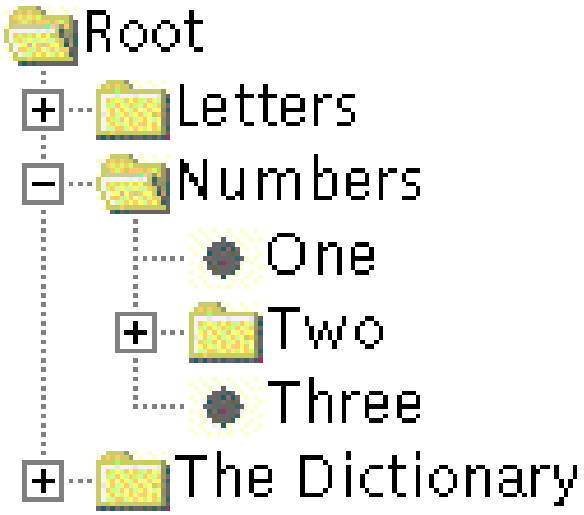
How to Use Trees

The practice of JTree in swing

The JTree component

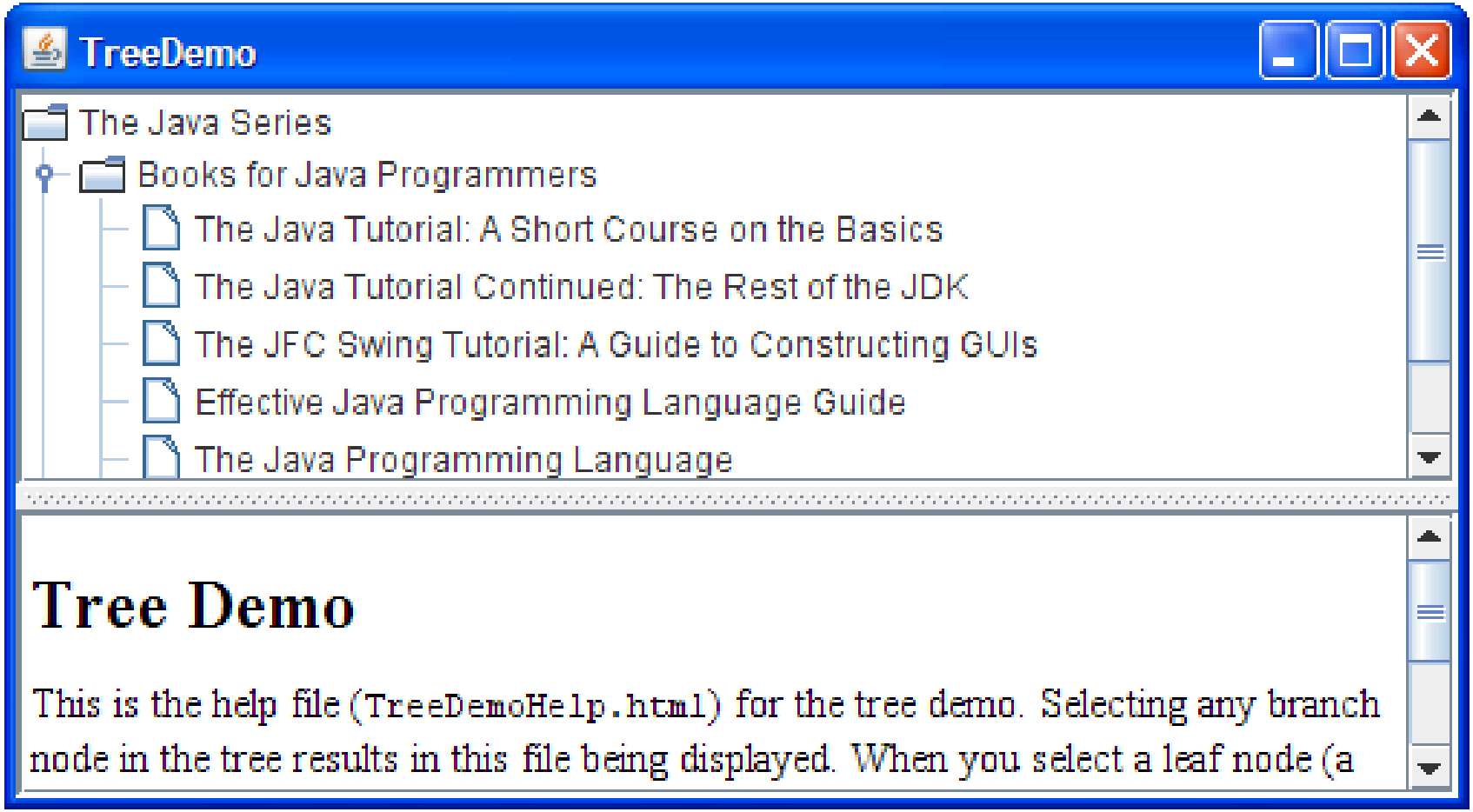
- **Abstract widget** so it does not hold the data
  - Tree gets data by querying the Model
- JTree displays its data vertically.
  - Each row displayed by the tree contains exactly one item of data, which is called a ***node***.
  - Every tree has a ***root* node** from which all nodes descend
  - A node can either have children or not.
    - ***branch* nodes** are those that can have children
    - ***leaf* nodes** are those that can not have children



