Les Arbres

- Le Composant JTree
- JTree et TreeUI
- TreeModel & TreeNode
- TreeCellRenderer
- DefaultTreeNode

Les Arbres

- Le **`JTree`** est un composant affichant/editant des données sous forme hiérarchique.
- Vision d'ensemble : six autres classes/interfaces utilisées :
  - **`TreeModel`** : contient les données figurant dans l'arbre
  - **`TreeNode`** : interface des noeuds et de la structure d'arbre
  - **`TreeSelectionModel`** : model de sélection des noeuds
  - **`TreePath`** : chemin dans l'arbre (de la racine vers le sommet sélectionné par exemple)
  - **`TreeCellRenderer`** : interface de rendu d'un noeud
  - **`TreeCellEditor`** : l'éditeur pour un noeud est éditable
- Elles sont définies dans **`javax.swing.tree.*`**

JTree et TreeUI

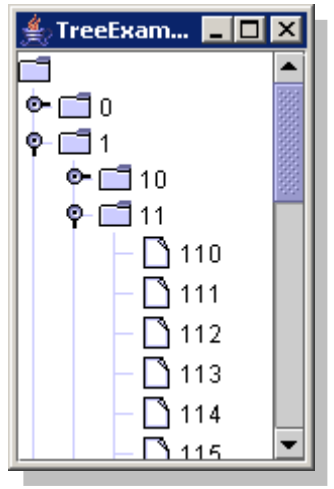
- Le **`JTree`** utilise l'interface **`TreeUI`** pour effectuer l'affichage de l'arbre. **`TreeUI`** impose pour l'affichage de représenter l'arbre comme une liste.
- Le **`TreeUI`** peux utiliser un cache associant un noeud à l'index de la ligne à afficher.
- L'association est faite par les deux méthodes :
  - **`int getRowForPath(TreePath path)`**
  - **`TreePath getPathForRow(int row)`**
- Il est possible de paramètrer ce cache dans le cas où l'arbre est gros :
  - **`is/setLargeModel(boolean)`**

JTree et taille des noeuds

- Le **`JTree`** gère différemment les noeuds si ceux-ci ont ou non la même taille.
- Si les noeuds sont de tailles différentes :
  - parcours à l'initialisation sur l'ensemble de ses données pour obtenir la taille du composant.
  - Cache : **`VariableHeightLayoutCache`**.
- Si les noeuds sont de même taille :
  - Calcul la taille du premier noeud et multiplie.
  - Cache : **`FixedHeightLayoutCache`**.
- Fixer la taille de tous les noeuds
  - **`get/setRowHeight(int pixel)`**
- Demander le calcul de la taille des noeuds (défaut)
  - **`setRowHeight(0)`**

Pliage/dépliage de l'arbre

- La classe **`JTree`** maintient pour chaque noeud de l'arbre une information indiquant si l'arbre est plié (**`expand`**) ou déplié (**`collapse`**).
- Test si replié ou déplié :
  - **`boolean isCollapsed(TreePath path)`**
  - **`boolean isExpanded(TreePath path)`**
- Replie ou déplie l'arbre :
  - **`collapsePath(TreePath path)`**
  - **`expandPath(TreePath path)`**
- Test si un noeud est visible et rend un noeud visible :
  - **`boolean isVisible(TreePath path)`**
  - **`void makeVisible(TreePath path)`**



TreePath

- Représente un chemin dans un arbre jusqu'à un noeud. Implanter par une liste chaînée d'objet **`TreePath`** du noeud vers la racine.
- Constructeurs

```
TreePath(Object noeud)
```

```
TreePath(Object[] path)
```

- Obtenir le chemin jusqu'au pére :

```
TreePath getParentPath()
```

- Créer un chemin en ajoutant un fils :

```
TreePath pathByAddingChild(Object child)
```

TreePath (2)

- Accéder au dernier noeud du chemin

```
Object getLastPathComponent()
```

- Obtenir le nombre de noeud dans le chemin : O(n)

```
int getPathCount()
```

- Obtenir le n-ième élément : O(2n)

```
Object getPathComponent(int element)
```

- Obtenir le chemin sous forme de tableau : O(2n)

```
Object[] getPath()
```

- Eviter de parcourir un **`TreePath`** comme un tableau.

