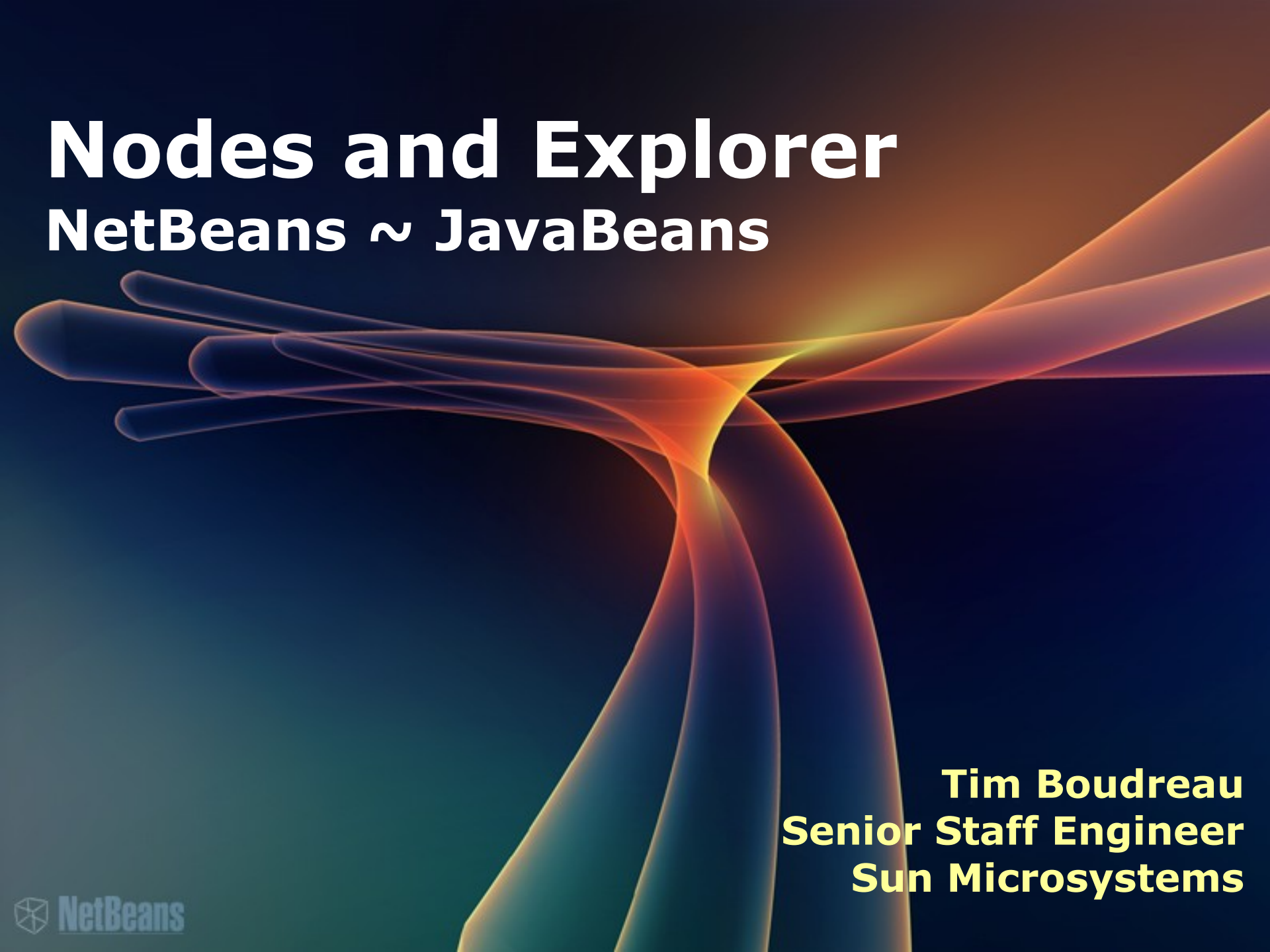
Nodes and Explorer

NetBeans ~ JavaBeans



Tim Boudreau
Senior Staff Engineer
Sun Microsystems

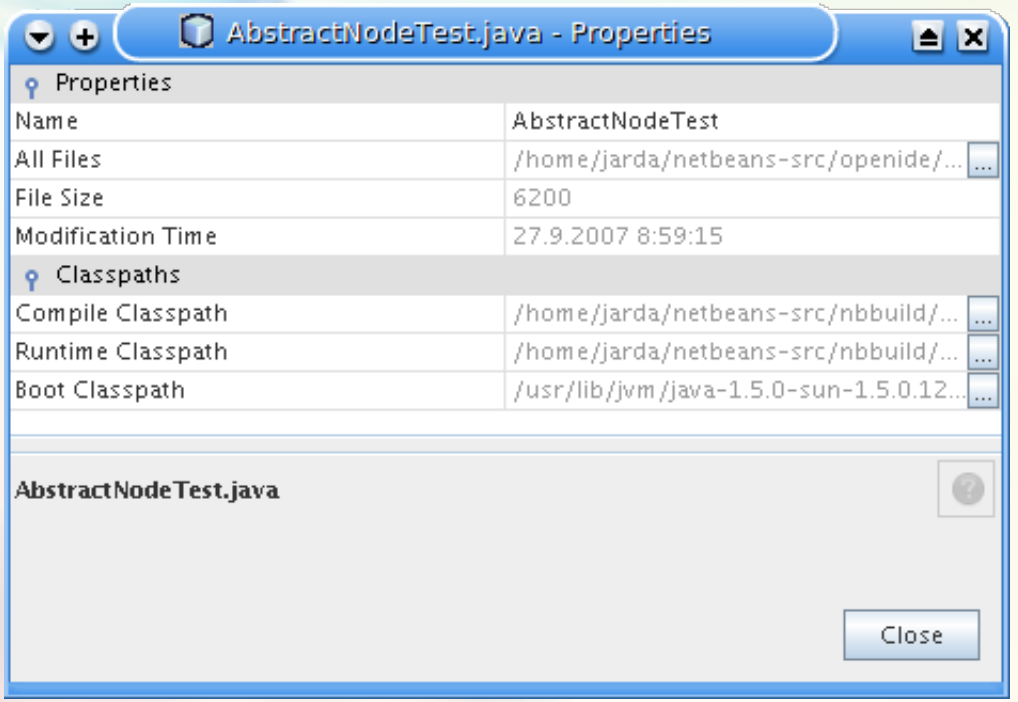
NetBeans

Agenda

- Why NetBeans is called NetBeans?
- Nodes
- Node views
- Composition
- Q/A

Why is it called NetBeans?

- JavaBeans for the network
- Beans everywhere 1.0
  - > bean context
  - > property sheet
- Problems
  - > API vs. SPI



API and SPI

- API – Application Program Interface
  - You are given an object you can call
  - Object should usually be a *final* class
    - Otherwise you cannot change it backward-compatibly
    - You can still use Lookup in an API to make it extensible, even though it is final
- SPI – Service Provider Interface
  - You provide an implementation of some interface
  - It adds new functionality to an existing library
    - Which may have an API

Separating API and SPI

- Backward compatibility
  - A social responsibility when creating an API
- Final classes for APIs
- Interfaces/Abstract classes for SPI
- You can compatibly add methods to a final class
- You can compatibly remove methods from an interface
  - *If* you ensure API clients can never get an actual instance of the SPI class – wrap them in a final class
- If you mix API and SPI, no changes are provably backward-compatible

Nodes

- Typed JavaBeans
  - no reflection
  - standard listeners
  - extensibility
- Support for hierarchy
  - correctness guaranteed
- Bridge to beans via BeanNode

Presentation Layer

- Nodes are a *presentation layer*
- Nodes are hierarchical
  - They have child nodes that can have child nodes
- Nodes take a random object and provide human-friendly features
  - Actions
  - Display name
  - Description
  - Icon
  - Properties (can be shown/edited in property sheet)
  - Clipboard operations