JTree Structure

- The user can expand and collapse branch nodes — making their children visible or invisible — by clicking them.
  - By default, all branch nodes except the root node start out collapsed.
- A program can detect changes in branch nodes' expansion state by listening for tree expansion or tree-will-expand events and responding to them.
- A specific node in a tree can be identified either by a **TreePath**, an object that encapsulates a node and all of its ancestors, or by its display row
  - An expanded node is a non-leaf node, that will display its children when all its ancestors are expanded.
  - A collapsed node is one which hides them.
  - A hidden node is one which is under a collapsed ancestor.

Creating a Tree



TreeDemo.java

DefaultMutableTreeNode serves as the root node for the tree

```
private JTree tree;
...
public TreeDemo() {
    ...
    DefaultMutableTreeNode top =
        new DefaultMutableTreeNode("The Java Series");
    createNodes(top); // create tree to link to model

    tree = new JTree(top); //build JTree around it
    ...
    JScrollPane treeView = new JScrollPane(tree);
    ...
}
```

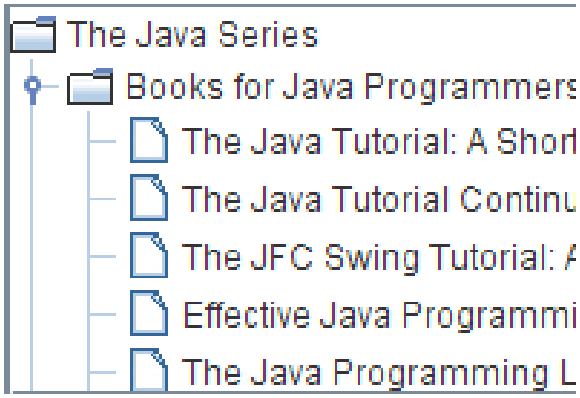
puts the tree in a scroll pane, a common tactic because showing the full, expanded tree would otherwise require too much space.

DefaultMutableTreeNode

- A DefaultMutableTreeNode is a general-purpose node in a tree data structure.
- A DefaultMutableTreeNode may also hold a reference to a user object, the use of which is left to the user.
- Asking a DefaultMutableTreeNode for its string representation with toString() returns the string representation of its user object.
- Class provides enumerations for efficiently traversing a tree or subtree in various orders or for following the path between two nodes.