Modèle d'arbre

- Un arbre est créé à partir d'un **`TreeModel`**
  - Obtenir la racine de l'arbre

```
Object getRoot()
```

  - Savoir si un noeud est une feuille

```
boolean isLeaf(Object node)
```

  - Obtenir le nombre de fils d'un noeud

```
int getChildCount(Object parent)
```

  - Obtenir le nième fils d'un noeud

```
Object getChild(Object parent, int index)
```

  - Obtenir l'index d'un fils pour un noeud

```
int getIndexOfChild(Object parent, Object
child)
```

Modèle d'arbre (suite)

- Modification de l'arbre
  - Changer la valeur d'un noeud

```
void valueForPathChanged(TreePath path, Object newValue)
```

Peut ne pas être implanter (get/setEditable())

- Gestion des évènement de données
  - Ajouter un écouteur de modification de données

```
void  addTreeModelListener(TreeModelListener l)
```

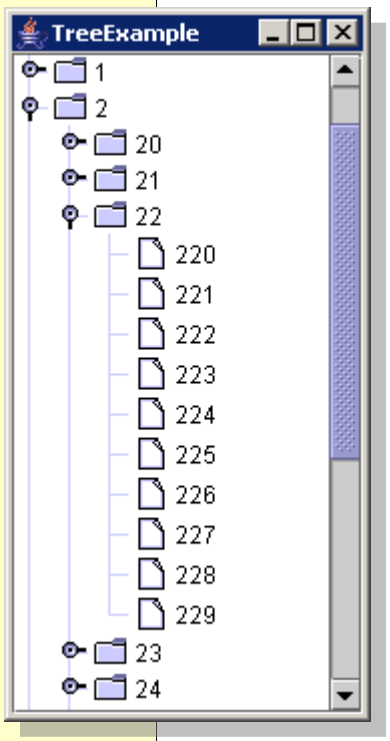
  - Retirer un écouteur

```
void  removeTreeModelListener(TreeModelListener l)
```

Exemple simple de modèle d'arbre

- Affiche sous forme d'arbre les valeurs de 0 à 999.

```
class SimpleTreeModel implements TreeModel {
  public String getRoot() {
    return "";
  }
  public int getChildCount(Object parent) {
    return (isLeaf(parent))?0:10;
  }
  public boolean isLeaf(Object node) {
    return node.toString().length()>2;
  }
  public String getChild(Object parent, int index) {
    return parent.toString()+index;
  }
  public int getIndexOfChild(Object parent, Object child) {
    String text=child.toString();
    char lastCharacter=text.charAt(text.length()-1);
    return lastCharacter-'0';
  }
  public void valueForPathChanged(TreePath path, Object newValue) {
    throw new UnsupportedOperationException();
  }
  public void addTreeModelListener(TreeModelListener l) {}
  public void removeTreeModelListener(TreeModelListener l) {}
}
```



