
            System.out.print("Enter name: ");
            name = reader.readLine();
        }
        catch(Exception e){
            System.out.println("Invalid input");
            System.exit(0);
        }
        //for loop that prints the name one hundred times
        for(int counter = 0; counter < 100; counter++)
        {
            System.out.println(name);
        }
    }
}
6.4 Powers

Using while-loop:

```java
import javax.swing.JOptionPane;

/**
 * Computes the power of a number given the base and the
 * exponent. The exponent is limited to positive numbers only.
 */
public class Powers {

    public static void main(String[] args) {
        int base = 0;
        int exp = 0;
        int power = 1;
        int counter = 0;

        // gets the user input for base and power using
        // JOptionPane
        base = Integer.parseInt(JOptionPane.showInputDialog("Base"));
        exp = Integer.parseInt(JOptionPane.showInputDialog("Exponent"));

        // limits the exp to positive numbers only
        if (exp < 0) {
            JOptionPane.showMessageDialog(null, "Positive numbers only please");
            System.exit(0);
        }

        // while loop that solves for the power
        while (counter < exp) {
            power = power * base;
            counter++;
        }

        // displays the result
        JOptionPane.showMessageDialog(null, base + " to the " + exp + " is " + power);
    }
}
```
J.E.D.I

Using do-while loop:
import javax.swing.JOptionPane;
/**
* Computes the power of a number given the base and the
* exponent. The exponent is limited to positive numbers only.
*/
public class Powers
{
public static void main(String[] args){
int
int
int
int

base = 0;
exp = 0;
power = 1;
counter = 0;

//gets the user input for base and power
//using JOptionPane
base = Integer.parseInt(JOptionPane.showInputDialog
("Base"));
exp = Integer.parseInt(JOptionPane.showInputDialog
("Exponent"));
//limits the exp to positive numbers only
if(exp < 0 ){
JoptionPane.showMessageDialog
(null,"Positive numbers only please");
System.exit(0);
}
//do-while loop that solves the power given the base
// and exponent
do{
if(exp != 0)
power = power*base;
counter++;
}while(counter < exp);

}

//displays the result
JoptionPane.showMessageDialog(null,base +
" to the "+exp
+ " is "+power);
}

Introduction to Programming I

259


Using for loop:

```java
import javax.swing.JOptionPane;

/**
* Computes the power of a number given the base and the
* exponent. The exponent is limited to positive numbers only.
*/
public class Powers {

    public static void main(String[] args){

        int base = 0;
        int exp = 0;
        int power = 1;
        int counter = 0;

        //gets the user input for base and power using
        // JOptionPane
        base = Integer.parseInt(JOptionPane.showInputDialog
        ("Base");
        exp = Integer.parseInt(JOptionPane.showInputDialog
        ("Exponent"));

        //limits the exp to positive numbers only
        if(exp < 0 ){
            JOptionPane.showMessageDialog(null,"Positive
            numbers only please");
            System.exit(0);
        }

        //for loop for computing the power
        for(counter = 0; counter < exp; counter++){
            power = power*base;
        }

        //displays the result
        JOptionPane.showMessageDialog(null,base +
                        " to the "+exp + " is "+power);
    }
}
```
Chapter 7 Exercises

7.1 Days of the Week

Using while loop:

```java
/**
 * Uses an array string to save the days of the week
 * then prints it on the screen.
 */
public class DaysOfTheWeek
{
    public static void main(String[] args)
    {
        //declares the String array of the days of the week
        String[] days = {"Sunday","Monday","Tuesday",
            "Wednesday","Thursday","Friday",
            "Saturday"};
        int counter = 0;
        //while loop that prints the days of the week
        while(counter < days.length){
            System.out.println(days[counter]);
            counter++;
        }
    }
}
```

Using do-while loop:

```java
/**
 * Uses an array string to save the days of the week
 * then prints it on the screen with a do-while loop.
 */
public class DaysOfTheWeek
{
    public static void main(String[] args)
    {
        //declares the String array of the days of
        // the week
        String[] days = {"Sunday","Monday","Tuesday",
            "Wednesday","Thursday","Friday",
            "Saturday"};
        int counter = 0;
        //do-while loop that prints the days of the
        // week
        do{
            System.out.println(days[counter]);
            counter++;
        }while(counter < days.length);
    }
}
```
Using for loop:

```java
/**
 * Uses an array string to save the days of the wee
 * then prints it on the screen with a for loop.
 */
public class DaysOfTheWeek {

    public static void main(String[] args) {
        // declares the String array of the days of
        // the week
        String[] days = {"Sunday", "Monday", "Tuesday",
            "Wednesday", "Thursday", "Friday",
            "Saturday"};

        // for loop that prints the days of the week
        for (int counter = 0; counter < days.length;
            counter++)
            System.out.println(days[counter]);
    }
}
```

### 7.2 Greatest number

```java
import javax.swing.JOptionPane;

/**
 * A program that uses JOptionPane to get ten numbers
 * from the user then outputs the largest number.
 */
public class GreatestNumber {

    public static void main(String[] args) {
        int[] num = new int[10];
        int counter;
        int max = 0;

        // for loop that gets the 10 numbers from the user
        for (counter = 0; counter < 10; counter++) {
            num[counter] = Integer.parseInt
                (JOptionPane.showInputDialog(  
                "Enter number "+(counter+1)));

            // gets the maximum number
            if ((counter == 0) || (num[counter] > max))
                max = num[counter];
        }

        // displays the number with the greatest number
        JOptionPane.showMessageDialog(null,
            "The number with the greatest value is "+max);
    }
}
```
Chapter 8 Exercises

8.1 Print Arguments

```java
/**
 * A program that prints the string from the command line if any.
 */
public class CommandLineSample
{
    public static void main(String[] args){
        //checks if a command line argument exists
        if(args.length == 0)
            System.exit(0);

        //for loop that prints the arguments from the command line
        for(int counter=0;counter<args.length; counter++){
            System.out.println(args[counter]);
        }
    }
}
```
Chapter 9 Exercises

9.1 Defining terms
See definitions in book.

9.2 Java Scavenger Hunt
To the teacher: These are just some sample methods in the Java API that you can use. Check the Java API for more answers.

Sample Usage:
```java
public class Homework1 {
    public static void main(String []args){
        //1. endsWith
        String str = "Hello";
        System.out.println( str.endsWith( "slo" ) );

        //2. forDigit
        System.out.println( Character.forDigit(13, 16) );

        //4. floor
        System.out.println( Math.floor(3.14) );

        //5. isDigit
        System.out.println( "0" + Character.isDigit('0') );
        System.out.println( "A" + Character.isDigit('A') );

        //3.
        System.exit(1);
        System.out.println("if this is executed, exit was not called");
    }
}
```

Class and Method declaration:
1. Class: String
   Method: public boolean endsWith( String suffix )

2. Class: Character
   Method: public static char forDigit( int digit, int radix )

3. Class: System
   Method: public static void exit( int status )

4. Class: Math
   Method: public static double floor( double a )

5. Class: Character
   Method: public static boolean isDigit( char ch )
Chapter 10 Exercises

10.1 Address Book Entry

/**
 * An address book class that record a persons
 * name, address, telephone number, and email address
 */
public class AddressBookEntry
{
    private String name;
    private String add;
    private int tel;
    private String email;

    /**
     * default constructor
     */
    public AddressBookEntry(){
        name = "";
        add = "";
        tel = 0;
        email = "";
    }

    /**
     * Creates an AddressBookEntry object with the given
     * name, address, telephone number and email adress
     */
    public AddressBookEntry(String name, String add,
                            int tel,  String email){
        this.name = name;
        this.add = add;
        this.tel = tel;
        this.email = email;
    }

    /**
     * returns the variable name
     */
    public String getName(){
        return name;
    }

    /**
     * changes the variable name
     */
    public void changeName(String name){
        this.name = name;
    }

    /**
     * returns the variable add
     */
    public String getAddress(){
        return add;
    }
}
public void changeAddress(String add) {
    this.add = add;
}

public int getTelNumber() {
    return tel;
}

public void changeTelNumber(int tel) {
    this.tel = tel;
}

public String getEmailAdd() {
    return email;
}

public void changeEmailAdd(String email) {
    this.email = email;
}
10.2 AddressBook

import java.io.*;
/**
 * Creates an addressbook that contains 100 AddressBookEntries
 */
public class AddressBook
{
    //index of the last entry
    private int top = 0;
    //constant number that indicates the maximum
    //number of entries in the address book
    private static final int MAXENTRIES = 100;
    //array of Address Book Entries
    private AddressBookEntry[] list;
    /**
     * The main method
     */
    public static void main(String[] args)
    {
        BufferedReader keyIn = new BufferedReader
            (new InputStreamReader(System.in));
        AddressBook addBook = new AddressBook();
        String act = "";
        while(true)
        {
            //displays the options
            System.out.println("\n[A] Add entry");
            System.out.println("[D] Delete entry");
            System.out.println("[V] View all entries");
            System.out.println("[U] Update entry");
            System.out.println("[Q] Quit");
            System.out.print("Enter desired action: ");
            try{
                //gets the choice
                act = keyIn.readLine();
            }catch(Exception e){
                System.out.println("Error");
            }
        }
    }
}
// checks for the appropriate action for
// his choice
if(act.equals("A")||act.equals("a"))
    addBook.addEntry();
else if(act.equals("D")||act.equals("d"))
    addBook.delEntry();
else if(act.equals("V")||act.equals("v"))
    addBook.viewEntries();
else if(act.equals("U")||act.equals("u"))
    addBook.updateEntry();
else if(act.equals("Q")||act.equals("q"))
    System.exit(0);
else
    System.out.println
    ("Unknown command");
}
/**
 * creates the AddressBook
 */
public AddressBook(){
    list = new AddressBookEntry[MAXENTRIES];
}

/**
 * method for adding an AddressBookEntry to the AddressBook
 */
public void addEntry(){
    BufferedReader keyIn = new BufferedReader
    (new InputStreamReader(System.in));
    String name = "";
    String add = "";
    int tel = 0;
    String email = "";
    if(top == MAXENTRIES){
        System.out.println("Address Book is full");
        return;
    }
    try{
        System.out.print("Name: ");
        name = keyIn.readLine();
        System.out.print("Address: ");
        add = keyIn.readLine();
        System.out.print("Telephone number: ");
        tel = Integer.parseInt(keyIn.readLine());
        System.out.print("Email Address: ");
        email = keyIn.readLine();
    }catch(Exception e){
        System.out.println(e);
        System.exit(0);
    }
    AddressBookEntry entry = new AddressBookEntry
    (name, add, tel, email);
    list[top] = entry;
    top++;
}
/**
 * method that deletes an AddressBookEntry from the
 * Adressbook with the index
 */
public void delEntry(){

BufferedReader keyIn = new BufferedReader
    (new InputStreamReader(System.in));
int index = 0;

//checks if the address book is empty
if(top == 0){
    System.out.println("Address Book is empty");
    return;
}

//asks for the entry which is to be deleted
try{
    //shows the current entries on the record book
    viewEntries();
    System.out.print("\nEnter entry number: ");
    index = Integer.parseInt(keyIn.readLine())-1;
}catch(Exception e){}

//checks if the index is with in bounds
if(index < 0 || index >= top){
    System.out.println("Index Out Of Bounds");
    return;
}else{
    for( int i=index; i<top; i++ ){
        list[i] = list[i+1];
    }
    list[top] = null;
    top--;
}
}

/**
 * method that prints all the entries in the AddressBook
 */
public void viewEntries(){

for(int index = 0; index < top; index++){
    System.out.println((index+1)+" Name:"+
        list[index].getName());
    System.out.println("Address:"+
        list[index].getAddress());
    System.out.println("Telephone Number:"+
        list[index].getTelNumber());
    System.out.println("Email Address:"+
        list[index].getEmailAdd());
}
}
public void updateEntry()
{
    BufferedReader keyIn = new BufferedReader(new InputStreamReader(System.in));
    int index = 0;
    String name = "";
    String add = "";
    int tel = 0;
    String email = "";

    //asks for the entries data
    try{
        System.out.print("Entry number:       ");
        index = Integer.parseInt(keyIn.readLine())-1;
        System.out.print("Name:              ");
        name = keyIn.readLine();
        System.out.print("Address:          ");
        add = keyIn.readLine();
        System.out.print("Telephone number: ");
        tel = Integer.parseInt(keyIn.readLine());
        System.out.print("Email Address:     ");
        email = keyIn.readLine();
    }catch(Exception e){
        System.out.println(e);
        System.exit(0);
    }

    //updates the entry
    AddressBookEntry entry = new AddressBookEntry(name, add, tel, email);
    list[index] = entry;
}
Chapter 11 Exercises

11.1 Extending StudentRecord

/**
 * An object that holds the data for a student
 */
public class StudentRecord {
    protected String name;
    protected String address;
    protected int age;
    protected double mathGrade;
    protected double englishGrade;
    protected double scienceGrade;
    protected double average;

    protected static int studentCount;

    /**
     * Returns the name of the student
     */
    public String getName() {
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName(String temp) {
        name = temp;
    }

    /**
     * Returns the address of the student
     */
    public String getAddress() {
        return address;
    }

    /**
     * Changes the address of the student
     */
    public void setAddress(String temp) {
        address = temp;
    }

    /**
     * Returns the age of the student
     */
    public int getAge() {
        return age;
    }

    /**
     * Changes the age of the student
     */
    public void setAge(int temp) {
        age = temp;
    }
}
/**
 * Returns the englishGrade of the student
 */
public double getEnglishGrade(){
    return englishGrade;
}

/**
 * Changes the englishGrade of the student
 */
public void setEnglishGrade(double temp){
    englishGrade = temp;
}

/**
 * Returns the mathGrade of the student
 */
public double getMathGrade(){
    return mathGrade;
}

/**
 * Changes the mathGrade of the student
 */
public void setMathGrade(double temp){
    mathGrade = temp;
}

