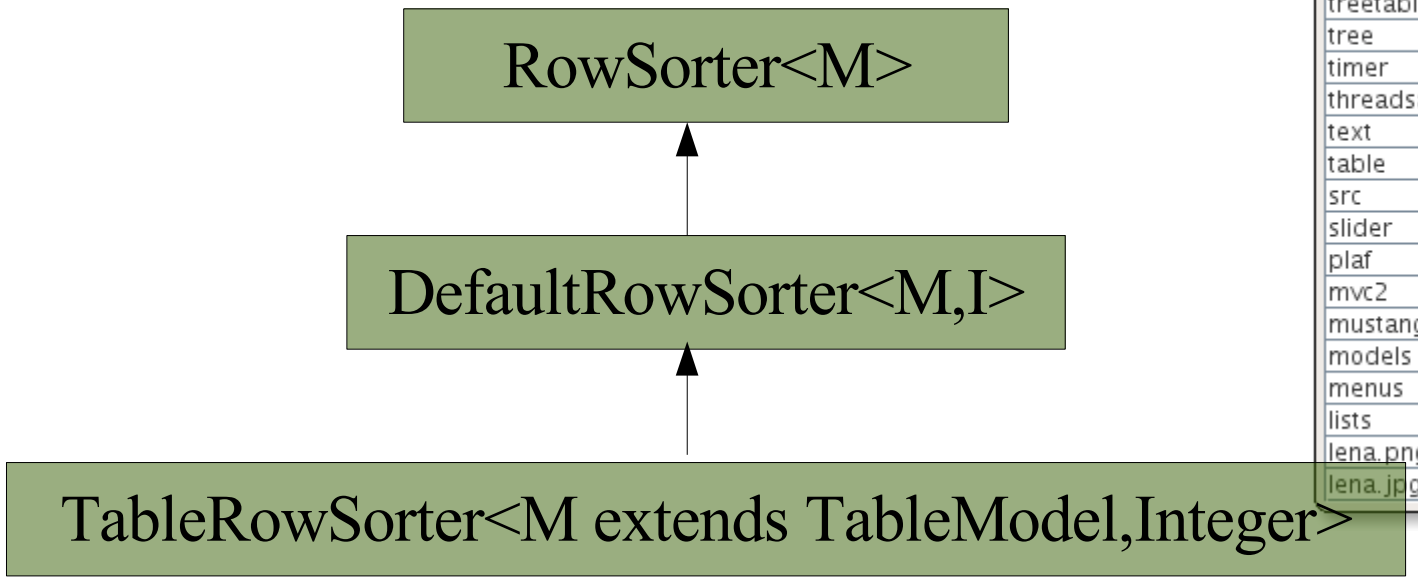
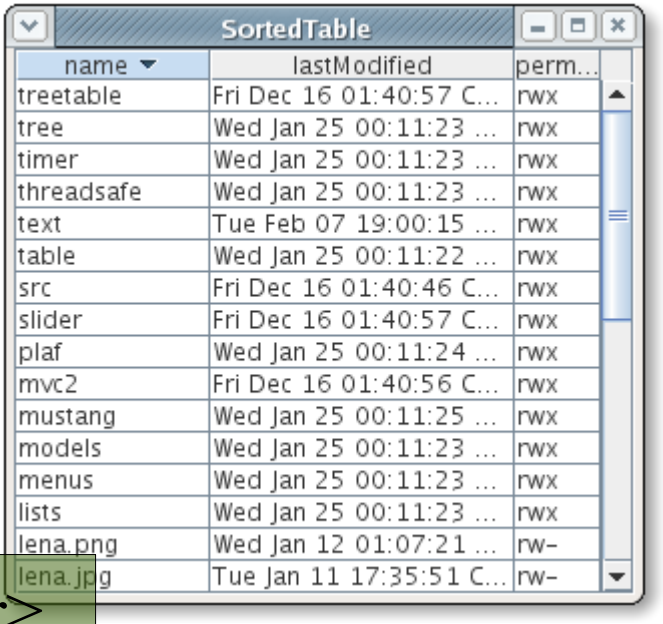
Table (suite)

Rémi Forax

Trie sur les JTable

- La version 1.6 ajoute un mécanisme (**javax.swing.RowSorter**) qui permet de trier des composants dont le modèle s'exprime sous forme de ligne
- Pour l'instant seule **JTable** implante ce mécanisme (**javax.swing.table.TableRowSorter**)





RowSorter

- javax.swing.RowSorter<M>
  - M représente le type de Modèle (bornée par Object)
  - Classe abstraite qui décrit le *mapping* entre les indices de la vues et ceux du modèle
  - Ne définie que la gestion des *listeners*

- Méthodes de conversion :
  - Index de ligne de la vue vers le modèle
    - int convertRowIndexToModel(int index)
  - Index de ligne du modèle vers la vue
    - int convertRowIndexToView(int index)