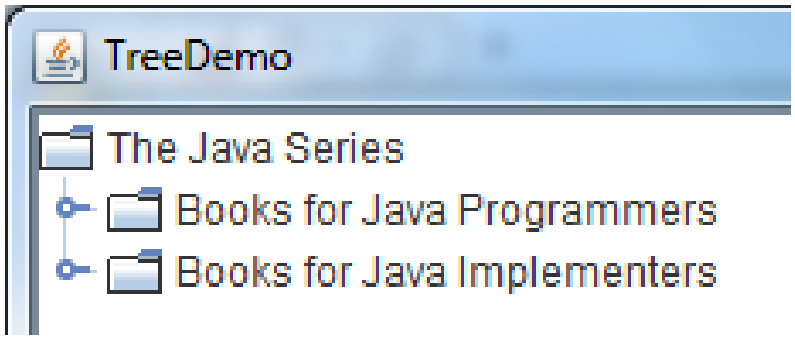
Building a Tree

```
private void createNodes(DefaultMutableTreeNode top) {
    DefaultMutableTreeNode category = null;
    DefaultMutableTreeNode book = null;

    category = new DefaultMutableTreeNode("Books for Java Programmers");
    top.add(category);

    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Tutorial: A Short Course on the Basics",
        "tutorial.html"));      //original Tutorial
    category.add(book);

    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Tutorial Continued: The Rest of the JDK",
        "tutorialcont.html")); //Tutorial Continued
    category.add(book);

    book = new DefaultMutableTreeNode(new BookInfo
        ("The JFC Swing Tutorial: A Guide to Constructing GUIs",
        "swingtutorial.html"));
    category.add(book); //JFC Swing Tutorial
```



Building a Tree

```
//...add more books for programmers...

    category = new DefaultMutableTreeNode("Books for Java Implementers");
    top.add(category);       // create second branch from root

    //VM
    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Virtual Machine Specification",
         "vm.html"));
    category.add(book);

    //Language Spec
    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Language Specification",
         "jls.html"));
    category.add(book);
}
```



Building a Tree

- The argument to the DefaultMutableTreeNode constructor is the *user object* which is an object that contains or points to the data associated with the tree node.
- The user object can be a string, or it can be a custom object.
- If you implement a custom object, you should implement its toString method so that it returns the string to be displayed for that node.
  - JTree, by default, renders each node using the value returned from toString, so it is important that toString returns something meaningful.
  - Sometimes, it is not feasible to override toString; in such a scenario you can override the convertValueToText of JTree to map the object from the model into a string that gets displayed.
- For example, the BookInfo class used in the previous code snippet is a custom class that holds two pieces of data: the name of a book, and the URL for an HTML file describing the book. The toString method is implemented to return the book name. Thus, each node associated with a BookInfo object displays a book name.

Node Selection

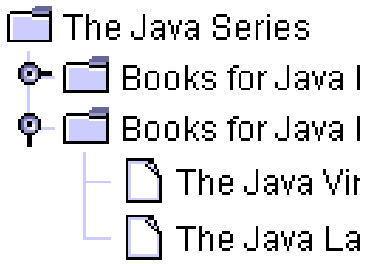
- To respond to tree node selections JTree supports a tree selection listener that you create and register on the JTree.
  - i.e. the JTree listens for user input events, as usual
- The TreeSelectionModel interface defines three values for the selection mode:
  - **SINGLE_TREE_SELECTION**
    - This is the mode used in the demo example. At most one node can be selected at a time
  - **DISCONTIGUOUS_TREE_SELECTION**
    - This is the default mode for the default tree selection model. With this mode, any combination of nodes can be selected.
  - **CONTIGUOUS_TREE_SELECTION**
    - With this mode, only nodes in adjoining rows can be selected

```
    //Where the JTree is initialized:
    tree.getSelectionModel().setSelectionMode
            (TreeSelectionModel.SINGLE_TREE_SELECTION);

    //Listen for when the selection changes.
    tree.addTreeSelectionListener(this);
.
.
public void valueChanged(TreeSelectionEvent e) {
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                       tree.getLastSelectedPathComponent();
    //Returns the last path element of the selection.
    //This method is useful only when the selection model allows a single selection.

    if (node == null)    //Nothing is selected.
    return;

    Object nodeInfo = node.getUserObject();
    if (node.isLeaf()) {
        BookInfo book = (BookInfo)nodeInfo;
        displayURL(book.bookURL);
    } else {
        displayURL(helpURL);
    }
}
```