/**
 * Returns the scienceGrade of the student
 */
public double getScienceGrade(){
    return scienceGrade;
}

/**
 * Changes the scienceGrade of the student
 */
public void setScienceGrade(double temp){
    scienceGrade = temp;
}

/**
 * Computes the average of the english, math and
 * science grades
 */
public double getAverage(){
    return (mathGrade+englishGrade+scienceGrade)/3;
}

/**
 * Returns the number of instances of the
 * StudentRecords
 */
public static int getStudentCount(){
    return studentCount;
}
/**
 * A student record for a Computer Science student
 */
public class ComputerScienceStudentRecord extends StudentRecord
{
    private String     studentNumber;
    private double     comSciGrade;

    /**
     * Returns the studentNumber of the student
     */
    public String getStudentNumber(){
        return studentNumber;
    }

    /**
     * Changes the studentNumber of the student
     */
    public void setStudentNumber(String temp){
        studentNumber = temp;
    }

    /**
     * Returns the comSciGrade of the student
     */
    public double getComSciGrade(){
        return comSciGrade;
    }

    /**
     * Changes the comSciGrade of the student
     */
    public void setComSciGrade(double temp){
        comSciGrade = temp;
    }
}
11.2 Abstract Classes

```java
/**
 * Definition of shape abstract class
 */
public abstract class Shape {
    /**
     * returns the area of a certain shape
     */
    public abstract double getArea();

    /**
     * returns the name of the shape
     */
    public abstract String getName();
}

/**
 * Class definition for object circle
 */
public class Circle extends Shape {
    private static final double pi = 3.1416;
    private double radius = 0;

    /**
     * Constructor
     */
    public Circle(double r) {
        setRadius(r);
    }

    /**
     * returns area
     */
    public double getArea() {
        return pi * radius * radius;
    }

    /**
     * returns shape name
     */
    public String getName() {
        return "circle";
    }

    /**
     * set radius
     */
    public void setRadius(double r) {
        radius = r;
    }

    /**
     * returns radius
     */
    public double getRadius() {
        return radius;
    }
}
```
/**
 * Class definition for object square
 */
public class Square extends Shape
{
    private double side = 0;

    /**
     * Constructor
     */
    public Square(double s)
    {
        setSide( s );
    }

    /**
     * returns area
     */
    public double getArea()
    {
        return side*side;
    }

    /**
     * returns shape name
     */
    public String getName()
    {
        return "square";
    }

    /**
     * set length of side
     */
    public void setSide(double s)
    {
        side = s;
    }

    /**
     * returns length of one side
     */
    public double getSide()
    {
        return side;
    }
}
Chapter 12 Exercises

12.1 Catching Exceptions 1

public class TestExceptions{
    public static void main( String[] args ){
        try{
            for( int i=0; true; i++ ){
                System.out.println("args["+i+"]="+args[i]);
            }
        }catch( ArrayIndexOutOfBoundsException e ){)
            System.out.println("Exception caught:");
            System.out.println(" +e");
            System.out.println("Quiting...");
        }
    }
}

12.2 Catching Exceptions 2

Here are three sample programs that we did before, wherein we included some exception handling.

    import javax.swing.JOptionPane;
    /**
     * A program that uses JOptionPane to get ten numbers
     * from the user then outputs the largest number.
     */
    public class GreatestNumber{
        public static void main(String[] args){
            int[] num = new int[10];
            int counter;
            int max = 0;

            //for loop that gets the 10 numbers from the user
            for(counter = 0; counter < 10; counter++){
                try{
                    num[counter] = Integer.parseInt
                        (JOptionPane.showInputDialog
                            ("Enter number "+(counter+1)));
                }catch(NumberFormatException e){
                    JOptionPane.showMessageDialog
                        (null,"Error "+e);
                }

                //gets the maximum number
                if((counter == 0)||(num[counter] > max))
                    max = num[counter];
            }

            //displays the number with the greatest number
            JOptionPane.showMessageDialog
                (null,"The number with the greatest value is "+max);
        }
    }
import javax.swing.JOptionPane;

/**
 * Transforms a number input from 1-10 to words using switch.
 */
public class NumWords {
    public static void main(String[] args) {
        String msg = "";
        int input = 0;

        try {
            // gets the input string
            input = Integer.parseInt(JOptionPane.showInputDialog("Enter number"));
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Invalid input");
            System.exit(0);
        }

        // sets msg to the string equivalent of input
        switch (input) {
            case 1:
                msg = "one";
                break;
            case 2:
                msg = "two";
                break;
            case 3:
                msg = "three";
                break;
            case 4:
                msg = "four";
                break;
            case 5:
                msg = "five";
                break;
            case 6:
                msg = "six";
                break;
            case 7:
                msg = "seven";
                break;
            case 8:
                msg = "eight";
                break;
            case 9:
                msg = "nine";
                break;
            case 10:
                msg = "ten";
                break;
            default:
                msg = "Invalid number";
                break;
        }

        // displays the number in words if within range
        JOptionPane.showMessageDialog(null, msg);
    }
}
```java
import javax.swing.JOptionPane;

/**
 * Computes the power of a number given the base and the exponent.
 * The exponent is limited to positive numbers only.
 */
public class Powers {
    public static void main(String[] args) {
        int base = 0;
        int exp = 0;
        int power = 1;
        int counter = 0;

        // gets the user input for base and power using JOptionPane
        try {
            base = Integer.parseInt(JOptionPane.showInputDialog("Base"));
            exp = Integer.parseInt(JOptionPane.showInputDialog("Exponent"));
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "Input Error");
            System.exit(0);
        }

        // limits the exp to positive numbers only
        if (exp < 0) {
            JOptionPane.showMessageDialog(null, "Positive numbers only please");
            System.exit(0);
        }

        // for loop for computing the power
        for (; counter < exp; counter++) {
            power = power * base;
        }

        // displays the result
        JOptionPane.showMessageDialog(null, base + " to the " + exp + " is " + power);
    }
}
```
Appendix D : Machine Problems

Machine Problem 1: Phone Book

Write a program that will create an phonebook, wherein you can add entries in the phonebook, delete entries, view all entries and search for entries. In viewing all entries, the user should have a choice, whether to view the entries in alphabetical order or in increasing order of telephone numbers. In searching for entries, the user should also have an option to search entries by name or by telephone numbers. In searching by name, the user should also have an option if he/she wants to search by first name or last name.

MAIN MENU
1 - Add phonebook entry
2 - Delete phonebook entry
3 - View all entries
   a - alphabetical order
   b - increasing order of telephone numbers
4 - Search entries
   a - by name
   b - by telephone number
5 - Quit

The following will appear when one of the choices in the main menu is chosen.

Add phonebook entry
Enter Name:
Enter Telephone number:
(* if entry already exists, warn user about this)

View all entries
Displays all entries in alphabetical order
Displays all entries in increasing order of telephone #s

Search entries
Search phonebook entry by name
Search phonebook entry by telephone number

Quit
close phonebook
**Machine Problem 2: Minesweeper**

This is a one player game of a simplified version of the popular computer game minesweeper. First, the user is asked if he or she wants to play on a 5x5 grid or 10x10 grid. You have two 2-dimensional arrays that contains information about your grid. An entry in the array should either contain a 0 or 1. A 1 signifies that there is a bomb in that location, and a 0 if none.

For example, given the array:

```c
int bombList5by5[][]={{0, 0, 1, 0, 0},
                      {0, 0, 0, 0, 0},
                      {0, 0, 0, 1, 1},
                      {0, 1, 1, 0, 0}};
```

Given the bomb list, we have 6 bombs on our list. The bombs are located in (row,col) cells, (0,2), (2,1), (3,3), (3,4), (4,1) and (4,2).

If the user chooses a cell that contains a bomb, the game ends and all the bombs are displayed. If the user chooses a cell that does not contain a bomb, a number appears at that location indicating the number of neighbors that contain bombs. The game should end when all the cells that do not contain bombs have been marked (player wins) or when the user steps on a bomb (player loses).

Here's a sample output of the game, given the bombList5by5.

```
Welcome to Minesweeper!
Choose size of grid (Press 1 for 5x5, Press 2 for 10x10): 1
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
Enter row and column of the cell you want to open[row  col]: 1  1
[ ] [ ] [ ] [ ] [ ]
[ ] [2 ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
Enter row and column of the cell you want to open[row  col]: 3  2
[ ] [ ] [ ] [ ] [ ]
[ ] [2 ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [4 ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
Enter row and column of the cell you want to open[row  col]: 0 2
[ ] [ ] [ ] [ ] [ ]
[ ] [2 ] [ ] [ ] [ ]
[ ] [X ] [ ] [ ] [ ]
[ ] [ ] [4 ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
Ooppss! You stepped on a bomb. Sorry, game over!
```
Machine Problem 3: Number Conversion

Create your own scientific calculator that will convert the inputted numbers to the four number representations (Decimal, Binary, Octal, Hexadecimal). Your program should output the following menu on screen.

MAIN MENU:
Please type the number of your choice:
1 – Binary to Decimal
2 – Decimal to Octal
3 – Octal to Hexadecimal
4 – Hexadecimal to Binary
5 – Quit

The following will appear when one of the choices in the main menu is chosen.

Choice 1:
Enter a binary number: 11000
11000 base 2 = 24 base 10
(goes back to main menu)

Choice 2:
Enter a Decimal number: 24
24 base 10 = 30 base 8
(goes back to main menu)

Choice 3:
Enter an Octal number: 30
30 base 8 = 18 base 16
(goes back to main menu)

Choice 4:
Enter a Hexadecimal number: 18
18 base 16 = 11000 base 2

Choice 1:
Enter a binary number: 110A
Invalid binary number!
Enter a binary number: 1
1 base 2 = 1 base 10
(goes back to main menu)

(user chooses 5)
Goodbye!

You can be more creative with your user interface if you want to, as long as the program outputs the correct conversion of numbers.
Appendix E : Hands-on Laboratory

Note to the Teacher
This part of the manual is not included in the student's manual. You can give a copy of this to your students if you wish for them to do the exercises on their own. Some of the answers for "Creating your own" exercises are found in the last part of this section.

Chapter 1 Hands-on
None

Chapter 2 Hands-on
None
Chapter 3 Hands-on

3.1 Things to check before you start the lab

Once you installed J2SE SDK, please make sure you do the following:

1. Make sure the installation has set %JAVA_HOME% (Windows) or $JAVA_HOME (Solaris/Linux) environment variable to the installation directory of J2SE 1.4.2_06 (or later version)

2. Type "echo %JAVA_HOME%" (Windows) or "echo $JAVA_HOME" (Solaris/Linux) in a terminal window. You should see the following:
   - c:\j2sdk1.4.2_06 (Windows)
   - /usr/jdk/jdk1.4.2_06 (Solaris/Linux)

3. Make sure the installation has placed %JAVA_HOME%\bin (Windows) or $JAVA_HOME/bin (Solaris/Linux) in the "path" environment variable. Type "java -version" in a terminal window. You should see something like following:
   - java version "1.4.2_06"
   - Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_06-b03)
   - Java HotSpot(TM) Client VM (build 1.4.2_06-b03, mixed mode)
3.2 Write, Compile, and Run Hello Java Program

1. `mkdir c:\lab`

2. `cd \lab`

3. Create **Hello.java** using your editor of choice

```java
public class Hello {
    /**
     * My first Java program
     */
    public static void main( String[] args ){
        //prints the string "Hello world" on screen
        System.out.println("Hello world");
    }
}
```

4. Compile **Hello.java**

    `javac Hello.java`

5. Make sure **Hello.class** file has been created

    `dir`

6. Run the **Hello** program

    `java Hello`

7. Verify that the result is as following

    `C:\lab>java Hello
    Hello world`

8. Modify, compile, and run the **Hello.java** so that it prints the following

    "This is my first Java program" (instead of "Hello world")
3.3 Write, Compile, and Run Hello Java Program using NetBeans

1. Start the NetBeans IDE 4.1
   - Windows: Start > All Programs > NetBeans 4.1 > NetBeans IDE or click NetBeans IDE 4.1 desktop icon
   - Solaris/Linux: <NETBEANS41_HOME>/bin/netbeans

2. Create a new NetBeans project and Hello main class
   - Select File from the menu bar and select New Project.
   - Under Choose Project, select General and Java Application
   - Click Next.
   - Under Name and Location pane, (Figure-10 below)
     - For Project Name field, fill it with Hello
     - For Create Main Class field, change it to hello.Hello (from hello.Main)
   - Click Finish

   ![Create new Java application](image)

   - Note that the IDE generated Hello.java code gets displayed in the source editor.

3. Modify the Hello class
   - Replace the code of Hello class of IDE generated Hello.java code in the source editor with the one in Chapter 2 while leaving the package statement on the top.