RowSorter

- Méthodes de modification de la façon de trier :
  - Permet d'indiquer quelle colonne trier et dans quel ordre
    - get/setSortKeys(List<? extends SortKeys> keys)
  - Demande de changer l'ordre de trie d'une colonne
    - toggleSortOrder(int column)
- La classe RowSorter.SortKey(int column,SortOrder order) permet de définir la colonne et l'ordre de trie
- L'énumération **javax.swing.SortOrder** définie les ordres de trie (UNSORTED, ASCENDING, DESCENDING)

RowSorter

- Il est posible d'enregister des listeners
  - **add/removeRowSorterListener(RowSorterListener l)**
- Pour être averti d'un changement :
  - de l'ordre d'une colonne par l'utilisateur **(SORT_ORDER_CHANGED)**
  - Du trie (ou filtre) d'une la colonne **(SORTED)**
- L'interface **RowSorterListener** possède une méthode
  - **void sorterChanged(RowSorterEvent e)**
- RowSorterEvent indique le type d'évènement :
    - **Type.SORT_ORDER_CHANGED**
    - **Type.SORTED**

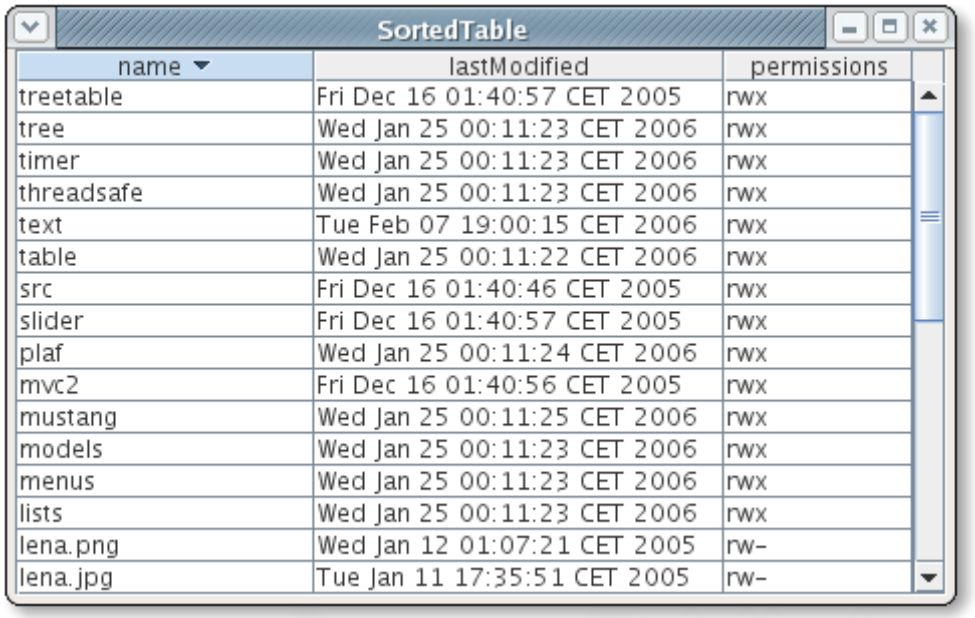
Exemple

- En utilisant un **TableRowSorter**, on utilise la méthode **setRowSorter()** sur la **JTable**.

```
FileTableModel model=new FileTableModel(new File("."));

TableRowSorter<FileTableModel> sorter=
  new TableRowSorter<FileTableModel>(model);

JTable table=new JTable(model);
table.setRowSorter(sorter);
```



DefaultRowSorter

- **javax.swing.DefaultRowSorter<M,I>**
  - I représente le type permettant d'identifier une ligne (JTable, JList: Integer, JTree: Object (node))
  - implantation par défaut d'un **RowSorter** utilisant un **Comparator** par colonne

- Il est possible de filtrer les lignes en utilisant un **RowFilter**
  - **setRowFilter(RowFilter<? super M,? super I> filter)**

DefaultRowSorter

- Permet de changer/demander le comparateur d'une colonne
  - **get/setComparator(Comparator<?> c,int column)**
    Par défaut, la comparaison se fait sur les représentations en String des objets (**toString**())
- Indique le nombre de clés pour trier utilisée par la méthode **toggleSortOrder(int column)**
  - **get/setMaxSortKeys()** [3 par défaut]
- Trie automatiquement lorsque le modèle est mis à jour
  - **get/setSortOnUpdate()** [attention couteux]
- Demande un trie/filtre manuel
  - **sort**()

RowFilter

- **javax.swing.RowFilter<M,I>** classe abstraite qui permet d'indiquer si oui ou non une ligne du modèle doit être incluse à l'affichage
  - **boolean include(Entry<? extends M,? extends I> entry)**