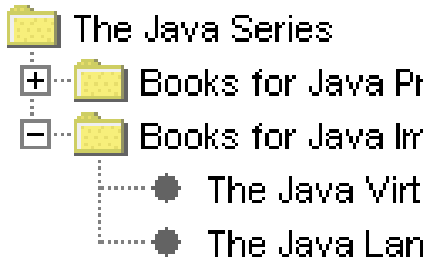
DefaultMutableTreeNode provides methods to access both its own properties and the user object and nodes properties

Customising Display

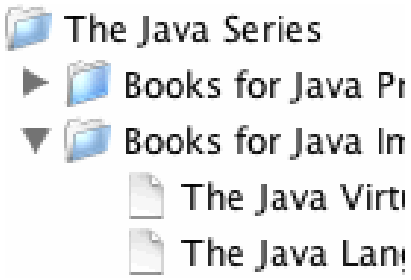
- tree.setRootVisible(true)
  - to show the root node or tree.setRootVisible(false) to hide it.
- tree.setShowsRootHandles(true)
  - request that a tree's top-level nodes — the root node (if it is visible) or its children (if not) — have handles that let them be expanded or collapsed.
- tree.putClientProperty("JTree.lineStyle",);
  - Varies with look & feel: Remember
  - E.g. “Angled”, “Horizontal”, “None” in Java L&F
  - Defines the lines that group items together