4. Run Hello class
   - Right click Hello.java node under Hello->Source Packages->hello and select Run File (Shift+F6)
   - Note that the Output window displays the result


**Chapter 4 Hands-on**

**4.1 Declaring, Initializing, Printing Variables**

1. Create **OutputVariable.java** using your editor of choice

   ```java
   public class OutputVariable {
       public static void main( String[] args ){
           int value = 10;
           char x;
           x = 'A';
           System.out.println( value );
           System.out.println( "The value of x=" + x );
       }
   }
   ```

2. Compile and run the code

   ```
   javac OutputVariable.java
   java OutputVariable
   ```

3. Verify that the result is as following

   ```
   C:\lab>java OutputVariable
   10
   The value of x=A
   ```

4. Modify OutputVariable.java as following and compile and run the code

   - Define another primitive type as following
     ```java
     double grade = 11;
     ```
   
     - Print out the value of grade variable as following
     ```
     System.out.println( "The value of grade =" + grade );
     ```
4.2 Conditional Operator

1. Create `ConditionalOperator.java` using your editor of choice
   ```java
   public class ConditionalOperator {
       public static void main( String[] args ){
           String status = "";
           int grade = 80;

           //get status of the student
           status = (grade >= 60)?"Passed":"Fail";

           //print status
           System.out.println( status );
       }
   }
   ```

2. Compile and run the code
   ```
javac ConditionalOperator.java
java ConditionalOperator
   ```

3. Verify that the result is as following
   ```
C:\lab>java ConditionalOperator
Passed
   ```

4. Modify `ConditionalOperator.java` as following, compile and run the code
   ```java
   int salary = 100000;
   Print "rich" if the salary is over 50000. Print "poor" otherwise.
   ```
Chapter 5 Hands-on

5.1 Getting Input From Keyboard via BufferedReader

1. Create `GetInputFromKeyboard.java` using your editor of choice

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class GetInputFromKeyboard {

    public static void main(String[] args) {
        BufferedReader dataIn = new BufferedReader(new InputStreamReader(System.in));
        String name = "";
        System.out.print("Please Enter Your Name:");
        try {
            name = dataIn.readLine();
        } catch (IOException e) {
            System.out.println("Error!");
        }
        System.out.println("Hello " + name + "!");
    }
}
```

2. Compile and run the code

```
javac GetInputFromKeyboard.java
java GetInputFromKeyboard
```

3. Modify `GetInputFromKeyboard.java` as following, compile and run the code

- Make the program to ask the following question
  
  Please enter your age

- Display the entered age as following
  - If the age is over 100, display
    Hello <name>
    You are old!
  - Otherwise
    Hello <name>
    You are young!
5.2 Getting Input From Keyboard via JOptionPane

1. Create `GetInputFromKeyboardJOptionPane.java` using your editor of choice

   ```java
   import javax.swing.JOptionPane;
   public class GetInputFromKeyboardJOptionPane {
       public static void main( String[] args ){
           String name = "";
           name=JOptionPane.showInputDialog("Please enter your name");
           String msg = "Hello " + name + "!";
           JOptionPane.showMessageDialog(null, msg);
       }
   }
   ```

2. Compile and run the code

   ```
   javac GetInputFromKeyboardJOptionPane.java
   java GetInputFromKeyboardJOptionPane
   Enter your name
   CTRL/C to close the application
   ```

3. Modify `GetInputFromKeyboardJOptionPane.java` as following, compile and run the code

   - Make the program to ask the following question
     Please enter your age
   - Display the entered age as following
     - If the age is over 100, display
       Hello <name>
       You are old!
     - Otherwise
       Hello <name>
       You are young!
Chapter 6 Hands-on

6.1 For Loop

1. Create ForLoop.java using your editor of choice

   ```java
   public class ForLoop {
       public static void main( String[] args ){
           String names[] = {"Beah","Bianca","Lance","Belle","Nico","Yza","Gem","Ethan"};
           String searchName = "Yza";
           boolean foundName = false;
           for (int i=0;i<names.length;i++){
               if (names[i].equals(searchName)){
                   foundName = true;
                   break;
               }
           }
           if (foundName)
               System.out.println(searchName +" is found!");
           else
               System.out.println(searchName +" is not found.");
       }
   }
   ```

2. Compile and run the code

   ```bash
   javac ForLoop.java
   java ForLoop
   ```

3. Verify that the result is as following

   ```bash
   C:\lab>java ForLoop
   Yza is found!
   ```

4. Modify ForLoop.java as following, compile and run the code
   - Change the code to use while loop
Chapter 7 Hands-on

7.1 Arrays

1. Create **ArraySample.java** using your editor of choice

   ```java
   public class ArraySample {
       public static void main( String[] args ){
           int[] ages = new int[100];
           for( int i=0; i<ages.length; i++ ){
               System.out.print( ages[i] );
           }
       }
   }
   ```

2. Compile and run the code

   ```bash
   javac ArraySample.java
   java ArraySample
   ```

3. Verify the result is as following

   ```bash
   C:\lab>java ArraySample
   000000000000000000000000000000000000000000000000000000000000000
   00000000000000000
   000000000000000
   ```

4. Modify **ArraySample.java** as following, compile and run the code
   - Just before the for loop that prints out the value of each entry of the ages[] array, create another for loop in which a value of 100 is assigned to the first entry of the array, ages[0], 101 to the next entry of the array, ages[1], and so on

   ```java
   ```
Chapter 8 Hands-on
None

Chapter 9 Hands-on

9.1 Pass-by-Value
1. Create .java using your editor of choice

```java
public class TestPassByValue {
    public static void main(String[] args){
        int i = 10;
        //print the value of i
        System.out.println(i);
        //Call method test
        //and pass i to method test
        test( i );
        // print the value of i. i not changed
        System.out.println(i);
    }
    public static void test(int j){
        // change value of parameter i
        j = 33;
    }
}
```

2. Compile and run the code

```bash
javac TestPassByValue.java  
java TestPassByValue
```

3. Verify the result is as following

```bash
C:\lab>java TestPassByValue
10
10
```
9.2 Pass-by-Reference

1. Create .java using your editor of choice

```java
public class TestPassByReference {
    public static void main(String[] args){
        //create an array of integers
        int [] ages = {10, 11, 12};

        //print array values
        for (int i=0; i<ages.length; i++) {
            System.out.println(ages[i]);
        }

        //call test and pass references to array
        test(ages);

        //print array values again
        for (int i=0; i<ages.length; i++) {
            System.out.println(ages[i]);
        }
    }

    public static void test(int[] arr){
        // change values of array
        for (int i=0; i<arr.length; i++) {
            arr[i] = i + 50;
        }
    }
}
```

2. Compile and run the code

```
javac TestPassByReference.java
java TestPassByReference
```

3. Verify the result is as following

```
C:\lab>java TestPassByReference
10
11
12
50
51
52
```
9.3 Comparing Objects

1. Create **EqualsTest.java** using your editor of choice

```java
class EqualsTest {
    public static void main(String[] arguments) {
        String str1, str2;
        str1 = "Free the bound periodicals.";
        str2 = str1;
        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));
        str2 = new String(str1);
        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("Same object? " + (str1 == str2));
        System.out.println("Same value? " + str1.equals(str2));
    }
}
```

2. Compile and run the code

```
javac EqualsTest.java
java EqualsTest
```

3. Verify the result is as following.

```
C:\lab>java EqualsTest
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true
```

4. Create **EqualsTestInteger.java**

```java
class EqualsTestInteger {
    public static void main(String[] arguments) {
        Integer integer1, integer2;
        integer1 = new Integer(5);
        integer2 = new Integer(5);
        System.out.println("Integer1: " + integer1);
        System.out.println("Integer2: " + integer2);
        System.out.println("Same object? " + (integer1 == integer2));
        integer1 = integer2;
        System.out.println("Integer1: " + integer1);
        System.out.println("Integer2: " + integer2);
        System.out.println("Same object? " + (integer1 == integer2));
        System.out.println("Same value? " + integer1.equals(integer2));
    }
}
```

5. Compile and run the code

```
javac EqualsTestInteger.java
java EqualsTestInteger
```
Chapter 10 Hands-on

10.1 Create your own class

Using Text Editor:

1. Create StudentRecord.java using your editor of choice

```java
public class StudentRecord {
    // instance variables
    private String name;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;

    // static variables
    private static int studentCount = 0;

    /**
     * Returns the name of the student
     */
    public String getName(){
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName(String temp ){
        name =temp;
    }

    /**
     * Computes the average of the english, math and science grades
     */
    public double getAverage(){
        double result =0;
        result =(mathGrade+englishGrade+scienceGrade )/3;
        return result;
    }

    /**
     * Returns the number of instances of StudentRecords
     */
    public static int getStudentCount(){
        return studentCount;
    }
}
```

2. Create StudentRecordExample.java using your editor of choice

```java
public class StudentRecordExample{
    public static void main(String [] args ){
        //create three objects for Student record
        StudentRecord annaRecord =new StudentRecord();
```
StudentRecord beahRecord = new StudentRecord();
StudentRecord crisRecord = new StudentRecord();

// set the name of the students
annaRecord.setName("Anna");
beahRecord.setName("Beah");
crisRecord.setName("Cris");

// print anna's name
System.out.println(annaRecord.getName());

// print number of students
System.out.println("Count=":StudentRecord.getStudentCount());
}

3. Compile and run the code

javac *.java (or javac StudentRecord.java
StudentRecordExample.java)
java StudentRecordExample

4. Verify the result

C:\lab1>java StudentRecordExample
Anna
Count=0
Using NetBeans:

1. Start the NetBeans IDE 4.1 (if you have not done so yet)
   - Windows: Start > All Programs > NetBeans 4.1 > NetBeans IDE or click NetBeans IDE 4.1 desktop icon
   - Solaris/Linux: `<NETBEANS41_HOME>/bin/netbeans`

2. Create a new NetBeans project and StudentRecordExample main class
   - Select File from the menu bar and select New Project.
   - Under Choose Project, select General and Java Application
   - Click Next.
   - Under Name and Location pane, (Figure-10 below)
   - For Project Name field, fill it with StudentRecordExample
   - For Create Main Class field, change it to studentrecordexample.StudentRecordExample (from studentrecordexample.Main)
   - Click Finish

3. Modify the NetBeans generated code
   - Replace the NetBeans generated StudentRecordExample.java code in the source editor with the one you have written before.

4. Write StudentRecord.java
   - Right click StudentRecordExample project node and select New->Java Class. The
New Java Class window appears.
• Under Name and Location pane, for Class Name field, type StudentRecord for Package field, choose studentrecordexample from the drop-down menu (or you can type studentrecordexample)
• Click Finish

5. Modify the NetBeans generated code

• Replace the NetBeans generated StudentRecord.java code in the source editor with the one you have written before.

6. Run StudentRecordExample application

• Right click StudentRecordExample.java node under Hello->Source Packages->studentrecordexample and select Run File (Shift+F6)
• Note that the Output window displays the result
Creating your own:

1. Modify StudentRecord.java as following
   • Add setMathGrade(double grade) method
   • Add setEnglishGrade(double grade) method
   • Add setScienceGrade(double grade) method
   • Add static method called increaseStudentCount(), this method increase the static variable studentCount by 1

2. Modify StudentRecordExample.java as following
   • Create another StudentRecord object, call it myOwnRecord
   • Call setName() method of the myOwnRecord object passing "myOwn" as the value to set
   • Display the name of the myOwnRecord object
   • Set Math grade of myOwnRecord object
   • Set English grade of myOwnRecord object
   • Set Science grade of myOwnRecord object
   • Display the average grade of myOwnRecord

3. Run StudentRecordExample application
10.2 Overloading

Using Text Editor:

1. Modify StudentRecord.java as following. The code fragment that needs to be added is highlighted with bold. Add two overloaded print(..) methods.

   ```java
   public class StudentRecord {
   ...
   
   public void print(String name ){
      System.out.println("Name:"+name);
   }

   public void print(String name, double averageGrade){
      System.out.print("Name:"+name+" ");
      System.out.println("Average Grade:"+averageGrade);
   }
   }
   ```

2. Create StudentRecordExample2.java as follows

   ```java
   public class StudentRecordExample2{
   public static void main(String [] args) {
      StudentRecord annaRecord =new StudentRecord();
      annaRecord.setName("Anna");
      annaRecord.setEnglishGrade(95.5);
      annaRecord.setScienceGrade(100);

      //overloaded methods
      annaRecord.print(annaRecord.getName());
      annaRecord.print(annaRecord.getName(),
                     annaRecord.getAverage());
   }
   }
   ```

3. Compile and run the code. If you experience compile errors, fix the compile errors.

   ```bash
   javac *.java (or javac StudentRecord.java
   StudentRecordExample2.java)
   java StudentRecordExample2
   ```

4. Verify the result

   Name:Anna
   Name:Anna Average Grade:65.16666666666667
**Using NetBeans:**

It is assumed you are using the same NetBeans project you created in 10.1.

1. Modify the StudentRecord.java

2. Create StudentRecordExample2.java
   
   - Right studentrecordexample package node (Not StudentRecordExample project node) and select New->Java Class
   - Under Name and Location pane,
   - for Class Name field, type StudentRecordExamle2
   - Click Finish

3. Modify the NetBeans generated StudentRecordExample2.java
   
   - Replace the code of the NetBeans generated StudentRecordExample2.java with the one of above while leaving the package statement at the top

4. Right click studentrecordexample package node (Not StudentRecordExample project node) and select Compile Package (F9)

5. Right click StudentRecordExamle2 and select Run File
Creating your own:

1. Modify `StudentRecord.java` as following
   - Add another `print()` method which takes the following three parameters
     - name
     - grade average
     - student count

2. Modify `StudentRecordExample2.java` as following
   - Invoke the newly added `print()` method
10.3 Packaging

Please do this exercise at the command line instead of using NetBeans. This is to learn the packaging structure without the help of NetBeans.

0. If have used NetBeans to do the exercise 13 above, please create StudentRecord.java and StudenRecordExample.java as following:

```java
public class StudentRecord {
    // instance variables
    private String name;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;

    // static variables
    private static int studentCount = 0;

    /**
     * Returns the name of the student
     */
    public String getName(){
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName(String temp){
        name =temp;
    }

    /**
     * Computes the average of the english, math and science grades
     */
    public double getAverage(){
        double result =0;
        result =(mathGrade+englishGrade+scienceGrade )/3;
        return result;
    }

    /**
     * returns the number of instances of StudentRecords
     */
    public static int getStudentCount(){
        return studentCount;
    }
}
```
public class StudentRecordExample{
    public static void main(String[] args){
        //create three objects for Student record
        StudentRecord annaRecord = new StudentRecord();
        StudentRecord beahRecord = new StudentRecord();
        StudentRecord crisRecord = new StudentRecord();

        //set the name of the students
        annaRecord.setName("Anna");
        beahRecord.setName("Beah");
        crisRecord.setName("Cris");

        //print anna's name
        System.out.println(annaRecord.getName());

        //print number of students
        System.out.println("Count=\"+StudentRecord.getStudentCount());
    }
}

1. Modify StudentRecord.java as following to add a package statement. The code fragment that needs to be added is in bold characters.

    package studentpackage;
    public class StudentRecord {
        ...
    }

2. Modify StudentRecordExample.java as following to add a package statement. The code fragment that needs to be added is in bold characters.

    package studentpackage;
    public class StudentRecordExample {
        ...
    }

3. Compile code.

    javac StudentRecord.java StudentRecordExample.java

4. Run the code. You will experience an NoClassDefFoundError exception. Think about why you are getting this exception for a moment. It is because the java runtime is trying to find StudentRecordExample.class under studentpackage directory. It is because the StudentRecordExample.java now has a package statement which says the Java class file resides under studentpackage directory.

    • C:\lab>java StudentRecordExample
    Exception in thread "main"
    java.lang.NoClassDefFoundError: StudentRecordExample
        (wrong name: studentpackage/StudentRecordExample)
        at java.lang.ClassLoader.defineClass0(Native
5. Create a new directory called `studentpackage` and then move `StudentRecord.java` and `StudentRecordExample.java` under it.

