Le MVC (Suite)

Rémi Forax

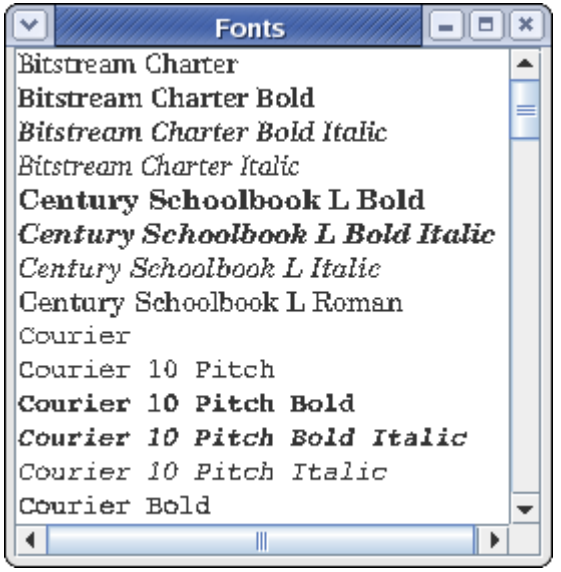
Plan

- Modèle et Adapter
- Gestion des évènements
- Mixer deux composants

Un modèle de liste de fontes

- Permet d'afficher toutes les fontes disponibles

```
public class FontListModel extends AbstractListModel {
  public FontListModel(GraphicsEnvironment env) {
    Font[] fonts=env.getAllFonts();
    for(int i=0;i<fonts.length;i++)
      fonts[i]=fonts[i].deriveFont(14f);
    Arrays.sort(fonts,new Comparator<Font>() {
      public int compare(Font f1,Font f2) {
        return f1.getName().compareTo(f2.getName());
      }
    });
    this.fonts=fonts;
  }
  public int getSize() {
    return fonts.length;
  }

  public Font getElementAt(int index) {
    return fonts[index];
  }
  private final Font[] fonts;
  ...
```

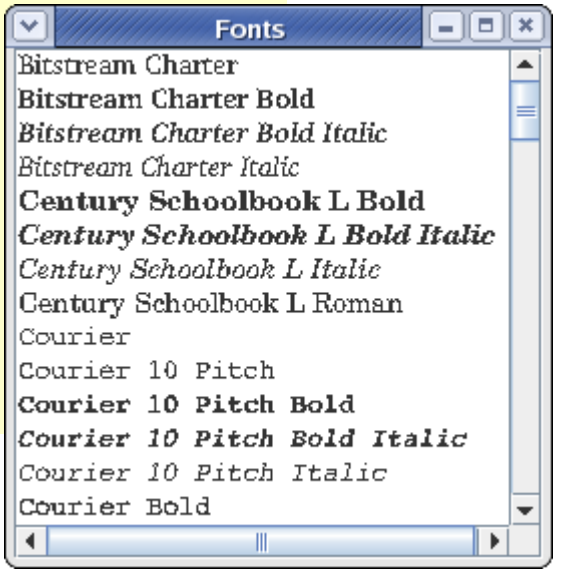


- Un renderer particulier affiche le texte avec la bonne fonte

Le renderer

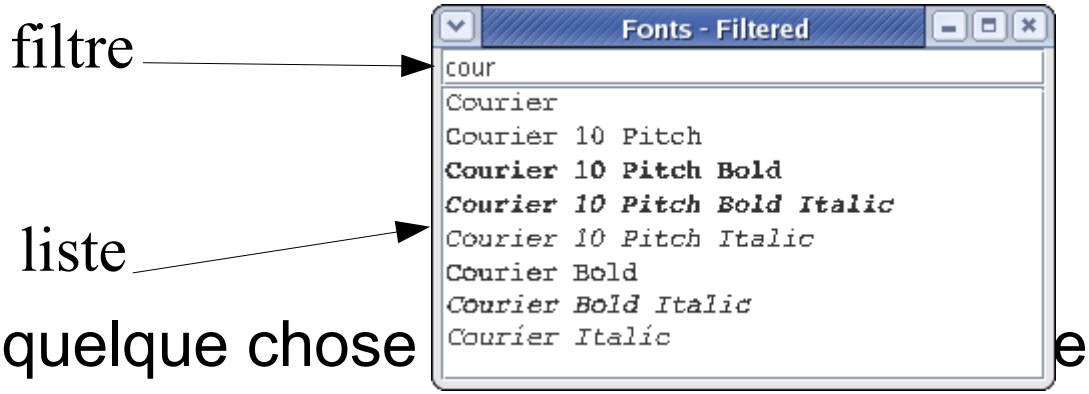
- Affiche chaque fonte en utilisant elle-même comme fonte d'affichage.

```
public class FontListModel extends AbstractListModel {
  ...
  public static void main(String[] args) {
    GraphicsEnvironment env=
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    ListModel model=new FontListModel(env);
    JList list=new JList(model);
    list.setCellRenderer(new DefaultListCellRenderer() {
      public Component getListCellRendererComponent(JList list,
        Object value,int index,boolean isSelected,boolean cellHasFocus) {

        super.getListCellRendererComponent(
          list,value,index,isSelected,cellHasFocus);
        Font font=(Font)value;
        setText(font.getName());
        setFont(font);
        return this;
      }
    });

  ...
```