Java look & feel



Windows look & feel

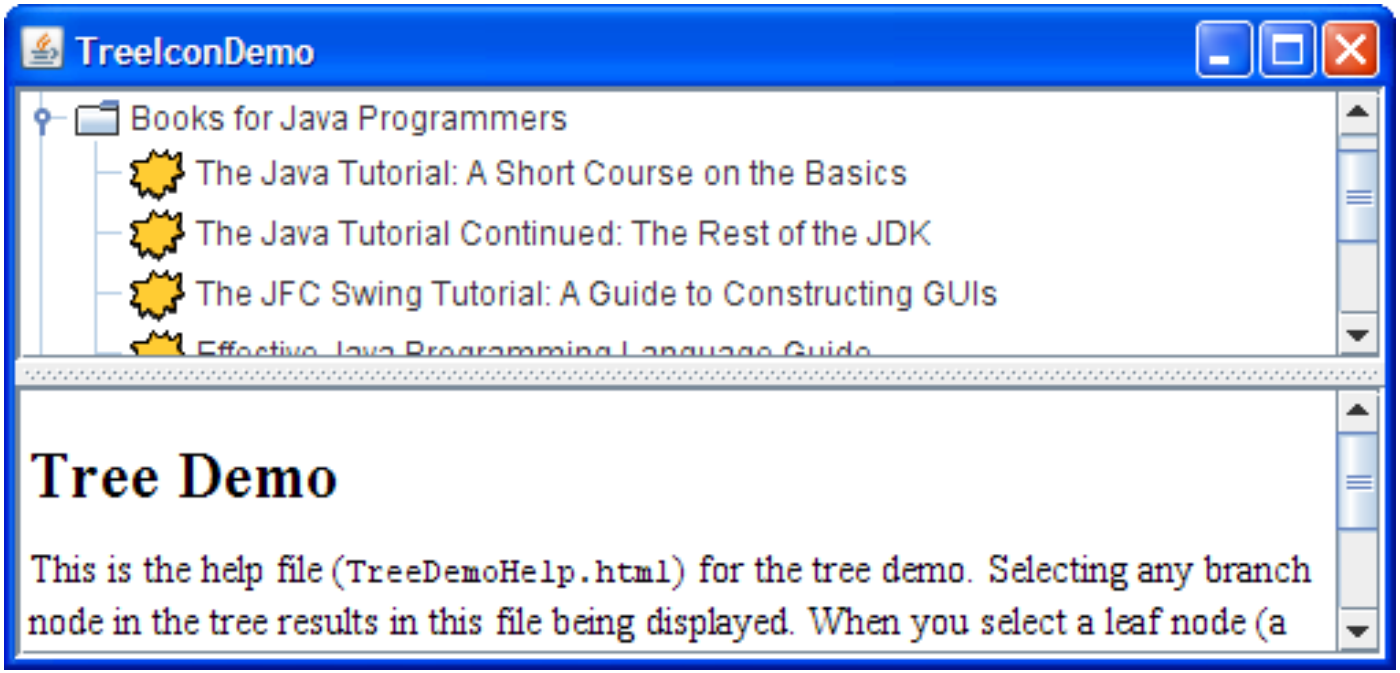


Mac look & feel

More Customisation

- To change the default icon used for leaf, expanded branch, or collapsed branch nodes:
  - create an instance of DefaultTreeCellRenderer.
  - specify the icons to use
    - setLeafIcon (for leaf nodes),
    - setOpenIcon (for expanded branch nodes),
    - setClosedIcon (for collapsed branch nodes). .
  - use JTree's setCellRenderer method to specify that the DefaultTreeCellRenderer paint its nodes.

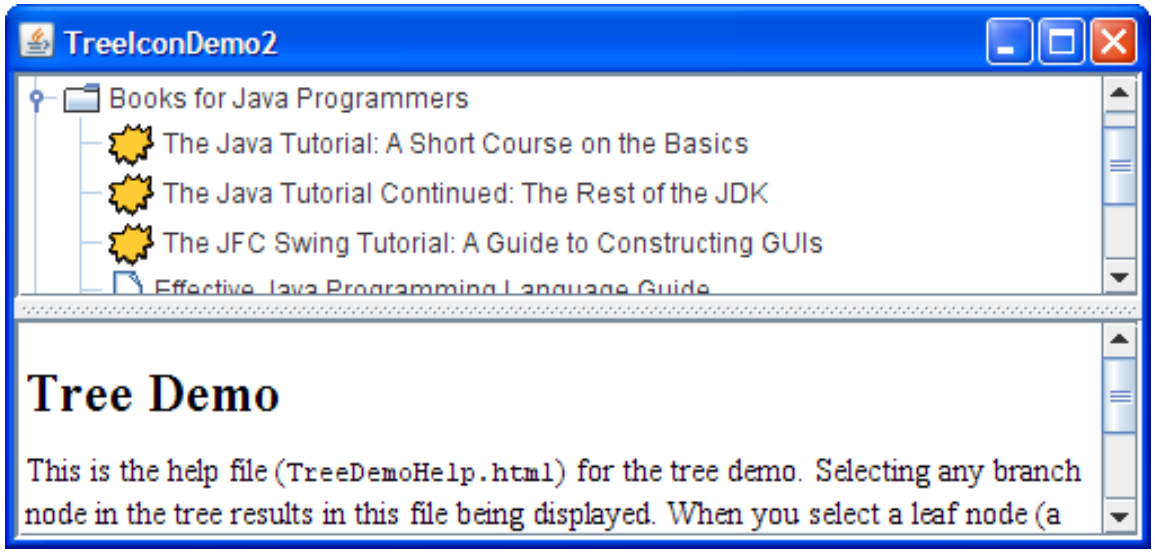
```
ImageIcon leafIcon = createImageIcon("images/middle.gif");
if (leafIcon != null) {
    DefaultTreeCellRenderer renderer =
            new DefaultTreeCellRenderer();
    renderer.setLeafIcon(leafIcon);
    tree.setCellRenderer(renderer); // tree is our JTree
}
```



TreeIconDemo.java

Still More Customisation

- TreeIconDemo2.java creates a cell renderer that extends DefaultTreeCellRenderer and
  - varies the leaf icon depending on whether the word "Tutorial" is in the node's text data
  - enables tool-tips and specifies tool-tip text