```
    mkdir \lab\studentpackage
    move \lab\StudentRecordExample.java \lab\studentpackage\StudentRecordExample.java
    move \lab\StudentRecord.java \lab\studentpackage\StudentRecord.java
```

6. Compile code. You will experience compile errors as following. You get this compile error because you are trying to compile the two Java files that are not present in the current directory anymore.

```
    del StudentRecord.class
    del StudentRecordExample.class
    C:\lab>javac StudentRecord.java StudentRecordExample.java
    error: cannot read: StudentRecord.java
    1 error
```

7. Compile the code using a directory structure. The compilation should succeed. Note that the class files are now created under `studentpackage` directory not in the current directory.

```
    javac studentpackage\StudentRecord.java
    studentpackage\StudentRecordExample.java
    C:\lab>dir studentpackage
    Volume in drive C is S3A1256D004
    Volume Serial Number is 447E-6EBC
```
8. Run the code as follows. You will experience NoClassDefFoundError because it is trying to find the class in the current directory instead of in the studentpackage directory.

C:\lab>java StudentRecordExample
Exception in thread "main" java.lang.NoClassDefFoundError: StudentRecordExample

9. Run the code with proper package structure. It should work this time.

C:\lab>java studentpackage.StudentRecordExample
Anna
Count=0

10. Now you thought you should be able to run the application under the studentpackage directory itself so you go into the directory and run the code. And the following is what you will experience. It is because it is still looking for studentpackage/StudentRecordExample.class in the currently directory and it could not find it.

C:\lab>cd studentpackage
C:\lab\studentpackage>java StudentRecordExample
Exception in thread "main" java.lang.NoClassDefFoundError: StudentRecordExample
(wrong name: studentpackage/StudentRecordExample)
at java.lang.ClassLoader.defineClass0(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:539)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:123)
at java.security.Security$ClassLoader.defineClass(Security$ClassLoader.java:139)
at java.net.URLClassLoader$1.run(URLClassLoader.java:221)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader$1.run(URLClassLoader.java:187)
at java.lang.ClassLoader.loadClass(ClassLoader.java:421)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:274)
java.lang.ClassLoader.loadClass(ClassLoader.java:235)
at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:302)
11. Now there is a way you can specify the classpath using -classpath command line option as following:

```java
C:\lab\studentpackage>java -classpath \lab StudentRecordExample
Exception in thread "main" java.lang.NoClassDefFoundError:
StudentRecordExample
```

```java
C:\lab\studentpackage>java -classpath \lab
studentpackage.StudentRecordExample
Anna
Count=0
```

**Creating your own:**

1. Create a class called **Food** under **foodpackage.fruitpackage** package
   - Food.java should have the following package statement at the top
     - package foodpackage.fruitpackage
   - Add a couple of methods of your own

2. Create a class called **FoodMain** under **foodpackage.fruitpackage** package
   - FoodMain class creates an Food object
   - FoodMain class then calls a method of Food object

3. Compile and run the code
Chapter 11 Hands-on

11.1 Inheritance – Constructor

1. Write Person.java

```java
package personpackage;

public class Person {
    private String name;
    private String address;

    public Person(){
        System.out.println("Inside Person:Constructor");
    }

    public Person (String name, String address){
        System.out.println("Inside Person:Constructor 2 receiving two parameters: " + name + ", " + address);
        this.name = name;
        this.address = address;
    }

    public String getName(){
        System.out.println("Person: getName()");
        return name;
    }

    public void setName(String s){
        name = s;
    }

    public String getAddress(){
        return address;
    }

    public void setAddress(String s){
        address = s;
    }
}
```

2. Write Student.java

```java
package personpackage;

public class Student extends Person {
    private String hobby;

    public Student() {
        System.out.println("Inside Student:Constructor");
    }

    public String getHobby() {
        return hobby;
    }

    public void setHobby(String s) {
        hobby = s;
    }
}
```

3. Write Main.java

```java
package personpackage;

public class Main {
    public static void main(String[] args) {
        Student student1 = new Student();
    }
}
```

4. Compile and run the code using a directory structure.

```
cd \lab
javac personpackage/*.java
java personpackage.Main
```

5. Verify the result is as following

```
C:\lab>java personpackage.Main
Inside Person:Constructor
Inside Student:Constructor
```
6. Modify the Student.java as following. The code fragment that needs to be added is in bold characters.

```java
package personpackage;

public class Student extends Person {
    private String hobby;

    public Student(){
        super("Sang", "1 Dreamland");
        System.out.println("Inside Student:Constructor");
    }
    public String getHobby(){
        return hobby;
    }
    public void setHobby(String s){
        hobby = s;
    }
}
```

7. Compile and run the code using a directory structure.

```shell
    cd \
    javac personpackage\.java
    java personpackage.Main
```

8. Verify the result is as following

```shell
    C:\lab>java personpackage.Main
    Inside Person:Constructor 2 receiving two parameters: Sang, 1 Dreamland
    Inside Student:Constructor
```
Using NetBeans:

1. Start the NetBeans IDE 4.1 (if you have not done so yet)
   - Windows: Start > All Programs > NetBeans 4.1 > NetBeans IDE or click NetBeans IDE 4.1 desktop icon
   - Solaris/Linux: `<NETBEANS41_HOME>/bin/netbeans`

2. Create a new NetBeans project and Main.java main class
   - Select File from the menu bar and select New Project.
   - Under Choose Project, select General and Java Application
   - Click Next.
   - Under Name and Location pane, (Figure-10 below)
     - For Project Name field, fill it with PersonPackage
     - Click Finish

3. Replace the code in the NetBeans generated Main.java

4. Create Person.java
   - Right personpackage package node (not PersonPackage project node) and select New->Java Class
   - Under Name and Location pane,
     - for Class Name field, type Person
     - Click Finish

5. Replaced the code in the NetBeans generated Person.java

6. Create Student.java
   - Right personpackage node (not PersonPackage project node) and select New->Java Class
   - Under Name and Location pane,
     - for Class Name field, type Student
     - Click Finish

7. Replaced the code in the NetBeans generated Student.java

8. Right click personpackage package node (not PersonPackage project node) and select Compile Package (F9)

9. Right click Main select Run File

10. Modify the Student.java.

11. Right click personpackage package node (not PersonPackage project node) and select Compile Package (F9)

12. Right click Main select Run File
Creating your own:

1. Write TuftsStudent.java as following
   - TuftsStudent class extends Student class
   - Write a constructor of the TuftsStudent class as following
     ```java
     public TuftsStudent()
     {
         System.out.println("Inside TuftsStudent:Constructor");
     }
     ```

2. Modify the Main.java to create an instance of TuftsStudent class as following
   ```java
   TuftsStudent student2 = new TuftsStudent();
   Student student3 = new TuftsStudent();
   ```

3. Compile and run the code. You should see the following:
   ```
   Inside Person:Constructor
   Inside Student:Constructor
   Inside Person:Constructor
   Inside Student:Constructor
   Inside TuftsStudent:Constructor
   Inside Person:Constructor
   Inside Student:Constructor
   Inside TuftsStudent:Constructor
   ```
11.2 Inheritance - Overriding

1. Modify Main.java as following. The code fragment that needs to be added is in bold characters.

```java
package personpackage;

public class Main {
    public static void main(String [] args ){
        Student student1 =new Student();

        // Calling methods defined in Person class, which is a parent class of Student class
        student1.setName("Sang");
        System.out.println("Calling getName() method: name is "+student1.getName());
    }
}
```

2. Compile and run the code using a directory structure.

   ```
   cd \lab
   javac personpackage/*.java
   java personpackage.Main
   ```

3. Verify the result is as following

   ```
   C:\lab>java personpackage.Main
   Inside Person:Constructor
   Inside Student:Constructor
   Person: getName()
   Calling getName() method: name is Sang
   ```
5. Modify the `Student.java` as following. The code fragment that needs to be added is in bold characters.

```java
package personpackage;

public class Student extends Person {
    private String hobby;
    
    public Student(){
        System.out.println("Inside Student:Constructor");
    }
    
    public String getHobby(){
        return hobby;
    }
    
    public void setHobby(String s){
        hobby = s;
    }
    
    // Override getName() method of the parent class
    public String getName(){
        System.out.println("Student: getName()");
        return "Passionate" + super.getName();
    }
}
```

6. Compile and run the code using a directory structure.

```bash
cd \lab
javac personpackage\*.java
java personpackage.Main
```

7. Verify the result is as following

```
C:\lab>java personpackage.Main
Inside Person:Constructor
Inside Student:Constructor
Student: getName()
Person: getName()
Calling getName() method: name is PassionateSang
```
Using NetBeans:

It is assumed you are using the same NetBeans project you are using the same NetBeans project you created in Chapter 10.

1. Modify the **Main.java**.

2. Right click **personpackage** package node (not PersonPackage project node) and select **Compile Package (F9)**

3. Right click **Main** select **Run File**

4. Modify click the **Student.java**

5. Right click **personpackage** package node (not PersonPackage project node) and select **Compile Package (F9)**

6. Right click **Main** select **Run File**

Creating your own:

1. In your TuftsStudent class, override getHobby() and setHobby() methods of the Student class as follows

   ```java
   public String getHobby(){
       System.out.println("Inside TuftsStudent:getHobby() method");
       return "My hobby is " + super.getHobby();
   }

   public void setHobby(String s){
       System.out.println("Inside TuftsStudent:setHobby() method");
       super.setHobby(s);
   }
   ```

2. Change Main.java to invoke setHobby() and getHobby() methods of the newly created TuftsStudent object instances as follows.

   ```java
   // set hobbies of student2 and student3
   student2.setHobby("swimming");
   student3.setHobby("dancing");

   // get hobbies of student2 and student3
   String hobby2 = student2.getHobby();
   System.out.println("Hobby of student2 "+ hobby2);
   String hobby3 = student3.getHobby();
   System.out.println("Hobby of student3 "+ hobby3);
   ```

3. Compile and run the code. You should see the following result.

```
Inside Person:Constructor
Inside Student:Constructor
Inside Person:Constructor
Inside Student:Constructor
Inside TuftsStudent:Constructor
Inside Person:Constructor
```
Inside Student:Constructor
Inside TuftsStudent:Constructor
Inside TuftsStudent:setHobby() method
Inside TuftsStudent:setHobby() method
Inside TuftsStudent:getHobby() method
Hobby of student2 My hobby is swimming
Inside TuftsStudent:getHobby() method
Hobby of student3 My hobby is dancing
11.3 Polymorphism

1. Write `Person.java`. This is the same `Person.java` as in the previous exercise except the package name. `Person` class is a parent class of both `Student` and `Employee` classes, which you will write in the subsequent steps.

```java
package polypackage;

public class Person {
    private String name;
    private String address;

    public Person() {
        System.out.println("Inside Person:Constructor");
    }

    public Person(String name, String address) {
        System.out.println("Inside Person:Constructor 2
receiving two parameters: "+ name + ", "+ address);
        this.name = name;
        this.address = address;
    }

    public String getName() {
        System.out.println("Person: getName()");
        return name;
    }

    public void setName(String s) {
        name = s;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String s) {
        address = s;
    }
}
```
2. Write `Student.java`. Student class is a subclass of a Person class.

```java
package polypackage;

public class Student extends Person {

    private String hobby;

    public Student()
    {
        System.out.println("Inside Student:Constructor");
    }

    public Student (String name, String address){
        super(name, address);
        System.out.println("Inside Student:Constructor 2 receiving two parameters: "+ name + ", "+ address);
    }

    public String getHobby(){
        return hobby;
    }

    public void setHobby(String s){
        hobby = s;
    }

    // Override getName() method of the parent class
    public String getName(){
        System.out.println("Student: getName()");
        return "Passionate Student " + super.getName();
    }
}
```
3. Write `Employee.java`. Employee class is subclass of Person class.

```java
package polypackage;

public class Employee extends Person {
    private String hobby;

    public Employee(){
        System.out.println("Inside Employee:Constructor");
    }

    public Employee(String name, String address){
        super(name, address);
        System.out.println("Inside Employee:Constructor 2 receiving two parameters: " + name + ", " + address);
    }

    public String getHobby(){
        return hobby;
    }

    public void setHobby(String s){
        hobby = s;
    }

    // Override getName() method of the parent class
    public String getName(){
        System.out.println("Employee: getName()");
        return "Not so Passionate Employee " + super.getName();
    }
}
```
4. Write Main.java

```java
package polypackage;

public class Main {
    public static void main( String[] args ) {
        Person ref;
        Student studentObject = new Student("Sang", "1 Dreamland");
        Employee employeeObject = new Employee("Young", "2 Dreamland");
        System.out.println("\n");
        ref = studentObject; //Person ref. points to a Student object
        String temp1 = ref.getName();
        System.out.println( "temp1 -" + temp1 + "\n" );
        ref = employeeObject; //Person ref. points to an Employee object
        String temp2 = ref.getName();
        System.out.println( "temp2 -" + temp2 + "\n" );
    }
}
```

5. Compile and run the code using a directory structure.