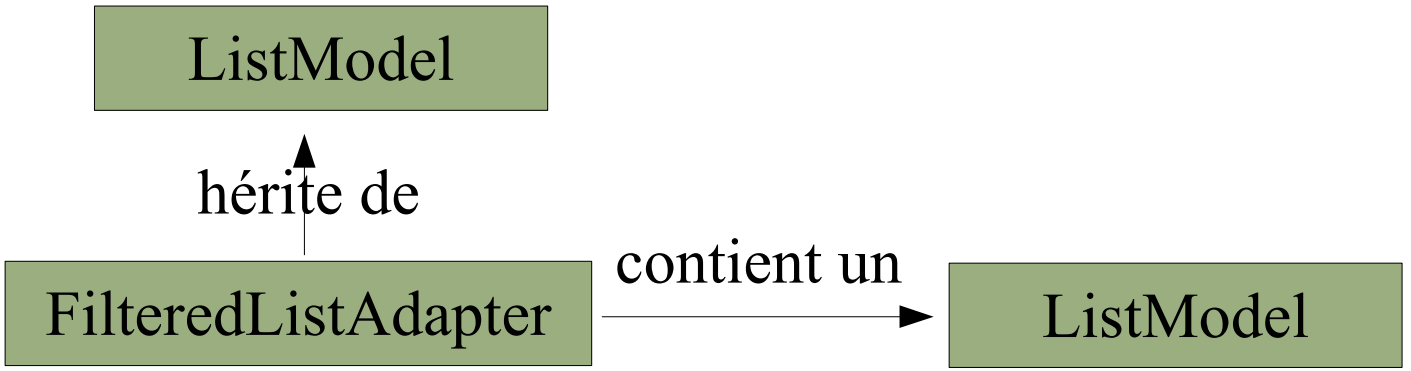


Affichage filtrée

- On souhaite filtrer l'affichage en fonction de l'entrée de l'utilisateur



- Si l'on veut faire quelque chose e



L'adapter

- Le **FilteredListAdapter** est un **ListModel** pour la vue mais délégue à un **ListModel** le soin de stocker les données
- Le **FilteredListAdapter** utilise un objet **Predicate** pour savoir si un objet doit ou nom apparaître dans la liste
- Un exemple de prédicat :

```
public interface Predicate<T> {
  boolean accept(T t);
}
```

```
public class StartsWithPredicate implements Predicate<String> {
  public StartsWithPredicate(String prefix) {
    this.prefix=prefix;
  }
  public boolean accept(String text) {
    return text.startsWith(prefix);
  }
  private final String prefix;
}
```

L'adapter (2)

```
public class FilteredListAdpater<T> extends AbstractListModel {
  public FilteredListAdapter(ListModel model,Predicate<T> predicate)
{
    this.model=model;
    this.predicate=predicate;
  }
  public int getSize() {
    int count=0;
    for(int i=0;i<model.getSize();i++) {
      if (predicate.accept((T)model.getElementAt(i)))
        count++;
    }
    return count;
  }

  public T getElementAt(int index) {
    int count=0;
    for(int i=0;i<model.getSize();i++) {
      T value=(T)model.getElementAt(i);
      if (predicate.accept(value)) {
        if (count==index)
          return value;
        count++;
      }
    }
    throw new AssertionError("bad index "+index);
  }

  private final ListModel model;
  private final Predicate<T> predicate;
```

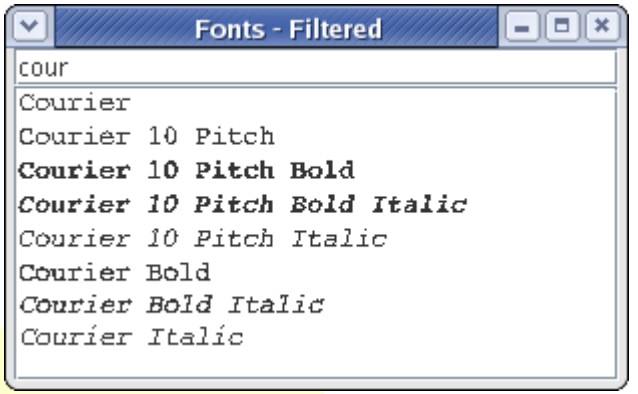
Utilisation de l'adapter

- Utilisation de l'adapter comme modèle de la liste

```
...
  ListModel model=new FontListModel(env);
  Predicate<Font> predicate=new Predicate<Font>() {
    public boolean accept(Font font) {
      return font.getName().toLowerCase().startsWith("c");
    }
  };
  ListModel adapter=new FilteredListAdapter(model,predicate);
  JList list=new Jlist(adapter);
...
```