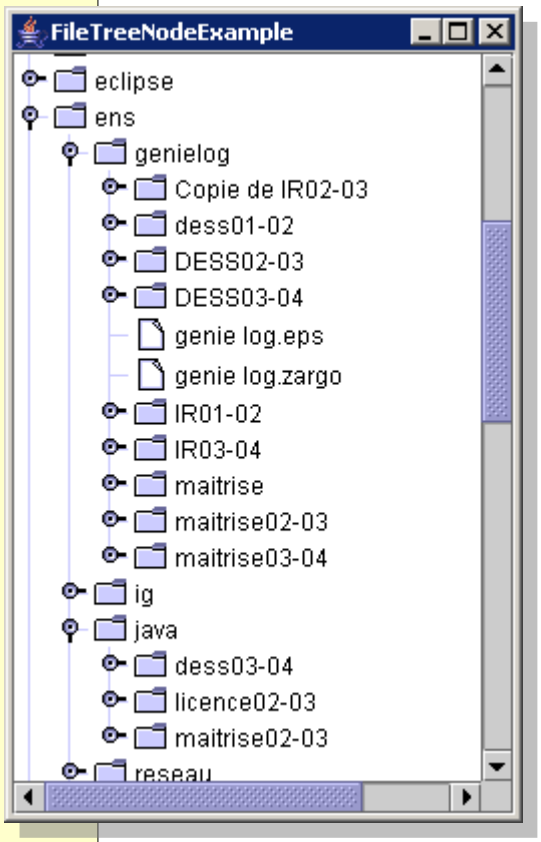
Le modèle d'arbre par défaut

- La classe **`AbstractTreeModel`** n'existe pas.
- La classe **`DefaultTreeModel`** implante le modèle d'arbre par défaut :
  - Les noeuds de l'arbre doivent alors implanter l'interface **`TreeNode`.**
  - Stocke uniquement la racine de l'arbre.
- Deux interfaces suivant si le noeud est editable :
  - **`TreeNode`**
  - **`MutableTreeNode`** (hérite de **`TreeNode`**)
- La classe **`DefaultMutableTreeNode`** est l'implantation par défaut de l'interface **`MutableTreeNode`** (utilise un **`Vector`** <= toujours MAL)

Arbre avec des TreeNodes

- Affiche l'arbre des fichier du disque.

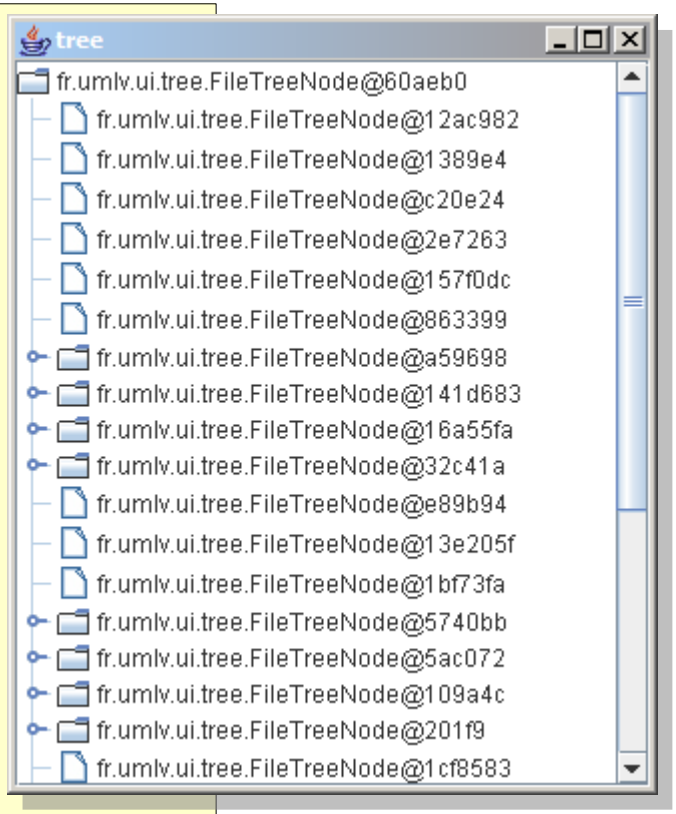
```
class FileTreeNode implements TreeNode {
  public FileTreeNode(FileTreeNode parent,File file) {
    this.parent=parent;
    this.file=file;
  }
  public int getChildCount() {
    return processChildren().size();
  }
  public boolean getAllowsChildren() {
    return true;
  }
  public boolean isLeaf() {
    return !file.isDirectory();
  }
  public Enumeration<?> children() {
    return Collections.enumeration(processChildren());
  }
  public TreeNode getParent() {
    return parent;
  }
  public TreeNode getChildAt(int childIndex) {
    return processChildren().get(childIndex);
  }
  public int getIndex(TreeNode node) {
    return processChildren().indexOf(node); }
```



Arbre avec des TreeNodes (2)

- Affiche l'arbre des fichier du disque.

```
class FileTreeNode implements TreeNode {
  ...
  private List<FileTreeNode> processChildren() {
    if (children!=null)
      return children;

    File[] files=file.listFiles();
    ArrayList<FileTreeNode> list=
      new ArrayList<FileTreeNode>(files.length);
    for(File file:files)
      list.add(new FileTreeNode(this,file));
    this.children=list;
    return list;
  }
  private final File file;
  private final FileTreeNode parent;
  private List<FileTreeNode> children;

  public static void main(String[] args) {
    FileTreeNode root=new FileTreeNode(null,new File("/"));
    DefaultTreeModel model=new DefaultTreeModel(root);
    JTree tree=new JTree(model);
    ...
```



Afficher la valeur d'un noeud

- Deux solutions :
  - Changer le renderer de l'arbre (**`TreeCellRenderer`**).
  - Redéfinir la méthode **`toString()`** du **`TreeNode`** car le **`DefaultTreeCellRenderer`** appel **`toString()`** sur le noeud (ici le **`TreeNode`**).