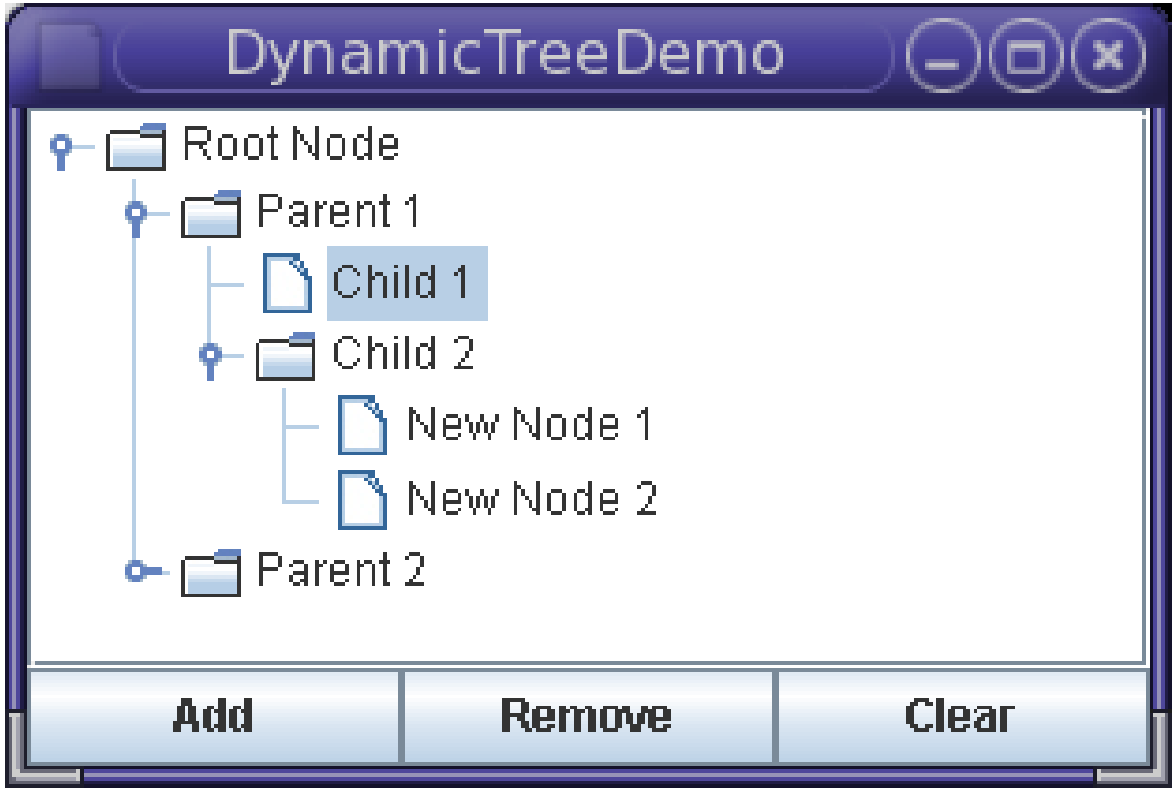


JTree & Tree Models

- We now have a tree structure created from **DefaultMutableNodes** embedded in a **JTree**
  - DefaultMutableNodes provide a tree that can link to external data
  - JTree properties & methods customise appearance
  - TreeSelectionListener picks up selection events
- The external trees JTree needs to interface to can have a wide variety of access functions/restrictions
- The model (in MVC sense) becomes a set of methods built on top of the tree structure that define what the JTree can do to/with the tree structure

- The **TreeModel** interface

Dynamically Changing a Tree



DynamicTreeDemo.java

JTree & Tree Models

```
rootNode = new DefaultMutableTreeNode("Root Node");
treeModel = new DefaultTreeModel(rootNode);
treeModel.addTreeModelListener(new MyTreeModelListener());

tree = new JTree(treeModel);
tree.setEditable(true);
tree.getSelectionModel().setSelectionMode
        (TreeSelectionModel.SINGLE_TREE_SELECTION);
tree.setShowsRootHandles(true);
```

Embed tree in a tree model, which is embedded in a JTree

By explicitly creating the tree's model, the code guarantees that the tree's model is an instance of **DefaultTreeModel**.