Prédicat dynamique

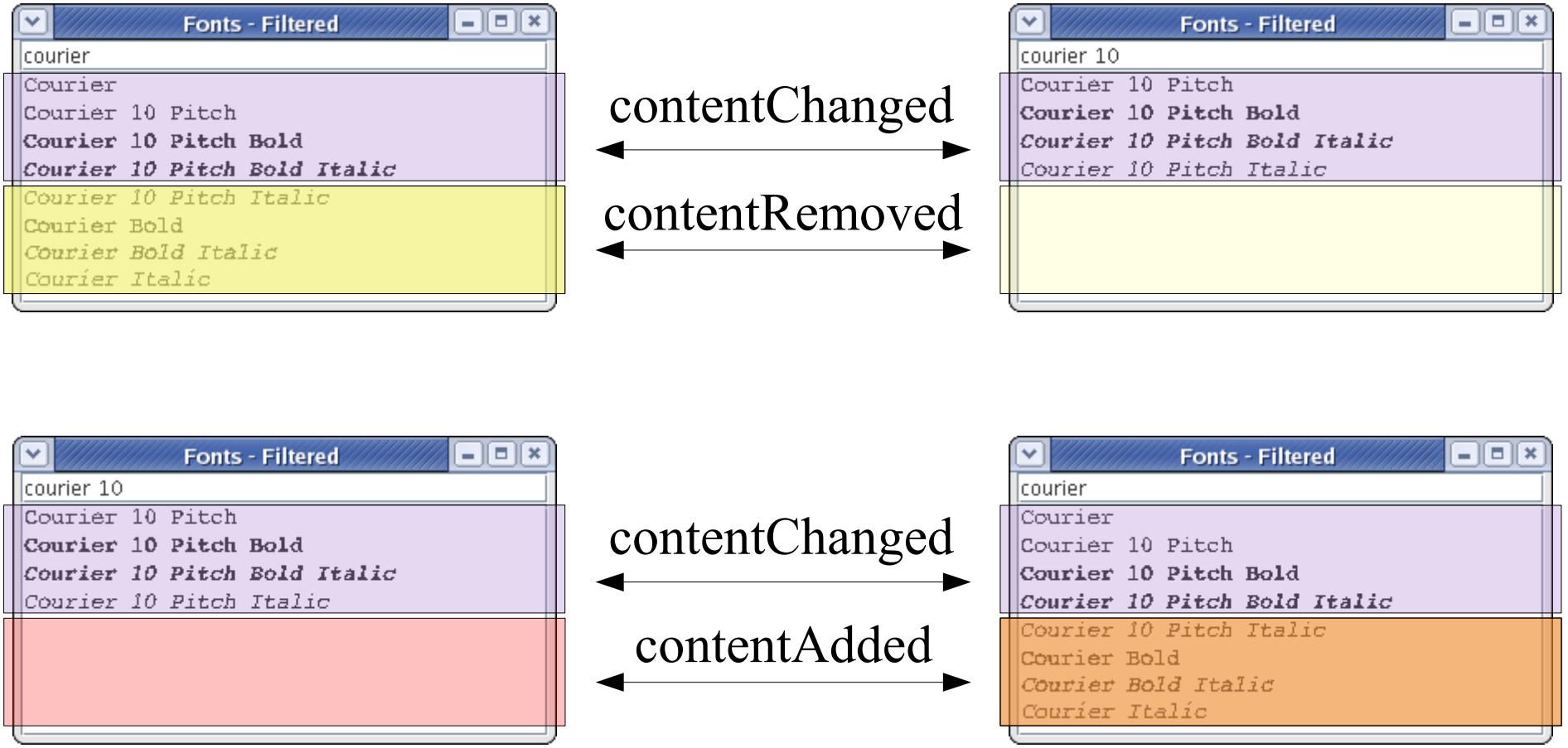
- Lorsque l'utilisateur tape des lettres en tant que filtre, il change le prédicat
- Il faut donc avertir la liste par des **fire***




```
...
public void setPredicate(Predicate<T> predicate) {
  int size1=getSize();
  this.predicate=predicate;
  int size2=getSize();

  fireContentsChanged(this,0,Math.min(size1,size2)-1);
  if (size1<size2)
    fireIntervalAdded(this,size1,size2-1);
  else
    if (size1>size2)
      fireIntervalRemoved(this,size2,size1-1);
}

private Predicate<T> predicate;
...
```

Prédicat dynamique (2)

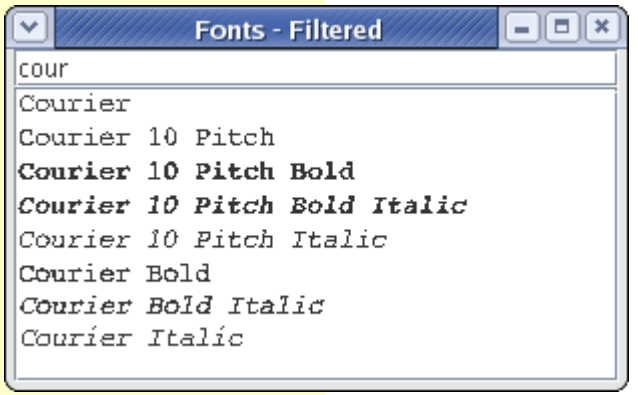
- En regardant la taille avant et après le changement de prédicat



Prédicat dynamique (3)