```
cd \lab
javac polypackage/*.java
java polypackage.Main
```

6. Verify the result is as following. Note that depending on what object type the `ref` variable refers to, Employee type or Student type, proper method gets invoked.

```
C:\lab>javac polypackage/*.java
C:\lab>java polypackage.Main
Inside Person: Constructor 2 receiving two parameters: Sang, 1 Dreamland
Inside Student: Constructor 2 receiving two parameters: Sang, 1 Dreamland
Inside Person: Constructor 2 receiving two parameters: Young, 2 Dreamland
Inside Employee: Constructor 2 receiving two parameters: Young, 2 Dreamland

Student: getName()
Person: getName()
temp1 -Passionate Student Sang

Employee: getName()
Person: getName()
temp2 -Not so Passionate Employee Young
```
Using NetBeans:

1. Start the NetBeans IDE 4.1 (if you have not done so yet)

   - Windows: **Start** > **All Programs** > **NetBeans 4.1** > **NetBeans IDE** or click NetBeans IDE 4.1 desktop icon
   - Solaris/Linux: `<NETBEANS41_HOME>/bin/netbeans`

2. Create a new NetBeans project and Main.java main class

   - Select **File** from the menu bar and select **New Project**.
   - Under **Choose Project**, select **General** and **Java Application**
   - Click **Next**.
   - Under **Name and Location** pane, (Figure-10 below)
     - For **Project Name** field, fill it with **PolyPackage**
     - Click **Finish**

3. Replace the code in the NetBeans generated **Main.java**

4. Create Person.java

   - Right polypackage package node (not PolyPackage project node) and select **New->Java Class**
   - Under **Name and Location** pane,
     - for **Class Name** field, type **Person**
     - Click **Finish**

5. Replaced the code in the NetBeans generated **Person.java**

6. Create Student.java

   - Right polypackage node (not PolyPackage project node) and select **New->Java Class**
   - Under **Name and Location** pane,
     - for **Class Name** field, type **Student**
     - Click **Finish**

7. Replaced the code in the NetBeans generated **Student.java**

8. Create Employee.java

   - Right polypackage node (not PolyPackage project node) and select **New->Java Class**
   - Under **Name and Location** pane,
     - for **Class Name** field, type **Employee**
     - Click **Finish**
9. Replaced the code in the NetBeans generated Employee.java

10. Right click polypackage package node (not PolyPackage project node) and select Compile Package (F9)

11. Right click Main select Run File

12. Modify the Student.java

13. Right click polypackage package node (not PolyPackage project node) and select Compile Package (F9)

14. Right click Main select Run File

Creating your own:

1. Create another class called Teacher.java as following
   • Teacher class extends Person class
   • Teacher class also has the following method
     ```java
     // Override getName() method of the parent class
     public String getName(){
        System.out.println("Teacher: getName()");
        return "Maybe Passionate Teacher" + super.getName();
    }
     ```

2. Modify the Main.java in which, getName() method of the Teacher object gets called

3. Compile and run the code. You should see the following result.
   ```
   C:\lab>java polypackage.Main
   Inside Person:Constructor 2 receiving two parameters: Sang, 1 Dreamland
   Inside Student:Constructor 2 receiving two parameters: Sang, 1 Dreamland
   Inside Person:Constructor 2 receiving two parameters: Young, 2 Dreamland
   Inside Employee:Constructor 2 receiving two parameters: Young, 2 Dreamland
   Inside Person:Constructor 2 receiving two parameters: Wende, 21 New York
   Inside Teacher:Constructor 2 receiving two parameters: Wende, 21 New York
   
   Student: getName()
   Person: getName()
   Passionate Student Sang
   temp1 -Passionate Student Sang
   
   Employee: getName()
   Person: getName()
   Not so Passionate Employee Young
   temp2 -Not so Passionate Employee Young
   
   Teacher: getName()
   Person: getName()
   temp3 -Maybe Passionate Teacher Wende
   ```
11.4 Abstract Classes

1. Write abstract class called LivingThing.java

```java
package abstractexercise;

public abstract class LivingThing {
    public void breath(){
        System.out.println("Living Thing breathing...");
    }
    public void eat(){
        System.out.println("Living Thing eating...");
    }

    /**
     * abstract method walk
     * We want this method to be overridden by subclasses of
     * LivingThing
     */
    public abstract void walk();
}
```

2. Write Main.java.

```java
package abstractexercise;

public class Main {
    public static void main( String[] args ) {
        LivingThing x = new LivingThing();
    }
}
```

3. Compile Livingthing.java and Main.java.

```bash
cd \lab
javac abstractexercise\LivingThing.java abstractexercise\Main.java
```

4. Note that you will experience a compile error since you cannot create an object instance from an abstract class.

```bash
C:\lab>javac abstractexercise\LivingThing.java
abstractexercise\Main.java:5: abstractexercise.LivingThing is abstract; cannot be instantiated
    LivingThing x = new LivingThing();
^
1 error
```
5. Write a concrete class called Human.java that extends the abstract LivingThing class.

```java
package abstractexercise;

public class Human extends LivingThing {
    public void walk(){
        System.out.println("Human walks...");
    }
}
```

6. Rewrite Main.java.

```java
package abstractexercise;

public class Main {
    public static void main(String[] args) {
        Human x = new Human();
        x.walk();
        LivingThing y = new Human();
        y.walk();
    }
}
```

7. Compile and run the code using a directory structure.

```
cd \lab
javac abstractexercise/*.java
java abstractexercise.Main
```

8. Verify the result is as following.

```
C:\lab>java abstractexercise.Main
Human walks...
Human walks...
```

Creating your own:

1. Define another abstract method in the LivingThing.java as following.

```java
public abstract void dance(String dancingStyle);
```

2. Implement a concrete method in the Human.java that implements the dance() abstract method.

```java
public void dance(String ds){
    System.out.println("Human dances..." + ds);
}
```

3. Modify the Main.java so that it calls dance(ds) method...
4. Compile and run the code. You should see the following result.

```
C:\lab>java abstractexercise.Main
Human walks...
Human dances in Swing
Human walks...
Human dances in Saturday Night Live
```
11.5 Interfaces 1

1. Write Relation.java which is an Interface.

```java
package interfaceexercise;

public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

2. Write Main.java.

```java
package interfaceexercise;

public class Main {
    public static void main( String[] args ) {
        Relation x = new Relation();
    }
}
```

3. Compile Relation.java and Main.java.

```
cd \lab
javac interfaceexercise\Relation.java
interfaceexercise\Main.java
```

4. Note that you will experience a compile error since you cannot create an object instance from an Interface.

```
C:\lab>javac interfaceexercise\Relation.java
interfaceexercise\Main.java
interfaceexercise\Main.java:5: interfaceexerciseRelation is abstract; cannot be instantiated
    Relation x = new Relation();
          ^
1 error
```
5. Write a concrete class that implements Relation.

    package interfaceexercise;
    public class Line implements Relation {
        private double x1;
        private double x2;
        private double y1;
        private double y2;

        public Line(double x1, double x2, double y1, double y2) {
            this.x1 = x1;
            this.x2 = x2;
            this.y1 = y1;
            this.y2 = y2;
        }

        public double getLength() {
            double length = Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
            return length;
        }

        public boolean isGreater(Object a, Object b) {
            double aLen = ((Line) a).getLength();
            double bLen = ((Line) b).getLength();
            return (aLen > bLen);
        }

        public boolean isLess(Object a, Object b) {
            double aLen = ((Line) a).getLength();
            double bLen = ((Line) b).getLength();
            return (aLen < bLen);
        }

        public boolean isEqual(Object a, Object b) {
            double aLen = ((Line) a).getLength();
            double bLen = ((Line) b).getLength();
            return (aLen == bLen);
        }
    }
6. Rewrite Main.java.

```java
package interfaceexercise;

public class Main {
    public static void main(String[] args) {
        Line line1 = new Line(1.0, 2.0, 1.0, 2.0);
        Line line2 = new Line(2.0, 3.0, 2.0, 3.0);

        boolean b1 = line1.isGreater(line1, line2);
        System.out.println("line1 is greater than line2: " + b1);
        boolean b2 = line1.isEqual(line1, line2);
        System.out.println("line1 is equal with line2: " + b2);

        Line line3 = new Line(1.0, 5.0, 1.0, 5.0);
        boolean b3 = line3.isEqual(line1, line3);
        System.out.println("line1 is equal with line3: " + b3);

        System.out.println("Length of line1 is ", line1.getLength());
        System.out.println("Length of line2 is ", line2.getLength());
        System.out.println("Length of line3 is ", line3.getLength());
    }
}
```

7. Compile and run the code using a directory structure.

```
cd \lab
javac interfaceexercise/*.java
java interfaceexercise.Main
```

8. Verify the result as following:

```
C:\lab>java interfaceexercise.Main
line1 is greater than line2: false
line1 is equal with line3: true
Length of line1 is 1.4142135623730951
Length of line2 is 1.4142135623730951
Length of line3 is 5.656854249492381
```
Creating your own:

1. Create another implementation class called NumberComparison that implements Relation interface.

2. Modify Main class that compares two int type numbers.

3. You should see the following result

```
C:\lab>java interfaceexercise.Main
line1 is greater than line2: false
line1 is equal with line2: true
line1 is equal with line3: false
Length of line1 is 1.4142135623730951
Length of line2 is 1.4142135623730951
Length of line3 is 5.656854249492381

1 is greater than 5 false
1 is equal with 5 false
1 is less than 5 true
```
11.6 Interfaces 2

1. Write PersonInterface.java which is an Interface.

```java
package interfaceexercise2;

public interface PersonInterface {
    public String getName();
    public void setName(String s);
    public String getAddress();
    public void setAddress(String s);
}
```

2. Write PersonImpl.java. PersonImpl class implements PersonInterface Interface.

```java
package interfaceexercise2;

public class PersonImpl implements PersonInterface {
    private String name;
    private String address;

    public PersonImpl() {
        System.out.println("Inside PersonImpl: Constructor");
    }

    public PersonImpl(String name, String address) {
        System.out.println("Inside PersonImpl: Constructor 2 receiving two parameters: ", name + ", ", address);
        this.name = name;
        this.address = address;
    }

    public String getName() {
        System.out.println("PersonImpl: getName()");
        return name;
    }

    public void setName(String s) {
        name = s;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String s) {
        address = s;
    }
}
```
3. Compile PersonInterface.java and PersonImpl.java.

```
cd \lab
javac interfaceexercise2\PersonInterface.java
interfaceexercise2\PersonImpl.java
```


```
package interfaceexercise2;

public interface StudentInterface extends PersonInterface {
    public String getHobby();
    public void setHobby(String s);
}
```

5. Write StudentImpl.java.

```
package interfaceexercise2;

public class StudentImpl implements StudentInterface {
    private String hobby;

    public StudentImpl(){
        System.out.println("Inside StudentImpl:Constructor");
    }

    public String getHobby(){
        System.out.println("StudentImpl: getHobby");
        return hobby;
    }

    public void setHobby(String s){
        hobby = s;
    }
}
```
6. Compile StudentInterface.java and StudentImpl.java. You will experience the compile error. This is because StudentImpl.java did not implement all the abstract methods defined in both StudentInterface and PersonInterface interfaces.

```
C:\lab>javac interfaceexercise2\StudentInterface.java
interfaceexercise2\StudentImpl.java
interfaceexercise2\StudentImpl.java:3:
interfaceexercise2.StudentImpl is not abstract and does not
override abstract method setAddress(java.lang.String) in
interfaceexercise2.PersonInterface
public class StudentImpl implements StudentInterface{
  ^
1 error
```

7. Modify StudentImpl.java. The code fragment that needs to be added is highlighted in bold font.

```
package interfaceexercise2;

public class StudentImpl extends PersonImpl implements
StudentInterface {

  private String hobby;

  public StudentImpl(){
    System.out.println("Inside StudentImpl:Constructor");
  }

  public String getHobby(){
    System.out.println("StudentImpl: getHobby()");
    return hobby;
  }

  public void setHobby(String s){
    hobby = s;
  }
}
```

8. Compile StudentInterface.java and StudentImpl.java. Compilation should succeed.

```
javac interfaceexercise2\StudentInterface.java
interfaceexercise2\StudentImpl.java
```
9. Write Main.java.

```java
package interfaceexercise2;

public class Main {
    public static void main(String[] args) {
        StudentInterface student1 = new StudentImpl();
        student1.setName("Ann");
        String s1 = student1.getName();
        System.out.println("student1's name is " + s1);
        student1.setHobby("Dancing");
        String s2 = student1.getHobby();
        System.out.println("student1's hobby is " + s2);
    }
}
```

10. Compile all the source code and run it.

```
javac interfaceexercise2*.java
java interfaceexercise2.Main
```

11. Verify the result is as following

```
C:\lab>java interfaceexercise2.Main
Inside PersonImpl:Constructor
Inside StudentImpl:Constructor
PersonImpl: getName()
student1's name is Ann
StudentImpl: getHobby()
student1's hobby is Dancing
```
Creating your own:

1. Write `TuftsStudentInterface` interface. It should "extend" `StudentInterface` interface. Define the following new methods inside the `TuftsStudentInterface` interface.