- can invoke the model's **insertNodeInto** method,
- **setEditable(true)** makes the text in the tree's nodes editable

Swing allows construction of Custom Tree models, but must all implement Swing's **TreeModel** interface (more later)

Using the Tree Model: Dynamically Changing a Tree

- When the user has finished editing a node, the model generates a tree model event that tells any listeners that tree nodes have changed. THIS INCLUDES JTREE
- To be notified of node changes, we can implement a TreeModelListener on the JTree. This receives events from the tree model.

TreeModelListener

```
class MyTreeModelListener implements TreeModelListener {

    public void treeNodesChanged(TreeModelEvent e) {
        DefaultMutableTreeNode node;
        node = (DefaultMutableTreeNode)
                (e.getTreePath().getLastPathComponent());// get selected node

        // If the event lists children, then by convention the changed node is the
        // child of the node we already have. Otherwise, the changed node and
        // the specified node are the same.
        try {
            int index = e.getChildIndices()[0];
            node = (DefaultMutableTreeNode)
                   (node.getChildAt(index));
        } catch (NullPointerException exc) {}

        System.out.println("The user has finished editing the node.");
        System.out.println("New value: " + node.getUserObject());
    }
    public void treeNodesInserted(TreeModelEvent e) { }
    public void treeNodesRemoved(TreeModelEvent e) { }
    public void treeStructureChanged(TreeModelEvent e) {}
}
```

Using the Tree Model: Adding a Node

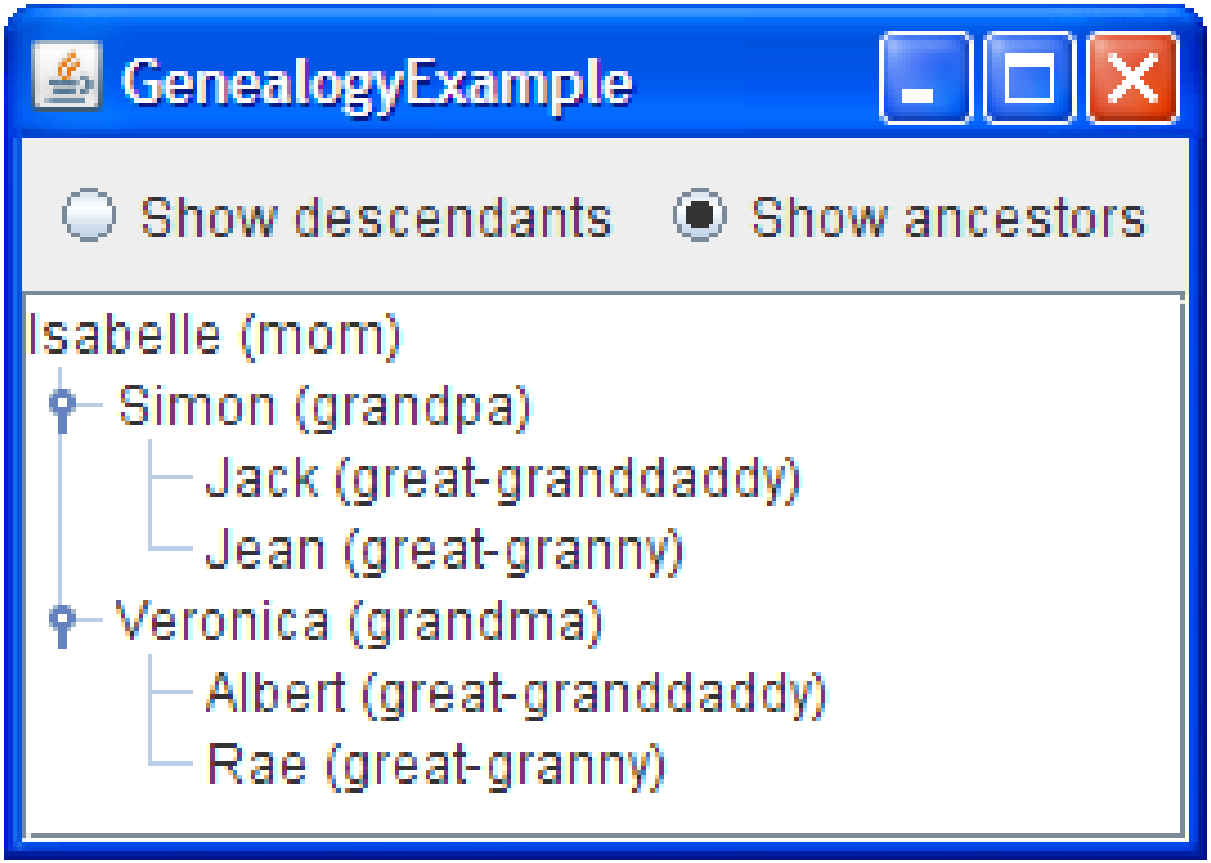
- Code with the add button adds a node to the tree
  - creates a node,
  - inserts it into the tree model,
  - if appropriate, requests that the nodes above it be expanded and the tree scrolled so that the new node is visible.
- To insert the node into the model, the code uses the insertNodeInto method provided by the DefaultTreeModel class.

```
treePanel.addObject("New Node " + newNodeSuffix++);// in action
                                // listener, called when add button pressed
...
public DefaultMutableTreeNode addObject(Object child) {// new child is parameter
    DefaultMutableTreeNode parentNode = null;
    TreePath parentPath = tree.getSelectionPath(); // parent is last node selected

    if (parentPath == null) { //There is no selection. Default to the root node.
        parentNode = rootNode;
    } else {
        parentNode = (DefaultMutableTreeNode)
                (parentPath.getLastPathComponent());
    }
    return addObject(parentNode, child, true); //now add new child to parent
}
...
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent, Object
child, boolean shouldBeVisible) {
    DefaultMutableTreeNode childNode =
            new DefaultMutableTreeNode(child); // create new node
    ...
    treeModel.insertNodeInto(childNode, parent,
                        parent.getChildCount()); // and insert it into the tree model

    if (shouldBeVisible) {//Make sure the user can see the lovely new node.
        tree.scrollPathToVisible(new TreePath(childNode.getPath()));
    }
    return childNode;
```