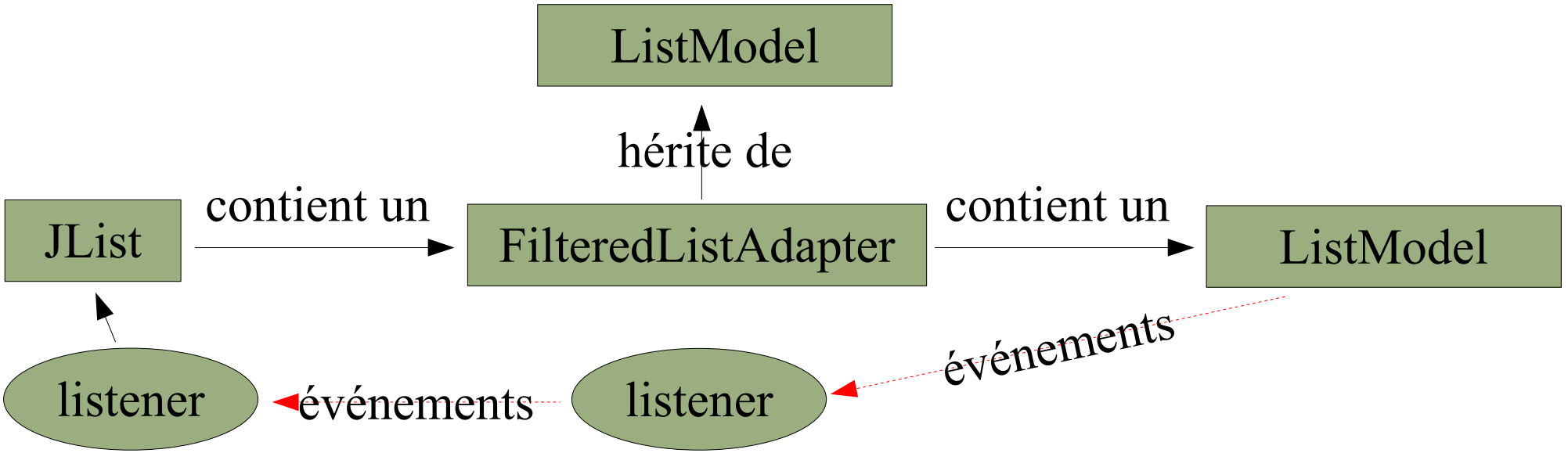
- Le texte du *textfield* est utilisé comme prédicat pour le filtrage effectuée par l'adapter

```
ListModel model=new FontListModel(env);
final ListModel adapter=new FilteredListAdapter(model);
JList list=new Jlist(adapter);
list.setCellRenderer(...);

final JTextField field=new JTextField();
field.addCaretListener(new CaretListener() {
  String getText() {
    String text=field.getText();
    if (text==null)
      return "";
    return text;
  }
  public void caretUpdate(CaretEvent e) {
    adapter.setPredicate(new Predicate<Font>() {
      public boolean accept(Font font) {
        return font.getName().toLowerCase().startsWith(getText());
      }
    });
  }
});
```



Modèle de fontes dynamique

- Le FilteredListAdpater n'écoute pas les changement du modèle (le modèle de fonte dans l'exemple)
- Il faut que l'adapter s'enregistre auprès du modèle en tant que **listener** et transforme les événement de changement du modèle en événement de changement pour l'adapter



Exemple dynamique

```
public class DynamicFontListModel extends AbstractListModel {
  public DynamicFontListModel(final File path) {
    ExecutorService service=Executors.newFixedThreadPool(10);
    for(final File file:path.listFiles(TTF_FILTER))
      service.execute(new Runnable() {
        public void run() {
          try {
            Font loadedFont=Font.createFont(Font.TRUETYPE_FONT,file);
            final Font font=loadedFont.deriveFont(14f);
            EventQueue.invokeLater(new Runnable() {
              public void run() {
                int index=fonts.size();
                fonts.add(font);
                fireIntervalAdded(this,index,index);
              }
            });
          } catch(FontFormatException e) {
          } catch(IOException e) {
          }
        }
      });
  }
  public int getSize() {
    return fonts.size();
  }
  public Font getElementAt(int index) {
    return fonts.get(index);
  }
  final ArrayList<Font> fonts=new ArrayList<Font>();
```

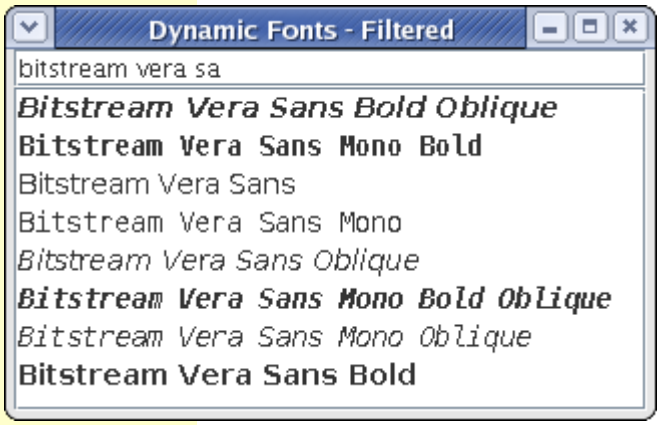
Exemple dynamique (2)

- Le **main** dépend de la plate-forme

```
public static void main(String[] args) {
  // windows XP
  ListModel model=new DynamicFontListModel(
   new File("C:\\windows\\fonts"));
  // linux fedora
  ListModel model=new DynamicFontListModel(
    new File("/usr/share/fonts/bitstream-vera"));
  JList list=new JList(model);
```