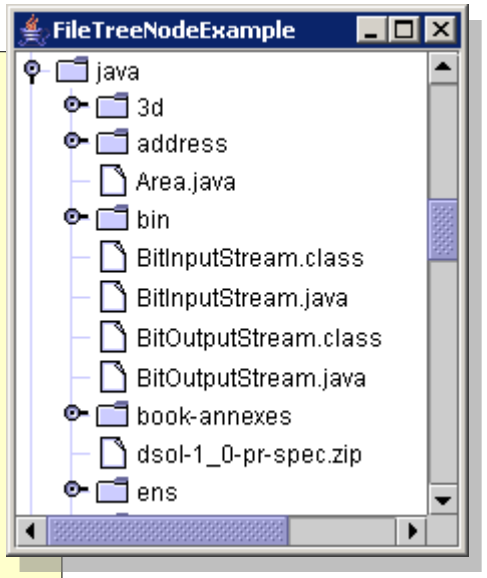
- Un **`TreeCellRenderer`**contient une méthode qui envoie le composant de rendu :

```
public class MyTreeCellRenderer implements TreeCellRenderer {
  public Component getTreeCellRendererComponent(
    JTree tree, Object value, boolean selected, boolean expanded,
    boolean leaf, int row, boolean hasFocus) {

    label.setText(node.file.getName());
    return label;
  }
  private final Jlabel label=new JLabel();
}
```

Afficher la valeur d'un noeud (2)

- Afficher les noeuds en redéfinissant la méthode **`toString()`**.

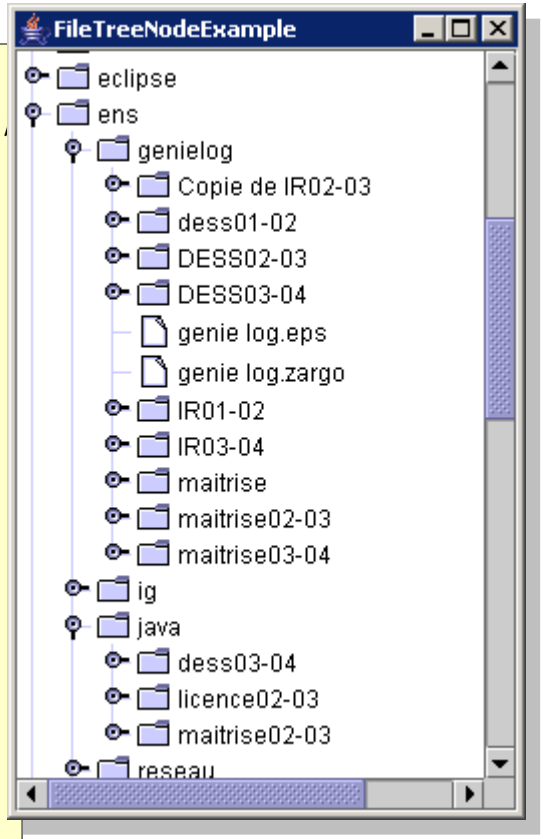
```
class FileTreeNode implements TreeNode {
  public FileTreeNode(FileTreeNode parent,File file) {
    this.parent=parent;
    this.file=file;
  }
  ...
  public String toString() {
    return file.getName();
  }

  private final File file;
  private final FileTreeNode parent;
  private List<FileTreeNode> children;
}
```



Afficher la valeur d'un noeud (3)

- Afficher les noeuds en changeant le renderer.

```
public static void main(String[] args) {
  FileTreeNode root=new FileTreeNode(null,new File("/"));
  DefaultTreeModel model=new DefaultTreeModel(root);
  JTree tree=new Jtree(model);
  tree.setCellRenderer(new DefaultTreeCellRenderer() {
    public Component getTreeCellRendererComponent(
      JTree tree,Object value,boolean selected,
      boolean expanded,boolean leaf,
      int row,boolean hasFocus) {

      super.getTreeCellRendererComponent(
        tree,value,selected,expanded,leaf,row,hasFocus);

      FileTreeNode node=(FileTreeNode)value;
      setText(node.file.getName());

      return this;
    }
  });
```