Creating a Tree Model

- **DefaultTreeModel** is a convenience class for this abstract data class that provides the role of default model
- You often need to write a custom data model. By implementing the TreeModel interface.
- **TreeModel** specifies methods for
  - getting a particular node of the tree,
  - getting the number of children of a particular node,
  - determining whether a node is a leaf,
  - notifying the model of a change in the tree,
  - adding and removing tree model listeners.
- TreeModel interface accepts any kind of object as a tree node.
  - It does not require that nodes be represented by DefaultMutableTreeNode objects, or even that nodes implement the Treenode interface.
- If you have a pre-existing hierarchical data structure, You just need to implement your tree model so that it uses the information in the existing data structure.

Genealogy


GenealogyExample
Show descendants
Show ancestors
Isabelle (mom)
Simon (grandpa)
Jack (great-granddaddy)
Jean (great-granny)
Veronica (grandma)
Albert (great-granddaddy)
Rae (great-granny)

Genealogy

- The model implements the TreeModel interface directly. This requires implementing methods for getting information about nodes, such as which is the root and what are the children of a particular node.
- In GenealogyModel, each node is represented by an object of type Person, a custom class that does not implementTreeNode.
- A tree model must also implement methods for adding and removing tree model listeners, must fire TreeModelEvents to those listeners when the tree's structure or data changes.
- When the user instructs GenealogyExample to switch from showing ancestors to showing descendants, the tree model makes the change and then fires an event to inform its listeners (such as the tree component).

Summary

- Discussed the use of JTree
- Shown how to modify the interface
- Talked about the event listeners
- Demonstated how to implement the tree model
- Shown a range of examples