Adapter et Listeners

```
public class FilteredListAdapter<T> extends AbstractListModel {
  public FilteredListAdapter(final ListModel model) {
    this.model=model;
    this.predicate=ALWAYS_TRUE_PREDICATE;
    model.addListDataListener(new ListDataListener() {
      private List<Integer> processIndexes(int first,int last) {
        ArrayList<Integer> indexes=new ArrayList<Integer>();
        int count=0;
        for(int i=0;i<=last;i++) {
          if (predicate.accept((T)model.getElementAt(i))) {
            if (i>=first)
              indexes.add(count);
            count++;
          }
        }
        return indexes;
      }
      public void intervalAdded(ListDataEvent e) {
        for(int index:processIndexes(e.getIndex0(),e.getIndex1()))
          fireIntervalAdded(FilteredListAdapter.this,index,index);
      }
      public void intervalRemoved(ListDataEvent e) {
        // même chose avec fireIntervalRemoved
      }
      public void contentsChanged(ListDataEvent e) {
        // même chose avec fireContentsChanged
      }
    });
  }
```



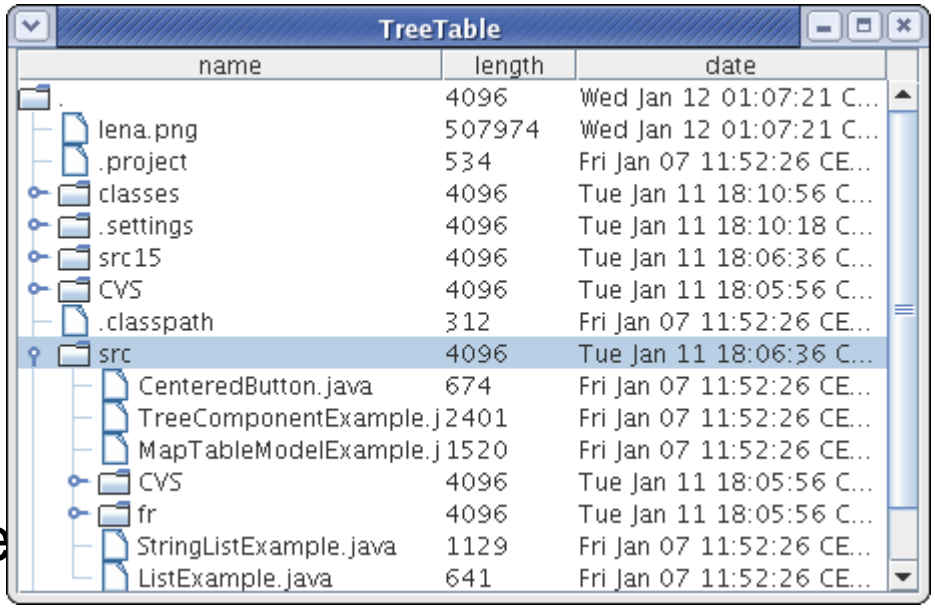
En résumé

- Lorsque l'on veut une propriété indépendante de la façon de stocker les données sur un modèle, on écrit un adapter à ce modèle
- Un adapter est lui-même un modèle qui délègue le stockage des données à un modèle sous-jacent
- Un adapter doit s'enregistrer en tant que *listener* du modèle sous-jacent et effectuer les transformations d'indices

- Il est possible d'avoir un adapter d'un modèle vers un autre, exemple, voir un arbre comme une liste

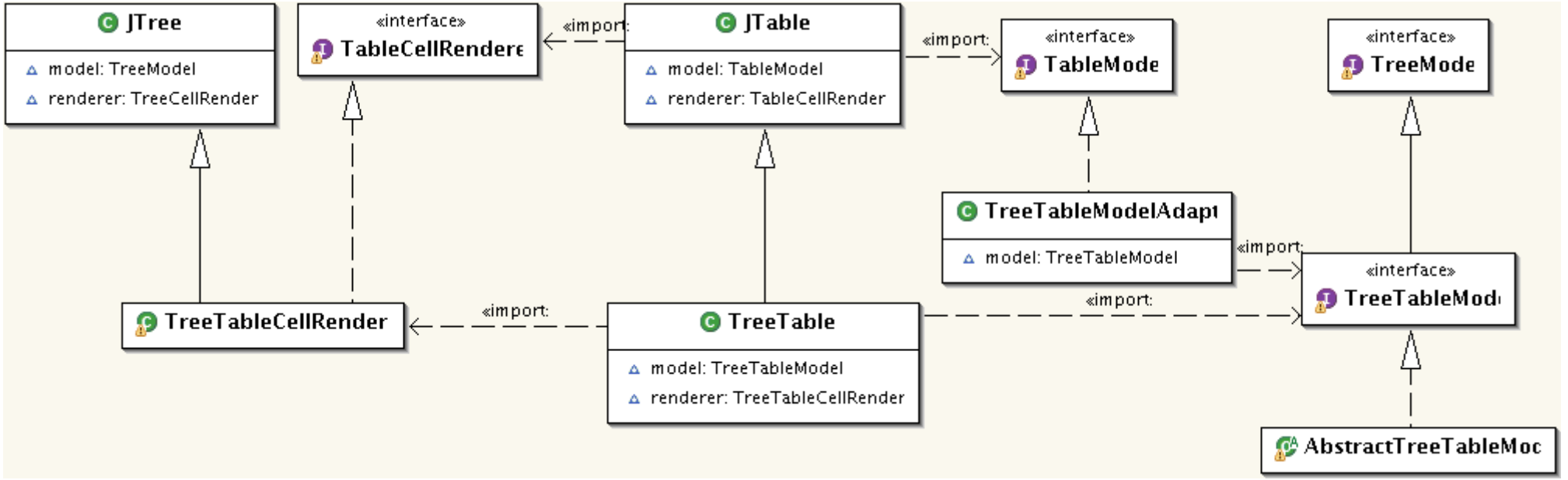
TreeTable

- Table dont la première colonne est un arbre
  Ex: le Finder de MacOS 9
- En swing, il n'existe pas de composant JTreeTable
- Il est possible d'écrire assez facilement ce composant grâce
- TreeTable est une JTable pour laquelle le renderer de la première colonne est un arbre



