Nodes & Explorer view
Problem statement

- When presenting data models to user in “plain” Swing
  - Need to program many low level details (e.g., cell renderers)
  - Switching from one control (e.g., JList) to another control (JTree) is hard
  - Context sensitivity
- A lot of plumbing code is needed!
Solution

• Nodes API:
  > Easy to create a tree-like model
• Explorer API:
  > Many UI components that can render the model
What is a node?

- Nodes hierarchical
  - child nodes
    - with own child nodes
- Nodes ~ a presentation layer
  - Actions
  - Display name
  - Description
  - Icon
  - Properties (can be shown/edited in property sheet)
  - Clipboard operations
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() {
        String[] names = {"Tom", "Dick", "Harry "};
        setKeys(names);
    }

    protected Node[] createNodes(String key) {
        MyNode n = new MyNode();
        n.setName(key);
        return new Node[] { n };
    }
}
Rules on nodes

• Nodes are the model
  > not the data
• Create them lazily
  > Children.addNotify, Children.setKeys
• Make sure they garbage collect
  > leaks with listeners
  > possible use of removeNotify
Typical usage of Children.Keys

1. Subclass
2. Decide what type your key should be.
3. Implement `createNodes(T)` to create some nodes per key
   - usually exactly one
4. Override `addNotify()` to compute a set of keys and set it using `setKeys(Collection)`
5. Override `removeNotify()` to just call `setKeys` on `Collections.EMPTY_SET`
6. When your model changes, call `setKeys` with the new set of keys
   - `Children.Keys` will be smart and calculate exactly what it needs to do efficiently
class MyNode extends AbstractNode {
    public MyNode() {
        super(new Children.create(new MyChildFactory(),true));
    }
}
class MyChildFactory extends ChildFactory<String> {
    @Override
    protected boolean createKeys(List toPopulate) {
        ...
    }
    @Override
    protected Node createNodeForKey(String key) {
        ...
    }
}
What is an Explorer View?

• “explorer” component
  > Swing component
• shows a Node and its children
• Many different components
  > Trees, Lists, Combo Boxes, Tree Tables, Property Sheet
  > all in org.openide.explorer.view

• Nodes
  > a universal tree-model for presenting data
• Explorer views
  > components to show that data to the user
class MyPanel extends JPanel

    implement ExplorerManager.Provider {

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

    public ExplorerManager getExplorerManager() {
        return myManager;
    }

}
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
Editor code generation
Code generator

- Generates code snippet
- Editor → Insert code (Alt+Insert)
- Tied to document type
- Created by code generator wizard
Code generator

- Implements org.netbeans.spi.editor.codegen.CodeGenerator
- public void invoke()
  > invoked when user chooses the generator from the Insert Code dialog