Nodes API

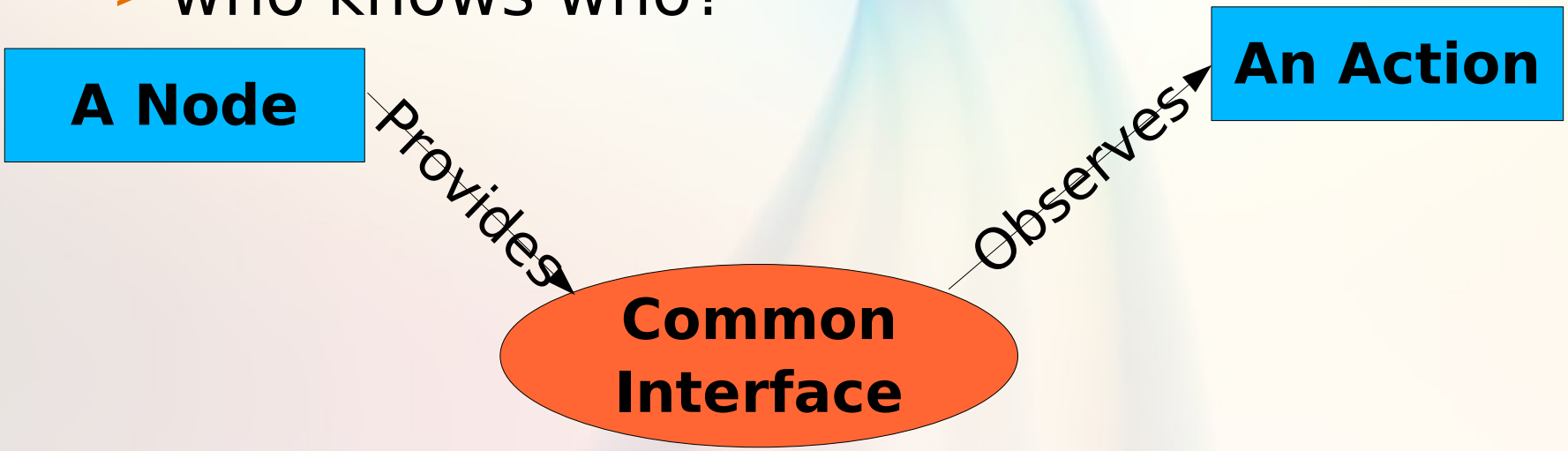
```
import org.openide.nodes.AbstractNode;
import org.openide.nodes.Children;
class MyNode extends AbstractNode {
  public MyNode() {
   super(new MyChildren());
}}
class MyChildren extends Children.Keys<String> {
  protected void addNotify() {
    setKeys(Collections.nCopies(1, “Child”));
  }
  protected Node[] createNodes(String key) {
    MyNode n = new MyNode();
    n.setName(key);
    return new Node[] { n };
}}
```

Rules

- Nodes are the *presentation layer – a model of data*
  - they are not the data
- Child nodes are created lazily
  - ChildFactory + Children.create()
    - Handles creating children on a background thread
  - Children.addNotify, Children.setKeys
- Make sure they garbage collect
  - leaks with listeners
  - possible use of removeNotify() in Children subclasses
- Never cast a Node to a particular type

Node Actions

- Addition over JavaBeans
- Swing Actions
  - Action[] Node.getAction(boolean)
- Multiselection
  - who knows who?



A Node's Context

- Lookup Node.getLookup()
  - Passed in constructor
  - Replacement for old getCookie(Class)
    - No marker interface
- OpenCookie, EditorCookie, etc.
  - Put an implementation of some class in the Nodes lookup
  - Write actions sensitive to that object
- Multiselection
  - ProxyLookup

Context Actions

- http://wiki.netbeans.org/wiki/view/DevFaqActionContex

```
public class FooAction extends AbstractAction implements LookupListener, ContextAwareAction
{
    private Lookup context;
    Lookup.Result lkpInfo;
    public FooAction() {
        this(Utilities.actionsGlobalContext());
    }
    private FooAction(Lookup context) {
        this.context = context;
    }
    void init() {
        lkpInfo = context.lookupResult (Whatever.class);
        lkpInfo.addLookupListener(this); resultChanged(null);
    }
    public boolean isEnabled() {
        init();
        return super.isEnabled();
    }
    public Action createContextAwareInstance(Lookup context) {
        return new FooAction(context);
    }
}
```

Explorer Views

- An “explorer” component is a Swing component
- It can show a Node and its children
- Many different components
  - Trees, Lists, Combo Boxes, Tree Tables, Property Sheet
  - all in org.openide.explorer.view
- Nodes provide a universal tree-model for presenting data
- Explorer views are components to show that data to the user

Root Container

```
class MyPanel extends JPanel implements
  ExplorerManager.Provider {


public MyPanel() {
  myManager = new ExplorerManager();
  add(new BeanTreeView());
  add(new PropertySheetView());
  myManager.setRootContext(myNode);
}
public ExplorerManager getExplorerManager() {
   return myManager;
}
```

Views

- ExplorerManager
  - root context
  - explored context
  - selected nodes (vetoable)
- General Model behind Swing
  - BeanTreeView, ContextTreeView
  - ListView
  - PropertySheet
  - TableTreeView

Write your own View

- Just a visual JavaBean
- Overwrite addNotify and removeNotify
  - search parents for ExplorerManager.Provider
  - add listeners
  - display what ExplorerManager says
- Control ExplorerManager
  - call setters
  - add vetoable listeners

Nodes and Selection

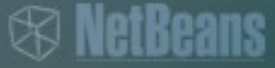
- Each window component has a Lookup
- Nodes have Lookups
- You can easily have the window component's Lookup proxy whatever the Lookup(s) of the selected Node(s) in an explorer view

Conclusion

- Nodes are typed JavaBeans
- Hierarchy
- Extensible
- Rich Set of Views
- Standalone

DEMO

- Standalone Explorer



Q&A

- http://bits.netbeans.org/dev/javadoc/org-openide-nodes/
- http://bits.netbeans.org/dev/javadoc/org-openide-explorer/
-