DefaultMutableTreeNode

- Implantation par défaut des **`MutableTreeNode`**, utilise un **`Vector`** mais n'est pas *ThreadSafe*.
- Constructeur

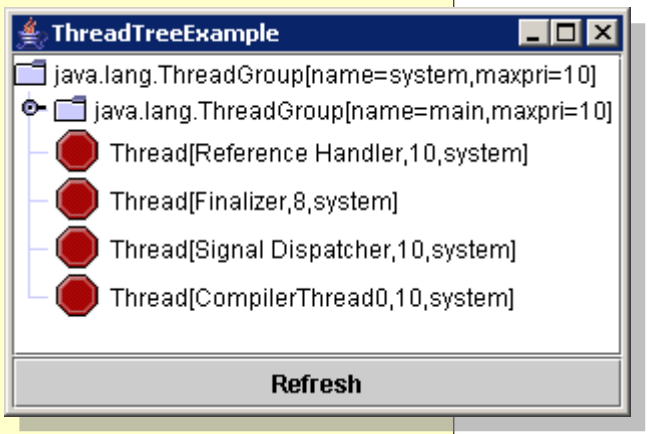
```
DefaultMutableTreeNode(Object donnée, boolean permetFils)
```

- Méthode de gestion des noeuds :
  - Donnée utilisateur (**`get/setUserObject()`**)
  - Ajouter/Retirer (**`add()/remove()`**)
  - Obtenir les fils (**`getChildCount()/getChildAt()`**)
  - Parcours sur l'arbre (**`post/preOrderEnumeration()`**)
- Ne connait pas le modèle, donc pour que les changements soient actifs, il faut appeler les méthodes **`fire*`** du modèle.

Arbre de Threads

- Utilise des **DefaultMutableTreeNode**

```
private static ThreadGroup getRoot(Thread thread) {
  ThreadGroup group=thread.getThreadGroup();
  for(;group.getParent()!=null;group=group.getParent());
  return group;
}
private static DefaultMutableTreeNode createTreeNode(ThreadGroup root) {
  DefaultMutableTreeNode node=new DefaultMutableTreeNode(root);
  ThreadGroup[] groups=new ThreadGroup[root.activeGroupCount()];
  int groupCount=root.enumerate(groups,false);
  for(int i=0;i<groupCount;i++)
    node.add(createTreeNode(groups[i]));
  Thread[] threads=new Thread[root.activeCount()];
  int threadCount=root.enumerate(threads,false);
  for(int i=0;i<threadCount;i++)
    node.add(new DefaultMutableTreeNode(threads[i]));
  return node;
}
static TreeNode createRootNode() {
  return createTreeNode(getRoot(Thread.currentThread()));
}
public static void main(String[] args) {
  final DefaultTreeModel model=new DefaultTreeModel(createRootNode());
```



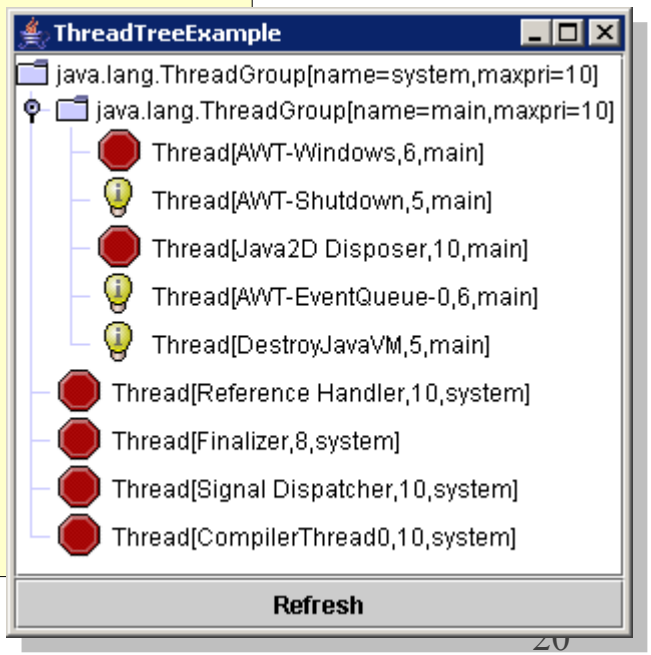
Icône & DefaultTreeCellRenderer (2)