TreeTable (2)

- Diagramme des classes (UML)



- AbstractTreeTableModel implante TreeTableModel et fournie la gestion des listeners

TreeTableModel

- Le TreeTableModel hérite de TreeModel et ajoute les méthodes :
  - getColumnCount() nombre de colonne
  - getColumnName(int column) nom d'une colonne
  - getColumnClass(int column) type d'une colonne
  - getValueAt(Object node,int column) valeur du noeud pour une colonne
  - isCellEditable(Object node,int column) une cellule est editable
  - setValueAt(Object value,Object node,int column) change la valeur du noeud pour une colonne

AbstractTreeTableModel

- Cette classe va s'occuper de la gestion des listeners sur le TreeTableModel

```
public abstract class AbstractTreeTableModel implements TreeTableModel {
  public boolean isLeaf(Object node) {
    return getChildCount(node)==0;
  }
  public int getIndexOfChild(Object parent, Object child) {
    for(int i=0;i<getChildCount(parent);i++)
      if (getChild(parent,i).equals(child))
        return i;
    return -1;
  }
  public String getColumnName(int column) {
    StringBuilder name=new StringBuilder();
    for(;column>=0;column=column/26-1) {
      name.insert(0,(char)(column%26+'A'));
    }
    return name.toString();
  }
  public Class<?> getColumnClass(int column) {
    return Object.class;
  }
```

AbstractTreeTableModel (2)

```
  public boolean isCellEditable(Object node, int column) {
    return false;
  }
  public void setValueAt(Object aValue, Object node, int column) {
  }
  public void valueForPathChanged(TreePath path, Object newValue) {
  }
  public void addTreeModelListener(TreeModelListener l) {
    listeners.add(l);
  }
  public void removeTreeModelListener(TreeModelListener l) {
    listeners.remove(l);
  }
  protected void fireTreeNodesChanged(Object source, Object[] path,
    int[] indices,Object[] children) {

    if (listeners.isEmpty())
      return;
    TreeModelEvent e=new TreeModelEvent(source,path,indices,children);
    for(int i=listeners.size();--i>=0;)
      listeners.get(i).treeNodesChanged(e);
  }
  // + les autres fires
  private final ArrayList<TreeModelListener> listeners=
    new ArrayList<TreeModelListener>();
}
```

Arbre comme renderer

- Installer l'arbre en tant que renderer

```
public class TreeTable extends JTable {
  public TreeTable(TreeTableModel model) {
    TreeTableCellRenderer renderer=
      new TreeTableCellRenderer(model);

    setModel(new TreeTableModelAdapter(model,renderer));
    renderer.setRowHeight(getRowHeight());

    getColumnModel().getColumn(0).setCellRenderer(renderer);

    setShowGrid(false);
    setIntercellSpacing(new Dimension(0, 0));
  }
  class TreeTableCellRenderer extends JTree
    implements TableCellRenderer {
    ...
  }
}
```

- Problème, seul le début de l'arbre sera visible dans chaque case

Arbre comme renderer (2)

- En fonction de la case à voir il faut décaler l'affichage

```
class TreeTableCellRenderer extends JTree implements TableCellRenderer {
  public TreeTableCellRenderer(final TreeTableModel model) {
    super(model);
  }
  public void setBounds(int x, int y, int w, int h) {
    super.setBounds(x, 0, w, TreeTable.this.getHeight());
  }
  public void paint(Graphics g) {
    g.translate(0, -visibleRow*getRowHeight());
    super.paint(g);
  }
  public Component getTableCellRendererComponent(JTable table,
    Object value,boolean isSelected,boolean hasFocus,int row,int column) {
    if (isSelected)
      setBackground(table.getSelectionBackground());
    else
      setBackground(table.getBackground());

    visibleRow = row;
    return this;
  }
  private int visibleRow;
}
```

Le renderer de l'arbre

- La table va utiliser l'arbre comme composant de rendu, l'arbre possède lui-même un renderer
- Par défaut, celui-ci va appeler toString() sur nun noued de l'arbre or il faut que celui-ci utilise la méthode getValueAt() avec comme index de colonne, 0

```
class TreeTableCellRenderer extends JTree implements TableCellRenderer {
  public TreeTableCellRenderer(final TreeTableModel model) {
    super(model);
    setCellRenderer(new DefaultTreeCellRenderer() {
      public Component getTreeCellRendererComponent(JTree tree,Object value,
        boolean sel,boolean expanded,boolean leaf,int row,boolean hasFocus) {

          return super.getTreeCellRendererComponent(tree,model.getValueAt(value,0),
            sel,expanded,leaf,row,hasFocus);
        }
      });
  }
  ...
}
```