   ```java
   public int add(int x, int y);
   public double multiply(double p, double q);
   ```

2. Write `TuftsStudentImpl` class. It "implements" `TuftsStudentInterface`. It should also "extend" `StudentImpl` class.

3. Modify the Main.java so that it calls add and multiply methods of the `TuftsStudentInterface` interface.

4. You should see the result something like following:

   ```bash
   C:\lab>java interfaceexercise2.Main
   Inside PersonImpl:Constructor
   Inside StudentImpl:Constructor
   PersonImpl: getName()
   student1's name is Ann
   StudentImpl: getHobby()
   student1's hobby is Dancing

   Inside PersonImpl:Constructor
   Inside StudentImpl:Constructor
   PersonImpl: getName()
   tuftsstudent1's name is Mario
   StudentImpl: getHobby()
   tuftsstudent1's hobby is Tennis
   StudentImpl: add()
   tuftsstudent1's addition is 11
   StudentImpl: multiply()
   tuftsstudent1's multiplication is 311.2
   ```
Chapter 12 Hands-on

12.1 Exception Handling

1. Write ExceptionExample.java

```java
package exceptionexercise;

public class ExceptionExample {
    public static void main( String[] args ){
        try{
            System.out.println( args[1] );
        } catch( ArrayIndexOutOfBoundsException exp ){
            System.out.println("Exception caught!");
        }
    }
}
```

2. Compile and run the code using a directory structure.

```bash
cd \lab
javac exceptionexercise/*.java
java exceptionexercise.ExceptionExample
```

3. Verify the result

```bash
C:\lab>java exceptionexercise.ExceptionExample
Exception caught!
```
Answers to Hands-on Exercises

10.1 Create your own class

StudentRecordExample.java

```java
public class StudentRecordExample {
    public static void main(String[] args) {
        // create three objects for Student record
        StudentRecord annaRecord = new StudentRecord();
        StudentRecord.increaseStudentCount();
        StudentRecord beahRecord = new StudentRecord();
        StudentRecord.increaseStudentCount();
        StudentRecord crisRecord = new StudentRecord();
        StudentRecord.increaseStudentCount();
        StudentRecord myOwnRecord = new StudentRecord();
        StudentRecord.increaseStudentCount();
        // set the name of the students
        annaRecord.setName("Anna");
        beahRecord.setName("Beah");
        crisRecord.setName("Cris");
        myOwnRecord.setName("myOwn");
        // print anna's name
        System.out.println(annaRecord.getName());
        System.out.println(myOwnRecord.getName());
        // set grades
        myOwnRecord.setMathGrade(60.2);
        myOwnRecord.setEnglishGrade(90.2);
        myOwnRecord.setScienceGrade(70.2);
        // print number of students
        System.out.println("Count="+StudentRecord.getStudentCount());
        System.out.println("Average of my Own="+myOwnRecord.getAverage());
    }
}
```
public class StudentRecord {
    // instance variables
    private String name;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;

    // static variables
    private static int studentCount = 0;

    /**
     * Returns the name of the student
     */
    public String getName(){
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName(String temp ){
        name =temp;
    }

    public void setMathGrade(double grade){
        mathGrade = grade;
    }

    public void setEnglishGrade(double grade){
        englishGrade = grade;
    }

    public void setScienceGrade(double grade){
        scienceGrade = grade;
    }

    public static void increaseStudentCount(){
        studentCount++;         
    }

    /**
     * Computes the average of the english,math and science
     * grades
     */
    public double getAverage(){
        double result =0;
        result =(mathGrade+englishGrade+scienceGrade )/3;
        return result;
    }

    /**
     * returns the number of instances of StudentRecords
     */
    public static int getStudentCount(){
        return studentCount;
    }
}
10.2 Overloading

StudentRecord.java

```java
public class StudentRecord {
    // instance variables
    private String name;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;

    // static variables
    private static int studentCount = 0;

    /**
     * Returns the name of the student
     */
    public String getName(){
        return name;
    }

    /**
     * Changes the name of the student
     */
    public void setName(String temp ){
        name =temp;
    }

    public void setMathGrade(double grade){
        mathGrade = grade;
    }

    public void setEnglishGrade(double grade){
        englishGrade = grade;
    }

    public void setScienceGrade(double grade){
        scienceGrade = grade;
    }

    public static void increaseStudentCount(){
        studentCount++;
    }

    /**
     * Computes the average of the english,math and science
     * grades
     */
    public double getAverage(){
        double result =0;
        result = (mathGrade+englishGrade+scienceGrade )/3;
        return result;
    }

    /**
     * Returns the number of instances of StudentRecords
     */
    public static int getStudentCount(){
        return studentCount;
    }
}
```
public void print(String name ){
    System.out.println("Name:"+name);
}

public void print(String name, double averageGrade){
    System.out.print("Name:"+name+" ");
    System.out.println("Average Grade:"+averageGrade);
}

public void print(String name, double averageGrade, int studentCount){
    System.out.print("Name:"+name+" ");
    System.out.println("Average Grade:"+averageGrade);
    System.out.println("Student count:"+studentCount);
}

StudentExample2.java
public class StudentRecordExample2{
    public static void main(String [] args) {
        StudentRecord annaRecord =new StudentRecord();
        annaRecord.setName("Anna");
        annaRecord.setEnglishGrade(95.5);
        annaRecord.setScienceGrade(100);

        //overloaded methods
        annaRecord.print(annaRecord.getName());
        annaRecord.print(annaRecord.getName(),
            annaRecord.getAverage());
        annaRecord.print(annaRecord.getName(),
            annaRecord.getAverage(),
            annaRecord.getStudentCount());
    }
}
10.3 **Packaging**

1. Create `Food.java` under a proper directory structure

   ```
   cd \lab
   mkdir foodpackage
   mkdir foodpackage\fruitpackage
   jedit foodpackage\fruitpackage\Food.java
   
   **Food.java**
   ```
   ```
   package foodpackage.fruitpackage;
   public class Food {
       //instance variables
       private String color = "white";
       public String getColor(){
           return color;
       }
   }
   ```

2. Create `FoodMain.java` under a proper directory structure

   **FoodMain.java**
   ```
   package foodpackage.fruitpackage;
   public class FoodMain{
       public static void main(String [] args ){
           Food food1 =new Food();
           System.out.println("Color of the food ="+food1.getColor());
       }
   }
   ```

3. Compile and run the code

   ```
   cd \lab
   javac foodpackage\fruitpackage\*.java
   java foodpackage.fruitpackage.FoodMain
   ```

4. Verify the result

   ```
   C:\lab>java foodpackage.fruitpackage.FoodMain
   Color of the food =white
   ```
11.1 Inheritance – Constructor

TuftsStudent.java

```java
public class TuftsStudent extends Student {
    /** Creates a new instance of TuftsStudent */
    public TuftsStudent() {
        System.out.println("Inside TuftsStudent:Constructor");
    }
}
```

Main.java

```java
public class Main {
    public static void main(String [] args) {
        Student student1 = new Student();
        TuftsStudent student2 = new TuftsStudent();
        Student student3 = new TuftsStudent();
    }
}
```

11.2 Inheritance - Overriding

TuftsStudent.java

```java
package personpackage;

/**
 * @author sang
 */
public class TuftsStudent extends Student {
    /** Creates a new instance of TuftsStudent */
    public TuftsStudent() {
        System.out.println("Inside TuftsStudent:Constructor");
    }

    public String getHobby() {
        System.out.println("Inside TuftsStudent:getHobby() method");
        return "My hobby is " + super.getHobby();
    }

    public void setHobby(String s) {
        System.out.println("Inside TuftsStudent:setHobby() method");
        super.setHobby(s);
    }
}
```
Main.java

package personpackage;

public class Main {

    public static void main(String [] args ){
        Student student1 =new Student();
        TuftsStudent student2 =new TuftsStudent();
        Student student3 =new TuftsStudent();

        // set hobbies of student2 and student3
        student2.setHobby("swimming");
        student3.setHobby("dancing");

        // get hobbies of student2 and student3
        String hobby2 = student2.getHobby();
        System.out.println("Hobby of student2 " + hobby2);
        String hobby3 = student3.getHobby();
        System.out.println("Hobby of student3 " + hobby3);
    }
}

Introduction to Programming I
# 11.3 Polymorphism

**Teacher.java**

```java
package polypackage;

public class Teacher extends Person {
    private String hobby;

    public Teacher() {
        System.out.println("Inside Teacher:Constructor");
    }

    public Teacher(String name, String address) {
        super(name, address);
        System.out.println("Inside Teacher:Constructor 2 receiving two parameters: "+ name + ", "+ address);
    }

    public String getHobby() {
        return hobby;
    }

    public void setHobby(String s) {
        hobby = s;
    }

    // Override getName() method of the parent class
    public String getName() {
        System.out.println("Teacher: getName()");
        return "Maybe Passionate Teacher " + super.getName();
    }
}
```

**Main.java**

```java
package polypackage;

public class Main {
    public static void main(String[] args) {
        Person ref;
        Student studentObject = new Student("Sang", "1 Dreamland");
        Employee employeeObject = new Employee("Young", "2 Dreamland");
        Teacher teacherObject = new Teacher("Wende", "21 New York");
        System.out.println("\n");
        ref = studentObject; //Person ref. points to a
        // Student object
        System.out.println("getName of Student class is called");
        String temp1 = ref.getName();
        System.out.println(temp1);
        System.out.println("temp1 =" + temp1 + "\n");
        ref = employeeObject; //Person ref. points to an
        // Employee object
    }
}
```
//getName of Employee class is called
String temp2 = ref.getName();
System.out.println( temp2 );
System.out.println( "temp2 -" + temp2 + "\n" );

ref = teacherObject; //Person ref. points to an
// Teacher object

//getName of Employee class is called
String temp3 = ref.getName();
System.out.println( "temp3 -" + temp3 + "\n" );

11.4 Abstract Classes

LivingThing.java

package abstractexercise;

class LivingThing {
    public void breath()
    {
        System.out.println("Living Thing breathing...");
    }

class LivingThing {
    public void eat()
    {
        System.out.println("Living Thing eating...");
    }

    /**
     * abstract method walk
     * We want this method to be overridden by subclasses of
     * LivingThing
     */
    public abstract void walk();

    /**
     * abstract method dance
     * We want this method to be overridden by subclasses of
     * LivingThing
     */
    public abstract void dance(String dancingStyle);
**Human.java**

```java
package abstractexercise;

public class Human extends LivingThing {
    public void walk(){
        System.out.println("Human walks...");
    }
    public void dance(String ds){
        System.out.println("Human dances in " + ds);
    }
}
```

**Main.java**

```java
package abstractexercise;

public class Main {
    public static void main( String[] args ) {
        Human x = new Human();
        x.walk();
        x.dance("Swing");

        LivingThing y = new Human();
        y.walk();
        y.dance("Saturday Night Live");
    }
}
```
11.5 Interfaces 1

NumberComparison.java

package interfaceexercise;

public class NumberComparison implements Relation {

    public boolean isGreater(Object a, Object b) {
        Integer ai = (Integer) a;
        Integer bi = (Integer) b;
        return (ai.intValue() > bi.intValue());
    }

    public boolean isLess(Object a, Object b) {
        Integer ai = (Integer) a;
        Integer bi = (Integer) b;
        return (ai.intValue() < bi.intValue());
    }

    public boolean isEqual(Object a, Object b) {
        Integer ai = (Integer) a;
        Integer bi = (Integer) b;
        return (ai.intValue() == bi.intValue());
    }
}

Main.java

package interfaceexercise;

public class Main {
    public static void main(String[] args) {

        Line line1 = new Line(1.0, 2.0, 1.0, 2.0);
        Line line2 = new Line(2.0, 3.0, 2.0, 3.0);

        boolean b1 = line1.isGreater(line1, line2);
        System.out.println("line1 is greater than line2: "+ b1);
        boolean b2 = line1.isEqual(line1, line2);
        System.out.println("line1 is equal with line2: "+ b2);

        Line line3 = new Line(1.0, 5.0, 1.0, 5.0);
        boolean b3 = line3.isEqual(line1, line3);
        System.out.println("line1 is equal with line3: "+ b3);

        System.out.println("Length of line1 is "+ line1.getLength());
        System.out.println("Length of line2 is "+ line2.getLength());
        System.out.println("Length of line3 is "+ line3.getLength());

        System.out.println("");

        Relation r1 = new NumberComparison();

        Integer x = new Integer(1);
        Integer y = new Integer(5);
boolean b4 = r1.isGreater(x, y);
System.out.println(x + " is greater than " + y + " " +
    b4);

boolean b5 = r1.isEqual(x, y);
System.out.println(x + " is equal with " + y + " " +
    b5);

boolean b6 = r1.isLess(x, y);
System.out.println(x + " is less than " + y + " " +
    b6);

}
11.6 Interfaces 2

TuftsStudentInterface.java

```java
package interfaceexercise2;

public interface TuftsStudentInterface extends StudentInterface {
    public int add(int x, int y);
    public double multiply(double p, double q);
}
```

TuftsStudentImpl.java

```java
package interfaceexercise2;

public class TuftsStudentImpl extends StudentImpl implements TuftsStudentInterface {
    public int add(int x, int y) {
        System.out.println("StudentImpl: add()");
        return x + y;
    }

    public double multiply(double p, double q) {
        System.out.println("StudentImpl: multiply()");
        return p * q;
    }
}
```