- Changer les icônes associés aux noeuds :
- Pour tous les noeuds de l'arbre :
  - feuilles : **`get/setDefaultLeafIcon()`**
  - Noeud ouvert : **`get/setDefaultOpenIcon()`**
  - Noeud fermé : **`get/setDefaultClosedIcon()`**
- Pour le noeud courant :
  - feuilles : **`get/setLeafIcon()`**
  - Noeud ouvert : **`get/setOpenIcon()`**
  - Noeud fermé : **`get/setClosedIcon()`**
- Les changement sur le noeud courant doivent être fait **AVANT** l'appel à **`super`** dans le renderer.

Icône & DefaultTreeCellRenderer

- Changer les icônes d'un arbre.

```
tree.setCellRenderer(new DefaultTreeCellRenderer() {
  public Component getTreeCellRendererComponent(
    JTree tree,Object value,boolean selected,boolean expanded,
    boolean leaf,int row,boolean hasFocus) {

    if (leaf) {
      DefaultMutableTreeNode node=(DefaultMutableTreeNode)value;
      Thread thread=(Thread)node.getUserObject();
      if (thread.isDaemon())
        setLeafIcon(stopIcon);
      else
        setLeafIcon(okIcon);
    }
    super.getTreeCellRendererComponent(tree,value,
      selected,expanded,leaf,row,hasFocus);
    return this;
  }

  private final Icon stopIcon=loadIcon("Stop24.gif");
  private final Icon okIcon=loadIcon("TipOfTheDay24.gif");
});
```



Edition des noeuds de l'arbre

- Le JTree doit être editable :
  - `tree.setEditable(true)`

- Deux façons de permettre la modification des données :
- Au niveau du **`TreeModel`** :
  - implanter la méthode :

```
valueForPathChanged(TreePath path, Object value)
```

- En utilisant un **`DefaultTreeModel`**:
  - les noeuds doivent implanter dans l'interface **`MutableTreeNode`** la méthode :

```
void setUserObject(Object object)
```

Gestion des évènements

- Evènements de données (sur le modèle)

```
addTreeModelListener(TreeModelListener l)
```

- Evènements de selection (sur le **JTree**)

```
addTreeSelectionListener(TreeSelectionListener l)
```

- Evènements d'expansion (sur le **JTree**)

```
addTreeExpansionListener(TreeExpansionListener l)
```

```
addTreeWillExpandListener(TreeWillExpandListener l)
```

Evènements de données

- Capter par l'interface **`TreeModelListener`**.
- 3 types de changement de données :
  - changement de la valeur d'un noeud.
  - ajout/suppression d'un noeud.
  - changement complexe de l'arborescence.
- Indique des changements sur un ensemble des noeuds fils d'un noeud père.
- Un objet évènement **`TreeModelEvent`**paramètré différemment suivant le changement.

Evènements reliés à la valeur des noeuds

- Contenu d'un noeud à changé, mais l'objet représentant un noeud est le même.
  - **`treeNodesChanged(TreeModelEvent e)`**

- **`TreeModelEvent(Object source, TreePath path, int[] indices, Object[] enfants)`**
  - Modèle générant l'évènement (source).
  - Chemin du noeud père à la racine (path).
  - Indices des noeuds fils ayant changés.
  - Enfants du noeud père ayant changés.

Evènements reliés à la structure des noeuds

- Noeuds fils à ajouter
  - **`treeNodesInserted(TreeModelEvent e)`**
- Noeuds fils à retirer
  - **`treeNodesRemoved(TreeModelEvent e)`**
- Même évènement **`TreeModelEvent`**que pour les changements du contenu d'un noeud :
  - **`TreeModelEvent(Object source, TreePath path, int[] indices, Object[] enfants)`**

Evènements reliés à la structure des noeuds (2)

- Changement complexe de la structure
  - **`treeStructureChanged(TreeModelEvent e)`**

- On indique seulement le noeud père à partir duquel toutes la hiérarchie a changé :
  `TreeModelEvent(Object source, TreePath path)`

- Au niveau de la vue, un changement complexe implique un repliement de l'arbre au niveau du noeud père.

Evènements et DefaultTreeModel

- Le **`DefaultTreeModel`** ne respecte pas complètement le MVC de Swing, il est possible de changer les données du modèle sans que celui-ci informe les vues.