- **RowFilter.Entry**, correspond à une ligne du modèle :
  - l'identifiant d'une ligne : **I getIdentifier()**
  - Le nombre de colonne : **int getValueCount()**
  - l'objet pour une colonne donnée : **Object getValue(int index)**
  - La représentation de l'objet sous forme de String : **String getStringValue(int index)**

Filtres par défaut

- Il existe des filtres par défaut :
  - Tester par rapport à une expression régulière
    <M,I> RowFilter<M,I> regexFilter(String regex, int... indices)
    si les indices sont fournis alors seul les lignes correspondant aux indices sont prisent en compte
  - Tester si des valeurs sont **AFTER**, **BEFORE**, **EQUALS** ou **NOT_EQUALS** à un nombre
    <M,I> RowFilter<M,I>numberFilter(
    RowFilter.ComparisonType type,Number number, int... indices)
  - Même chose avec une date
    <M,I> RowFilter<M,I> dateFilter(
    RowFilter.ComparisonType type, Date date, int... indices)

Combinaison de filtres

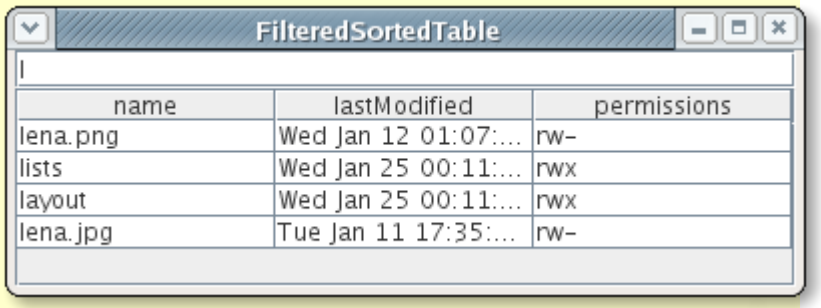
- Ainsi que des filtres qui permettent de faire des combinaisons de filtres
  - Inverser un filtre :
    <M,I> RowFilter<M,I> notFilter(RowFilter<M,I> filter)
  - ET généralisé sur des filtres :
    <M,I> RowFilter<M,I> andFilter(Iterable<? extends RowFilter<? super M,? super I>> filters)
  - OU généralisé sur des filtres :
    <M,I> RowFilter<M,I> orFilter(Iterable<? extends RowFilter<? super M,? super I>> filters)

Exemple de filtre

- Filtrer dynamiquement la table suivant le nom du fichier

```
final TableRowSorter<FileTableModel> sorter=
  new TableRowSorter<FileTableModel>(model);

final JTextField field=new JTextField();
field.addCaretListener(new CaretListener() {
  public void caretUpdate(CaretEvent e) {
    final String text=field.getText();
    if (text==null) {
      sorter.setRowFilter(null); // pas de filtre
      return;
    }
    sorter.setRowFilter(new RowFilter<FileTableModel,Integer>() {
      @Override
      public boolean include(Entry<? extends FileTableModel,? extends Integer> entry) {
        //File file=entry.getModel().getFile(entry.getIdentifier());
        //return file.getName().startsWith(text);
        return ((String)entry.getValue(0)).startsWith(text);
      }
    });
  }
});
```



TableRowSorter

- La classe **javax.swing.table.TableRowSorter** implante la classe **DefaultRowSorter<M extends TableModel,Integer>** spécifiquement pour les tables

- Les comparateurs utilisés sont :
  - Si un comparateur est spécifié on prend celui-ci, sinon
  - Si **getColumnClass**() est **String**, alors on compare les **String**
  - Si **getColumnClass**() est **Comparable** alors on utilise un **Comparator** qui délègue à **compareTo()**
  - Sinon on appel **toString**() sur l'objet et on compare les **String**

Attention avec la sélection

- Il y a un problème pour récupérer les lignes sélectionnées dans le cas où un **RowSorter** est installé
  - Les index des lignes affichées ne correspondent pas aux index des lignes du modèles

- Pour la sélection, **table.getSelectedRows()** indique les lignes de la table dans la vue et non par celle au niveau du modèle, il faut donc utiliser la méthode :
  - **convertRowIndexToModel(int rowInView)**

Exemple de sélection

- On peut demander à la vue de convertir les index

```
FileTableModel model=new FileTableModel(new File("."));

TableRowSorter<FileTableModel> sorter=
  new TableRowSorter<FileTableModel>(model);

JTable table=new JTable(model);
table.setRowSorter(sorter);

final Desktop desktop=Desktop.getDesktop();
Action print=new AbstractAction("Print") {
  public void actionPerformed(ActionEvent event) {
    int[] rows=table.getSelectedRows();
    for(int i=0;i<rows.length;i++) {
      rows[i]=table.convertRowIndexToModel(rows[i]);
    }
    for(int row:rows)
      try {
        desktop.print(model.getFile(row));
      } catch(IOException e) {
        e.printStackTrace(); // bof
      }
    }
  };
  print.setEnabled(desktop.isSupported(Desktop.Action.PRINT));
```