Adapter le TreeTableModel

- Pour visualiser le TreeTableModel dans la JTable, il faut un adapter qui permette de voir le modèle comme un TableModel

```
public class TreeTableModelAdapter extends AbstractTableModel {
  public TreeTableModelAdapter(TreeTableModel treeTableModel, JTree tree) {
    this.tree=tree;
    this.treeTableModel=treeTableModel;
  }
  public int getColumnCount() {
    return treeTableModel.getColumnCount();
  }
  public String getColumnName(int column) {
    return treeTableModel.getColumnName(column);
  }
  public Class<?> getColumnClass(int column) {
    return treeTableModel.getColumnClass(column);
  }
  public int getRowCount() {
    return tree.getRowCount();
  }
  public Object getValueAt(int row, int column) {
    return treeTableModel.getValueAt(nodeForRow(row),column);
  }
  ...
```

TreeTableModelAdapter

- Le JTree permet la conversion entre une ligne et le noeud correspondant

```
public class TreeTableModelAdapter extends AbstractTableModel {
  ...
  public boolean isCellEditable(int row, int column) {
    if (column==0) // voir transparent suivant
      return true;
    return treeTableModel.isCellEditable(nodeForRow(row),column);
  }
  public void setValueAt(Object value, int row, int column) {
    treeTableModel.setValueAt(value,nodeForRow(row),column);
  }
  private Object nodeForRow(int row) {
    TreePath treePath = tree.getPathForRow(row);
    return treePath.getLastPathComponent();
  }
  private final JTree tree;
  private final TreeTableModel treeTableModel;
}
```

Pliage/dépliage de l'arbre

- Pour permettre à l'utilisateur de plier/déplier l'arbre, il faut que l'évènement transmis à la liste soit intercepter par l'arbre.
- Il faut pour cela que :
  - la colonne contenant l'arbre soit déclarée comme éditable
  - l'arbre soit le composant renvoyé par le TableCellEditor
  - le modèle écoute les changements d'état de l'arbre pour demander un rafraîchissement de la table

```
public class TreeTable extends JTable {
  public TreeTable(TreeTableModel model) {
    ...
    getColumnModel().getColumn(0).setCellRenderer(renderer);
    getColumnModel().getColumn(0).setCellEditor(
      new TreeTableCellEditor(renderer));
    ...
  }
```

TableTreeCellEditor

- L'éditeur capte l'évènement, le retransmet à l'arbre mais indique que la cellule ne sera pas éditée

```
class TreeTableCellEditor extends AbstractCellEditor implements TableCellEditor {
  public TreeTableCellEditor(TreeTableCellRenderer tree) {
    this.tree=tree;
  }
  public Component getTableCellEditorComponent(JTable table,Object value,
    boolean isSelected,int row,int column) {
    return tree;
  }
  public Object getCellEditorValue() {
    throw new UnsupportedOperationException();
  }
  public boolean isCellEditable(EventObject e) {
    if (!(e instanceof MouseEvent))
      return false;
    MouseEvent event=(MouseEvent)e;
    tree.dispatchEvent(new MouseEvent(tree,event.getID(),
      event.getWhen(),event.getModifiers(),event.getX(),event.getY(),
      event.getClickCount(),event.isPopupTrigger(),event.getButton()));
    return false;
  }
  private final TreeTableCellRenderer tree;
}
```

Listener de l'adapter

- L'adapter doit écouter le JTree pour demander un rafraîchissement de la table si l'arbre est plié ou déplié.

```
public class TreeTableModelAdapter extends AbstractTableModel {
  public TreeTableModelAdapter(TreeTableModel treeTableModel, JTree tree) {
    this.tree=tree;
    this.treeTableModel=treeTableModel;
    tree.addTreeExpansionListener(new TreeExpansionListener() {
      public void treeExpanded(TreeExpansionEvent event) {
        fireTableDataChanged();
      }
      public void treeCollapsed(TreeExpansionEvent event) {
        fireTableDataChanged();
      }
    });
  }
  ...
}
```

Gestion de la sélection

- La sélection doit être la même pour le TreeTable (donc la JTable) et le JTree, le mieux serait qu'il partage le même modèle de sélection
- Problème :
  - JTree => TreeSelectionModel
  - JTable => ListSelectionModel (pour les lignes)
- Astuce :
  - DefaultTreeSelectionModel utilise un DefaultListSelectionModel pour stocker son état en interne

Gestion de la sélection (2)

- Le champs selectionModel de DefaultTreeSelectionModel est partagée par la JTable et le JTree

```
public class TreeTable extends JTable {
  public TreeTable(TreeTableModel model) {
    final TreeTableCellRenderer renderer=new TreeTableCellRenderer(model);
    setModel(new TreeTableModelAdapter(model,renderer));

    renderer.setSelectionModel(new DefaultTreeSelectionModel() {
      { // bloc d'initialisation
        TreeTable.this.setSelectionModel(selectionModel);
      }
    });
  }
  ...
```

- Léger problème, le DefaultTreeSelectionModel n'écoute pas les changements de son champs selectionModel

Gestion de la sélection (3)