Main.java

```java
package interfaceexercise2;

public class Main {
    public static void main(String[] args) {
        StudentInterface student1 = new StudentImpl();
        student1.setName("Ann");
        String s1 = student1.getName();
        System.out.println("student1's name is "+s1);
        student1.setHobby("Dancing");
        String s2 = student1.getHobby();
        System.out.println("student1's hobby is "+s2);

        System.out.println(");
        TuftsStudentInterface tuftsstudent1 = new TuftsStudentImpl();
        tuftsstudent1.setName("Mario");
        String s3 = tuftsstudent1.getName();
        System.out.println("tuftsstudent1's name is "+s3);
        tuftsstudent1.setHobby("Tennis");
    }
}
String s4 = tuftsstudent1.getHobby();
System.out.println("tuftsstudent1's hobby is " + s4);

int i1 = tuftsstudent1.add(5, 6);
System.out.println("tuftsstudent1's addition is " + i1);

double d1 = tuftsstudent1.multiply(10.0, 31.12);
System.out.println("tuftsstudent1's multiplication is " + d1);
Appendix F: Additional Exercises

Chapter 1: Introduction to Computer Programming
No Entries

Chapter 2: Introduction to Java
No Entries

Chapter 3: Getting to Know your Programming Environment
No Entries
Chapter 4: Programming Fundamentals

1. Create a program that will compute the electric bill of a person given the following ranges:

<table>
<thead>
<tr>
<th>Range</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 99 kilowatts</td>
<td>P10.00/kilowatt</td>
</tr>
<tr>
<td>100-249 kilowatt</td>
<td>P50.00/kilowatt after the first 99 kilowatt hours</td>
</tr>
<tr>
<td>250 or more</td>
<td>P 100 pero kilowatt every kilowatt hour succeeding</td>
</tr>
</tbody>
</table>

The program must trap erroneous values before allowing the user to continue. Erroneous values are negative numbers and zero.

Sample outputs:
- Enter number of kilowatt hours: 10
  Total Electric bill is P 100.00
- Enter number of kilowatt hours: 100
  Total Electric bill is P5000.00
- Enter number of kilowatt hours: 251
  Total Electric bill is 8690.00
- Enter number of kilowatt hours: 0
- Enter number of kilowatt hours: -4
  Total Electric bill: P100.00

2. Write a program that reads in three whole numbers and outputs the average of the three numbers.

3. Write a program that converts degrees Celsius to Fahrenheit using the formula
   \[ \text{degreesC} = \frac{5(\text{degreesF}) - 32}{9}. \]
   Prompt the user to enter a temperature in degrees Fahrenheit (just a whole number of degrees, without a fractional part) and print out the equivalent Celsius temperature including the fractional part to at least one decimal point.

   A possible dialog might be:
   - Enter a temperature in degrees Fahrenheit: 72
     72 degrees Fahrenheit = 22.2 degrees Celsius

4. Bunyan Lumber Co. needs to create a table of the engineering properties of its lumber. The dimensions of the wood are given as the base and the height in inches. Engineers need to know the following information about lumber:
   - cross-sectional area: base * height
   - moment inertia: \((\text{base} \times \text{height}^3) / 12\)
   - section modulus: \((\text{base} \times \text{height}^2) / 6\)

   The owner makes lumber with base sizes of 2, 4, 6, 8, and 10 inches. The height sizes are 2, 4, 6, 8, 10, and 12 inches. Produce a table with appropriate headings to show these values and the computed engineering properties. The first part of the table’s outline is shown.

<table>
<thead>
<tr>
<th>Lumber size</th>
<th>Cross-sectional area</th>
<th>Moment of inertia</th>
<th>Section Modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 x 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Write a program for an Automatic Teller Machine that dispense money. The user should enter the amount desired (a multiple of 10 dollars) and the machine dispenses this amount using the least number of bills. The bills dispensed are 50s, 20s and 10s. Write a function that determines how many each kind of bill to dispense.

6. Create a program that stores 100 integers in an array, get the sum of all the numbers in the array using loops and outputs the result.
The output should display the following:
0 1 2 3 4 5 6 7 .................... 99
The sum of numbers 1-100 is 4950.
Chapter 5: Getting Input from the keyboard

1. Using BufferedReader class and io package, create a program that will accept input for the students information, and displays the data inputted in JOptionPane class.
   - Student ID: [Your ID]
   - Name: [Your Name]
   - Course: [Your course] and [Your Year]

2. Create a program that will ask input from the user the 4 primitive data types and displays the data inputted by the user.

3. Make a program to compute the area of a circle with a radius 10 \( \text{area} = 2 \times \pi \times \text{rad} \).
   Given: \( \text{rad} = 10 \) Area = 19 square unit
   Use the BufferedReader class to ask the user for an input string and display it on the screen.

4. Create a program that creates 4 main menus:
   1) ADD    2) SUBTRACT    3) MULTIPLY    4) DIVIDE.
   Using for statement create a java program that will display the multiplication table from 1-10.
Chapter 6: Control Structures

1. Write a program that will accept the age of the user to determine whether he/she is qualified or not qualified to vote.

2. Write a program to read in a list of nonnegative integers and output: the largest integer, the smallest integer, and the average of all the integers. The end of the input is indicated by the user entering a negative sentinel value. Note that the sentinel value is not used in finding the largest, smallest, or average. It is only an end marker. The average should be a value of type double so that the average is computed with a fractional part.

3. Write a program to read a list of exam scores (integer percent scores, in the rage 0 to 100) and output the total number of grades and then number of grades in each letter-grade category (90 to 100 = A, 80 to 89 = B, 70 to 79 = C, 60 to 69 = D, and 0 to 59 = F). The end of the input is indicated by entering a negative score as a sentinel value (the negative value is used only to end the loop, so do not use it in the calculations). For example, if the input is: 98 87 86 85 85 78 73 72 72 70 66 63 50 -1 the output would be Total number of grades = 14 Number of A's = 1 Number of B's = 4 Number of C's = 6 Number of D's = 2 Number of F's = 1

4. Write a program that takes as input a bank account balance and an interest rate and outputs the value of the account in 10 years. The output should show the value of the account for three different methods of compounding interest: annually, monthly, and daily.
   When compounded annually, the interest is added once per year at the end of the year.
   When compounded monthly the interest is added in 12 times per year. When computed daily, the interest is added 365 times per year. You do not have to worry about leap years. Assume all years have 365 days. On annual interest, you can assume that the interest is posted exactly one year from the date of deposit. In other words, you do not have to worry about interest being posted on a specific day of the year, like December 31. Similarly, you can assume monthly interest is posted exactly one month after it is entered. Since the account earns interest on the interest, the account should have a higher balance when interest is posted more frequently. Be sure to adjust the interest rate for the time period of the interest. If the rate is 5%, then when posting monthly interest, you use (5/12%). When posting daily interest, you use (5/365)%. Do your calculations using a loop that adds in the interest for each time period. (Do not use some sort of algebraic formula). Your program should have an outer loop that allows the user to repeat this calculation for a new balance and interest rate. The calculation is repeated until the user indicates that she/he wants to end the program.

5. Write a program to take two numbers as input data and display their sum, their difference, their product and their quotient.

Data Requirements

Problem Inputs: 
  double x, y /* two items */

Problem Outputs:
  double sum /* sum of x and y */
  double difference /* difference of x and y */
  double product /* product of x and y */
  double quotient /* quotient of x divided by y */
6. Write a program that predicts the score needed on a final exam to achieve a desired grade in a course. The program should interact with the user as follows:
   Enter desired grade> B
   Enter minimum average required> 79.5
   Enter current average in course> 74.6
   Enter how much the final counts as a percentage of the course grade> 25
   You need a score of 94.20 on the final to get a B.

7. Write a program that takes as input the numerators and denominators of two fractions. Your program should display the numerator and denominator of the fraction that represents the product of the two fractions. Also, display the percent equivalent of the resulting product.

8. In shopping for a new house, you must consider several factors. In this problem the initial cost of the house, the estimated annual fuel costs, and the annual tax rate are available. Write a program that will determine the total cost of a house after a five-year period and run the program for each of the following sets of data.

<table>
<thead>
<tr>
<th>Initial House Cost</th>
<th>Annual Fuel Cost</th>
<th>Tax Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>67,000</td>
<td>2,300</td>
<td>0.025</td>
</tr>
<tr>
<td>62,000</td>
<td>2,500</td>
<td>0.025</td>
</tr>
<tr>
<td>75,000</td>
<td>1,850</td>
<td>0.020</td>
</tr>
</tbody>
</table>

To calculate the house cost, add the initial cost to the fuel cost for five years, then add the taxes for five years. Taxes for one year are computed by multiplying the tax rate by the initial cost. Write and call a function that displays instructions to the program user.

9. Given the lengths a, b, c of the sides of a triangle, write a function to compute the area A of the triangle. The formula for computing A is given by:
   $$A = \sqrt{s(s-a)(s-b)(s-c)}$$
   where s is the semiperimeter of the triangle. Write a driver program to get values for a, b, and c and call your function to compute A. The driver should print A, a, b, and c.

10. Implement the following decision table using a nested if statement. Assume that the grade point average is within the range 0.0 through 4.0.

<table>
<thead>
<tr>
<th>Grade Point Average</th>
<th>Transcript Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 – 0.99</td>
<td>Failed semester – registration suspended</td>
</tr>
<tr>
<td>1.0 – 1.99</td>
<td>On Probation for next semester</td>
</tr>
<tr>
<td>2.0 – 2.99</td>
<td>(no message)</td>
</tr>
<tr>
<td>3.0 – 3.49</td>
<td>Dean’s list for semester</td>
</tr>
<tr>
<td>3.5 – 4.00</td>
<td>Highest honors for semester</td>
</tr>
</tbody>
</table>

11. Implement the following decision table using a multiple – alternative if statement. Assume that the wind speed is given as an integer.

<table>
<thead>
<tr>
<th>Wind Speed (mph)</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>below 25</td>
<td>not a strong wind</td>
</tr>
<tr>
<td>25 – 38</td>
<td>strong wind</td>
</tr>
<tr>
<td>39 – 54</td>
<td>gale</td>
</tr>
<tr>
<td>55 – 72</td>
<td>whole gale</td>
</tr>
<tr>
<td>above 72</td>
<td>hurricane</td>
</tr>
</tbody>
</table>
12. The Air Force has asked you to write a program to label supersonic aircraft as military or civilian. Your program is to be given the plane’s observed speed in km/h and its estimated length in meters. For planes traveling in excess of 1100 km/h, you will label those longer than 52 meters “civilian” and shorter aircraft as “military”. For planes traveling at slower speeds, you will label those longer than 52 meters “civilian” and shorter aircraft as “military”. For planes traveling at slower speeds, you will issue an “aircraft type unknown” message.

13. Write a switch statement that assigns to the variable lumens the expected brightness of a standard light bulb whose wattage has been stored in watts. Use this table:

<table>
<thead>
<tr>
<th>Watts</th>
<th>Brightness (in Lumens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>125</td>
</tr>
<tr>
<td>25</td>
<td>215</td>
</tr>
<tr>
<td>40</td>
<td>500</td>
</tr>
<tr>
<td>60</td>
<td>880</td>
</tr>
<tr>
<td>75</td>
<td>1000</td>
</tr>
<tr>
<td>100</td>
<td>1675</td>
</tr>
</tbody>
</table>

Assign –1 to lumens if the value of watts is not in the table.

14. While spending the summer as a surveyor’s assistant, you decide to write a program that transforms compass headings in degrees (0 to 360) to compass bearings. A compass bearing consists of three items: the direction you face (north or south), an angle between 0 and 90 degrees, and the direction you turn walking (east or west). For example, to get the bearing for a compass heading of 110.0 degrees, you would first face due south (180 degrees) and then turn 70.0 degrees east (180.0-70.0 is 110.0). Therefore, the bearings is South 70.0 degrees East. Be sure to check the input for invalid compass headings.

15. The National Earthquake Information Center has asked you to write a program implementing the following decision table to characterize an earthquake based on its Richter scale number.

<table>
<thead>
<tr>
<th>Richter Scale Number (n)</th>
<th>Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n &lt; 5.0</td>
<td>Little or damage</td>
</tr>
<tr>
<td>5.0 &gt;= n &lt; 5.5</td>
<td>Some damage</td>
</tr>
<tr>
<td>5.5 &gt;= n &lt;6.5</td>
<td>Serious damage: walls may crack or fall</td>
</tr>
<tr>
<td>6.5 &gt;= n &lt;7.5</td>
<td>Disaster: houses and buildings may collapse</td>
</tr>
<tr>
<td>higher</td>
<td></td>
</tr>
</tbody>
</table>

16. Write a program that takes a classroom number, its capacity, and the size of the class enrolled so far and prints an output line showing the classroom number, the capacity, both the number of seats filled and the number available, and a message indicating whether the class is filled or not. Display the following headings before the requested output line.

- Room
- Capacity
- Enrollment
- Empty seats
- Filled/Not Filled
Display each part of the output line under the appropriate column heading. Test your program four times using the following classroom data:

<table>
<thead>
<tr>
<th>Room</th>
<th>Capacity</th>
<th>Enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>426</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>327</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>420</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>317</td>
<td>100</td>
<td>90</td>
</tr>
</tbody>
</table>

17. Write a program that takes the x-y coordinates of a point in the Cartesian plane and prints the message telling either an axis on which the point lies or the quadrant in which it is found. Sample lines of output: (-1.0, -2.5) is in quadrant III (0.0, 4.8) is on the y axis

18. Write a program that determines the day number 1 to 366 in a year for a date that is provided as input data. As an example, January 2, 1994 is day 1. December 31, 1993 is day 365. December 31, 1996 is day 366, since 1996 is a leap year. A year is a leap year if it is divisible by four, except that any year divisible by 100 is a leap year only if it is divisible by 400. Your program should accept the month, day and year as integers. Include a function leap that returns 1 if called with a leap year, 0 otherwise.

19. Write a program that will find the smallest, largest, and average values a collection of N numbers. Get the value of N before scanning each value in the collection of N numbers.

20. The Fibonacci sequence is a sequence of numbers beginning

1, 1, 2, 3, 5, 8, 13, 21, ....

The first two elements are defined to be 1. Each of the other elements is the sum of its two predecessors. Write a program that takes a value for n in the range of 1 to 30 and displays the first n elements of the Fibonacci sequence.

21.

a. Write a program to process a collection of daily high temperature. Your program should count and print the number of hot days(high temperature 85 or higher), the number of pleasant days(high temperature 60-84), and the number of cold days(high temperatures less than 60. It should also display the category of each temperature. Test your program on the following data:

<table>
<thead>
<tr>
<th>55</th>
<th>62</th>
<th>68</th>
<th>74</th>
<th>59</th>
<th>45</th>
<th>41</th>
<th>58</th>
<th>60</th>
<th>67</th>
<th>65</th>
<th>78</th>
<th>82</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>91</td>
<td>92</td>
<td>90</td>
<td>93</td>
<td>87</td>
<td>78</td>
<td>79</td>
<td>72</td>
<td>68</td>
<td>61</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

b. Modify your program to display the average temperature (a double number) at the end of the run.
Chapter 7: Java Arrays

1. Create a program that will accept a decimal value input and then choose to either convert it to one of the following number systems:

   1 – Decimal to Binary
   2 – Decimal to Octal
   3 – Decimal to Hexadecimal

USE ARRAYS TO FORM THE NEWLY CONVERTED NUMBER

Sample Output:
   Enter a positive decimal value: 14
   value to which number system:
   1 – Decimal to Binary
   2 – Decimal to Octal
   3 – Decimal to Hexadecimal
   Enter choice: 1
   Equivalent of 14 in binary is 1110

2. Create a program that will store student information in two dimensional array studentarr[3][3], the program should be able to display the names of the students and their corresponding grades and averages.

   Studentarray[3][3]={{“peter”, 75,77},
   {“clark”, 78, 80},
   {“logan”,82,84}};

3. Make a program that will display a 12 X 12 multiplication table that is stored in a two dimensional array.

4. Write a program that store an input list of 10 integers in an array; then display a table similar to the following showing each data value and what percentage each value is of the total of all 10 values

<table>
<thead>
<tr>
<th>n</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4.00</td>
</tr>
<tr>
<td>12</td>
<td>6.00</td>
</tr>
<tr>
<td>18</td>
<td>9.00</td>
</tr>
<tr>
<td>25</td>
<td>12.50</td>
</tr>
<tr>
<td>24</td>
<td>12.00</td>
</tr>
<tr>
<td>30</td>
<td>15.00</td>
</tr>
<tr>
<td>28</td>
<td>14.00</td>
</tr>
<tr>
<td>22</td>
<td>11.00</td>
</tr>
<tr>
<td>23</td>
<td>11.50</td>
</tr>
<tr>
<td>10</td>
<td>5.00</td>
</tr>
</tbody>
</table>
Chapter 8: Command-line Arguments

1. Create a program that will accept two arguments (arg[0], arg[1]) as a username and password to open a program. The user should be able to input the correct arguments otherwise, the program will display an error. The usernames and passwords are hardcoded in a 2xn array (hardcode n username/password pairs).

2. Create a program that will accept input for 10 integers using command line arguments. The program should display the characters inputted in ascending and descending order. The program should display similar output given below.
   Input 10 characters: 12 32 2 4 65 7 15 9 1 23
   Ascending Order: 1 2 4 7 9 12 15 23 32 67
   Descending Order: 67 32 23 15 12 9 7 4 2 1

3. Make a program that will ask the user to supply 3 numbers arguments on the command line and display it in ascending order.

4. There are exactly 2.54 centimeters to an inch. Write a program that takes a number of inches from the command line and converts it to centimeters and vice versa.
Chapter 9: Working with the Java Class Library

1. Create a program that will accept two strings and determine whether the two strings are equal. The two strings should be equal even if they are of different cases.

   Sample output:
   Enter a string: kagi
   Enter another string: KAGI
   Output: The two strings are equal!

   Sample output:
   Enter a string: Heller
   Enter another string: Hallo!
   Output: The two strings are different

2. Write a grading program for each class with the following grading policies:
   a. There are two quizzes, each graded on the basis of 10 points.
   b. There is one midterm exam and one final exam, each graded on the basis of 100 points.
   c. The final exam counts for 50% of the grade, the midterm counts for 25%, and the two quizzes together count for a total of 25%.(Do not forget to normalize the quiz scores. They should be converted to a percent before they are average in.) Any grade of 90 or more is an A, any grade of 80 or more (but less than 90 is B, any grade of 70 or more (but less than 80) is a C, any grade of 60 or more (but less than 70) is a D, and any grade below 60 is an F. The program will read in the student’s scores and output the student’s record, which consists of two quiz and two exam scores as well as the student’s overall numeric score for the entire course and final letter grade.

   Define and use a class for the student record. The class should have instance variables for the quizzes, midterm, final, course overall numeric score, and course final letter grade. The overall numeric score is a number in the range 0 to 100, which represents the weighted average of the student’s work. The class should have input and output methods. The input method should not ask for the final numeric grade nor should it ask for the final letter grade. The class should have methods to compute the overall numeric grade and the final letter grade. These last two methods will be void methods that set the appropriate instance variables. Remember, one method can call another method. If you prefer, you can define a single method that sets both the overall numeric score and the final letter grade, but if you do this, use a helping method. Your program should use all the methods we discussed. Your class should have a reasonable set of accessor and mutator methods, whether or not your program uses them. You may add other methods if you wish.

3. Create a class that graphs the grade distribution (number of A’s, B’s, C’s, D’s, and F’s) horizontally by printing lines with proportionate numbers of asterisks corresponding to the percentage of grades in each category. Write methods to set the number of each letter grade; red the number of each letter grade, return the total number of grades, return the percent of each letter grade as a whole number between 0 and 100, inclusive; and draw the graph. Set it up so that 50 asterisks correspond to 100% (each one corresponds 2%), include a scale on the horizontal axis indicating each 10% increment from 0 to 100%, and label each line with its letter grade. For example, if there are 1 A’s, 4 B’s, 6 C’s, 2 D’s and 1 F, the total number of grades is 14, the percentage of A’s is 7, percentage of B’s is 29, percentage of C’s is 43, percentage of D’s is 14, and percentage of F’s is 7. The A row would contain 4 asterisks (7% of 50 rounded to the nearest
integer), the B row 14, the C row 21, the D row 7, and the F row 4, so the graph would look like this

<table>
<thead>
<tr>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Implement a class called MyDate containing 3 integer data members (for month, day and year). Implement a constructor and getter and setter methods for your class. Override the toString and equals method of the Object class. Example:

```java
MyDate d = new MyDate(9,10,2004);
System.out.println(d);
Output should display September 10, 2004
```

5. Create a Java program simulating an assessment system with the following deliverables: The Main Class should have two arguments which will serve as the username and password for invoking the program. For the user to be able to use the program-correct user name and password should be set as an argument. On correct log-in the program will display the following menu for the user.

Student Assessment
Choose an option:
[a]. Enroll Student
[b]. Set Course Schedule and Teachers
[c]. Check Student Account
[d]. Exit

Press the letter of your choice

The program should be able to add a student into the list of student in a certain course. It should be able to allocate 40 students only in each course. The program can also list down all the subjects and their schedules with their corresponding teachers. Given the Student ID , the program can display all the subjects enrolled by the student and will calculate the total amount the student would have to pay given the formula TotalUnits * 165. The program will iterate until the exit key is pressed. The program should implement the following classes with their methods. Main Class Student Class /should inherit from the Person Class/

```java
setId();
getSubjects();
geCourse();
getTeacher();
geTotalUnits();
geSchoolYear();
setTotalUnits();
setSchoolYear();
```

Person Class
```java
getName();
getAddress();
geAge();
setName();
setAddress();
```
setAge();
Chapter 10: Creating your own classes

1. Your task is to create a class named **Time**. The following table describes the information that an object of the Time class has:

<table>
<thead>
<tr>
<th>Instance variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour</td>
<td>Time in hours, type int</td>
</tr>
<tr>
<td>Minutes</td>
<td>Time in minutes, type int</td>
</tr>
<tr>
<td>Seconds</td>
<td>Time in seconds, type int</td>
</tr>
</tbody>
</table>

For the methods, create the following:

a. Constructor method/s that will initialize the instance variables.
b. Mutator and accessor methods for each instance variable.
c. A `setTime` method that allows the user to set the time in hours, minutes and seconds.
d. A `displayTime` method that displays the time format in 24-hour format using the format string `00:00:00`
e. An `incrementTime` method that increments the time by one second. Minutes wrap when seconds is greater than 59. Hours wrap when minutes is greater than 23.

Create a class that declares objects of class `Time` and tests the method you have defined.

2. Your task is to extend the Time class that was implemented in the previous question. Create a class named **NewTime**. Add attributes and override existing methods in the superclass that will set the time in a 12-hour format and displays whether the time is in a.m. or p.m. (e.g. 11:30:00 am). To implement polymorphism, overload the constructors such that it will accept different number of arguments (hours, minutes, seconds, am/pm).

3. Your task is to create a TaxiMeter class entry. The following table describes the information that the taxi meter has:

<table>
<thead>
<tr>
<th>Attribute/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag Down</td>
<td>The initial amount</td>
</tr>
<tr>
<td>Increment Rate</td>
<td>The amount to increment prior to the previous amount</td>
</tr>
<tr>
<td>Distance to Increment</td>
<td>The distance in meter to increment the amount</td>
</tr>
<tr>
<td>Waiting Minute to Increment</td>
<td>The time in minute to increment the amount during waiting</td>
</tr>
<tr>
<td>Distance Travelled</td>
<td>The distance traveled by the Taxi</td>
</tr>
</tbody>
</table>

For the methods, create the following:

a. Provide the necessary accessor and mutator on each of the properties.
b. Constructor/s

4. In this exercise, you are going to create a more specialized Taxi Meter that contains additional information for *non-air* and *aircon Taxies*. Your task is to extend the TaxiMeter class that was implemented in the previous question. Add some attributes and methods that you think are needed for a *non-air* and *aircon Taxi* records. Try to override some existing methods in the superclass TaxiMeter, if you really need to. The following describe the billing for *non-air* and *aircon Taxi*.

**NON-AIR**

<table>
<thead>
<tr>
<th>Flag Down</th>
<th>P 26.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>For every 400 meters</td>
<td>P 1.00</td>
</tr>
</tbody>
</table>
For every 1.5 minutes waiting  P 1.00

**AIRCON**

Flag Down  P 30.00
For every 400 meters  P 1.50
For every 1.0 minutes waiting  P 1.50

Try to modify the TaxeMeter class to create `getAmount()` abstract method. This is a method for calculating the total amount to be paid. Below are the formulas:

\[
\text{totalAmount} = \text{amountOnWait} + \text{amountOnDistanceTravelled}
\]

\[
\text{amountOnWait} = \text{increments if waitingMinuteToIncrement is satisfied}
\]

\[
\text{amountOnDistanceTravelled} = \text{incrementRate} \times \left( \frac{\text{distanceTravelled}}{\text{incrementableDistance}} \right)
\]

Write two of its subclasses, the non-air and aircon. You can add additional methods to its subclass if you want to.
Chapter 11: Inheritance, Polymorphism and Interfaces

1. Create an interface named **Comparison** whose method can be used to compare two Time objects. The methods will include **isGreater**, **isLess**, and **isEqual**.

2. Create an interface class name **Dice** with one method (public void roll()). Create a class to implement the Dice class and call it **PairOfDice**. The class has two instance variables name dice1 and dice2. Create a method that would determine the value of the dice. This would illustrate how to implement an interface.
Chapter 12: Basic Exception Handling

1. Arabic to Roman Numerals Converter -> Roman Numerals to Arabic Converter

   Ex. 1234 = MCCXXXIV
   MCCXXXIV = 1234
   Notes:
   a. Valid values must range from 1 to 3999
   b. Invalid values (numbers out of the given range above) must be trapped
   c. Invalid non-numeric values (ex. 123D) should be trapped
   d. Invalid numeric and non-numeric values should have specific ERROR MESSAGES

2. Arabic Numbers to Words Converter

   Ex. 1234 = One Thousand Two Hundred Thirty-Four
   Notes:
   a. Valid values must range from 1 to 3999
   b. Invalid values (numbers out of the given range above) must be trapped
   c. Invalid non-numeric values (ex. 123D) should be trapped
   d. Invalid numeric and non-numeric values should have specific ERROR MESSAGES

3. Create a class FindArray that will try to iterate in the array displaying the content in each index. The program should be able to catch an ArrayIndexOutOfBoundsException, a NullPointerException and a RunTimeException.

4. Create a program that will ask the user a string input and will then try return the string in backward order, the program should be able to catch a StringIndexOutOfBoundsException.

5. Implement a class called Birthday that extends the MyDate class. Include 2 constructors, one with month, day and year parameters and another one with only month and day as parameters. The second constructor will call the parent constructor passing the default value 1970 for the year. Implement an inner class InvalidDateException that extends the Exception class. The message in the exception should read “Invalid Date!” Include a method called String getZodiacSign() that throws InvalidDate Exception that returns the zodiac sign of the person if it valid and throws an exception if not valid.

Use the table below:

<table>
<thead>
<tr>
<th>Zodiac</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aquarius</td>
<td>Jan 20 – Feb 18</td>
</tr>
<tr>
<td>Pisces</td>
<td>Feb 19 – Mar 20</td>
</tr>
<tr>
<td>Aries</td>
<td>Mar 21 – Apr 19</td>
</tr>
<tr>
<td>Taurus</td>
<td>Apr 20 – May 20</td>
</tr>
<tr>
<td>Gemini</td>
<td>May 21 – Jun 20</td>
</tr>
<tr>
<td>Cancer</td>
<td>Jun 21 – Jul 22</td>
</tr>
<tr>
<td>Leo</td>
<td>Jul 23 – Aug 22</td>
</tr>
<tr>
<td>Virgo</td>
<td>Aug 23 – Sep 22</td>
</tr>
<tr>
<td>Libra</td>
<td>Sep 23 – Oct 22</td>
</tr>
<tr>
<td>Scorpio</td>
<td>Oct 23 – Nov 21</td>
</tr>
<tr>
<td>Sagittarius</td>
<td>Nov 22 – Dec 21</td>
</tr>
<tr>
<td>Capricorn</td>
<td>Dec 22 – Jan 19</td>
</tr>
</tbody>
</table>
References

29. Java Language Keywords. Available at
http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.html