- En effet, les interfaces **`TreeNode`** et **`MutableTreeNode`** n'est pas par défaut relier au modèle.

- Par exemple, les changements effectuées sur un **`DefaultMutableTreeNode`** ne sont pas repercuté au niveau de la vue (même un add).

Evènements et DefaultTreeModel (2)

- Le DefaultTreeModel possède des méthodes publiques qui permettent de signaler un changement d'un noeud.
- Permet à un controleur de signaler les changements effectués sur un TreeNode.

- Indique qu'un noeud à changé
  - **`void nodeChanged(TreeNode node)`**
  - **`void nodesChanged(TreeNode node, int[] childIndices)`**

- Indique qu'un noeud est inséré/retiré
  - **`void nodesWereInserted(TreeNode node, int[] childIndices)`**
  - **`void nodesWereRemoved(TreeNode node, int[] childIndices, Object[] removedChildren)`**

Evènements et DefaultTreeModel (3)

- Indique un changement de la structure :
  - **`void nodeStructureChanged(TreeNode node)`**
  - **`void reload(TreeNode node)`**
- Changement de structure de tout l'arbre :
  - **`void reload()`**
- Équivalent à **`nodeStructureChanged(root)`**

Evènements de sélection

- Un **`TreeSelectionModel`**gère la selection des noeuds de l'arbre.
- Il possède trois modes de sélection :
  - SINGLE_TREE_SELECTION, une seul noeud à la fois.
  - CONTIGUOUS_TREE_SELECTION, noeuds contigus.
  - DISCONTIGUOUS_TREE_SELECTION, noeuds non contigus.
- Pour simplifier l'utilisation, quelques méthodes du modèle de sélection son présente sur **`JTree`**.
  - **`get/setSelectionRow(int row)`**
  - **`get/setSelectionRows(int[] rows)`**
  - **`get/setSelectionPath(TreePath path)`**
  - **`get/setSelectionPaths(TreePath[] paths)`**

Evènements d'expansion

- Deux listeners de dépliage/repliage des noeuds : évènement avant ou après pliage/dépliage.

- Listener des évènements avant dépliage/pliage : **`TreeWillExpandListener`**
  - **`void treeWillCollapse(TreeExpansionEvent event) throws ExpandVetoException`**
  - **`void treeWillExpand(TreeExpansionEvent event) throws ExpandVetoException`**

- On peut empêcher l'arbre d'effectuer une action en renvoyant une exception **`ExpandVetoException`**

Evènements d'expansion (2)

- Listener des évènements après dépliage/pliage
  :TreeExpansionListener
  - void treeCollapsed(TreeExpansionEvent event)
  - void treeExpanded(TreeExpansionEvent event)

- L'évènement **TreeExpansionEvent**indique le chemin vers le noeud déplié/replié :
  TreeExpansionEvent(Object source,
  TreePath path)

- La méthode : **getPath()** permet d'obtenir le chemin.

Exemple avec des évènements

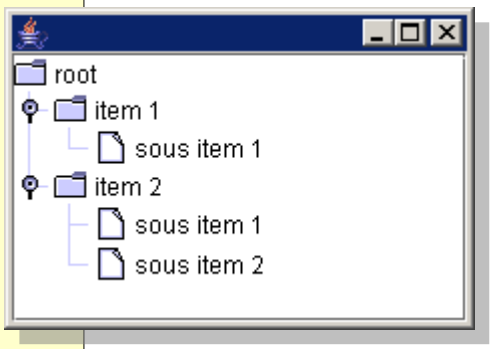
- **`MyTreeNode`** est une classe interne du modèle.

```
public class MutableTreeExample {
  public static class MyTreeModel<T> extends DefaultTreeModel {
    public MyTreeModel(T value) {
      super(null);
      setRoot(new MyTreeNode(null,value));
    }
    public MyTreeNode getRootTreeNode() {
      return (MyTreeNode)getRoot();
    }
    public class MyTreeNode implements TreeNode {
      MyTreeNode(MyTreeNode parent, T value) {
        this.parent= parent;
        this.value= value;
      }
      public int getChildCount() {
        return list.size();
      }
      public boolean getAllowsChildren() {
        return true;
      }
      public boolean isLeaf() {
        return list.isEmpty();
      }
      public Enumeration<?> children() {
        return Collections.enumeration(list);
      }
      public MyTreeNode getParent() {
        return parent;
      }
      public MyTreeNode getChildAt(int index){
        return list.get(index);
      }
      public int getIndex(TreeNode node) {
        return list.indexOf(node);
      }
      public MyTreeNode add(T value) {
        // à compléter
      }
      public String toString() {
        return value.toString();
      }
      private final T value;
      private final MyTreeNode parent;
      private final ArrayList<MyTreeNode> list=
        new ArrayList<MyTreeNode>();
    }
```