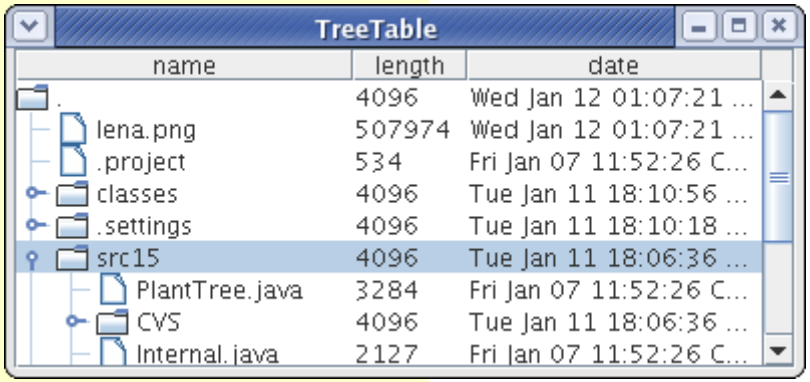
- On écoute les changement du ListSelectionModel et répercute les changement sur le DefaultTreeSelectionModel

```
public class TreeTable extends JTable {
  public TreeTable(TreeTableModel model) {
    final TreeTableCellRenderer renderer=new TreeTableCellRenderer(model);
    setModel(new TreeTableModelAdapter(model,renderer));

    renderer.setSelectionModel(new DefaultTreeSelectionModel() {
      { // bloc d'initialisation
        TreeTable.this.setSelectionModel(selectionModel);
        selectionModel.addListSelectionListener(new ListSelectionListener() {
          public void valueChanged(ListSelectionEvent e) {
            clearSelection(); // efface la selection de l'arbre
            if (selectionModel.isSelectionEmpty())
              return;
            for(int i=selectionModel.getMinSelectionIndex();
                    i<=selectionModel.getMaxSelectionIndex();i++) {
              addSelectionPath(renderer.getPathForRow(i));
            }
          }
        });
      }
    });
  }
  ...
```

Implantation d'un TreeTableModel

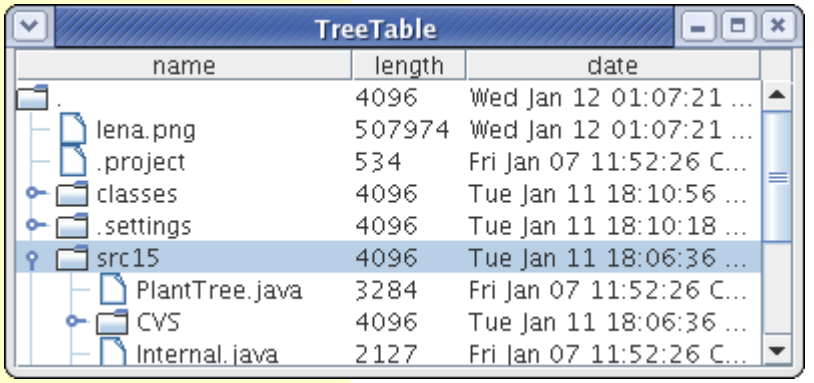
- Un TreeTableModèle pour les fichiers

```
public class FileTreeTableModel extends AbstractTreeTableModel {
  public FileTreeTableModel(File root) {
    this.root=root;
  }
  public Object getRoot() {
    return root;
  }
  public int getChildCount(Object parent) {
    File file=(File)parent;
    if (file.isDirectory())
      return file.list().length;
    return 0;
  }
  public Object getChild(Object parent, int index) {
    return ((File)parent).listFiles()[index];
  }
  public int getColumnCount() {
    return 3;
  }
  public String getColumnName(int column) {
    return COLUMN_NAMES[column];
  }
  ...
```



FileTreeTableModel

```
  ...
  public Object getValueAt(Object node, int column) {
    File file=(File)node;
    switch(column) {
      case 0:
        return file.getName();
      case 1:
        return file.length();
      case 2:
        return new Date(file.lastModified());
    }
    throw new AssertionError("bad column "+column);
  }
  private final File root;
  private static final String[] COLUMN_NAMES={
    "name","length","date"};
  public static void main(String[] args) {
    TreeTableModel model=new FileTreeTableModel(new File("."));
    TreeTable treeTable=new TreeTable(model);

    JFrame frame=new JFrame("TreeTable");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setContentPane(new JScrollPane(treeTable));
    frame.setSize(400,300);
    frame.setVisible(true);
  }
}
```



TreeTableModel dynamique ?

- Et si le modèle TreeTableModèle est lui-même dynamique.
- Il faut que la JTable soit avertie des changement du modèle

```
public class TreeTableModelAdapter extends AbstractTableModel {
  public TreeTableModelAdapter(TreeTableModel treeTableModel,JTree tree) {
    this.tree=tree;
    this.treeTableModel=treeTableModel;
    tree.addTreeExpansionListener(new TreeExpansionListener() {
      ...
    });
    treeTableModel.addTreeModelListener(new TreeModelListener() {
      public void treeNodesChanged(TreeModelEvent e) {
        fireTableDataChanged();
      }
      public void treeNodesInserted(TreeModelEvent e) {
        fireTableDataChanged();
      }
      public void treeNodesRemoved(TreeModelEvent e) {
        fireTableDataChanged();
      }
      public void treeStructureChanged(TreeModelEvent e) {
        fireTableDataChanged();
      }
    });
  }
  ...
```