Exemple avec des évènements (2)

- Le noeud signale les ajouts au modèle.

```
public static void main(String[] args) {
  MyTreeModel<String> model=
    new MyTreeModel<String>("root");
  MyTreeModel<String>.MyTreeNode root=model.getRootTreeNode();

  MyTreeModel<String>.MyTreeNode node1=root.add("item 1");
  node1.add("sous item 1");

  MyTreeModel<String>.MyTreeNode node2=root.add("item 2");
  node2.add("sous item 1");
  node2.add("sous item 2");

  final JTree tree=new JTree(model);
  final JFrame frame=new JFrame();

  JPanel panel=new JPanel(new BorderLayout());
  panel.add(new JScrollPane(tree),BorderLayout.CENTER);

  frame.setContentPane(panel);
  frame.pack();
  frame.setVisible(true);
}
```

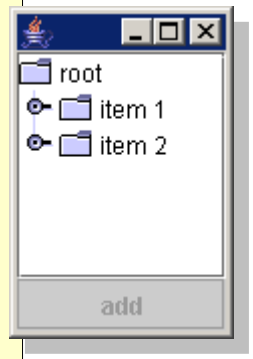


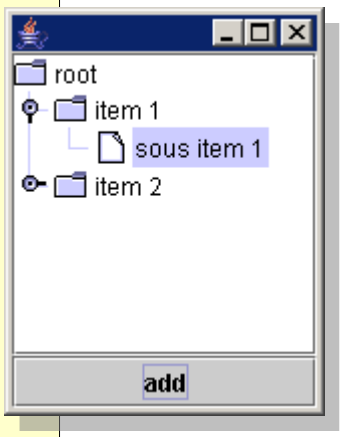

```
public MyTreeNode add(T value) {
  MyTreeNode child= new MyTreeNode(this, value);
  list.add(child);
  nodesWereInserted(this, new int[]{list.size()-1});
  return child;
}
```

Exemple avec des évènements (3)

- Boutton qui ajoute un noeud à la sélection.

```
final JButton button=new JButton("add");
button.setEnabled(false);
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    String text=JOptionPane.showInputDialog(frame,
      "entrer un nom");
    if (text==null)
      return;

    MyTreeModel<String>.MyTreeNode node=(MyTreeModel<String>.MyTreeNode)
      tree.getSelectionPath().getLastPathComponent();
    node.add(text);
  }
});
tree.getSelectionModel().addTreeSelectionListener(
  new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
      button.setEnabled(tree.getSelectionPath()!=null);
    }
  }
);
...
panel.add(button,BorderLayout.SOUTH);
```

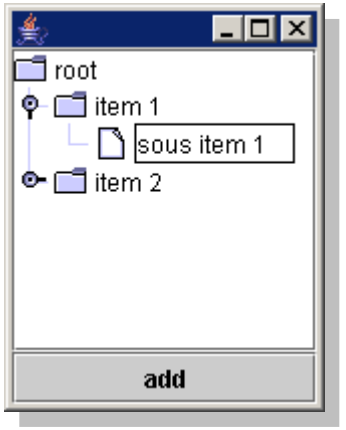




Exemple avec des évènements (4)

- Rendre les noeuds éditables.

```
public static void main(String[] args) {
  ...
  final JTree tree=new JTree(model);
  tree.setEditable(true);
  ...
}
```



- Implanter la méthode **`valueForPathChanged`**

```
public class MyTreeModel<T> extends DefaultTreeModel {
  ...
  public void valueForPathChanged(TreePath path, Object newValue) {
    MyTreeNode node=(MyTreeNode)path.getLastPathComponent();
    node.setValue((T)newValue);

    nodeChanged(node);
  }
  ...
}
```