

Java Avancé - Cours 4

Architecture MVC, JList, Table

V. DESLANDRES, I. GUIDARA

veronique.deslandres@univ-lyon1.fr

Introduction

- Pour visualiser et manipuler un gros volume d'informations : composants spécifiques
- Les informations peuvent être présentées sous forme de **tableau**, de **liste**, **d'arbre** ou de **graphe**
- L'API Java Swing propose plusieurs composants pour visualiser les informations :
 - *JTable, JList, JTree, JGraph,...*

Architecture MVC

Modèle, Vue, Contrôle

MVC: Principes de base

- Le modèle d'architecture MVC (Model View Controller) est à la base de nombreux systèmes de visualisation graphiques
- Principe de Base: **séparation des rôles**
 - Le **modèle** est l'élément principal du composant, il contient les **données**
 - Les **vues** du composant sont **abonnées** au modèle : ce sont des **visualisations des données** (interfaces utilisateur)
 - Le **contrôleur** assure la synchronisation entre modèle et vues (**traitement**)
- La Java Swing repose sur l'architecture MVC

MVC exemple (JSlider)

- *Modèle* :

- valeur minimale = 0
- valeur courante = 15
- valeur maximale = 100



- *Vue* :



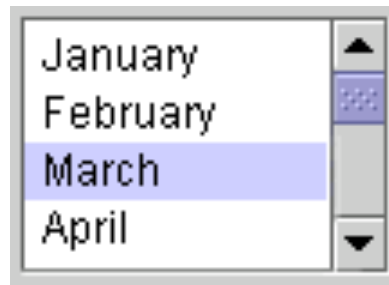
- *Contrôleur* :

- Traiter les clics de souris sur les boutons terminaux
- Gérer les *drags* de souris sur l'ascenseur

Modèles MVC des composants SWING

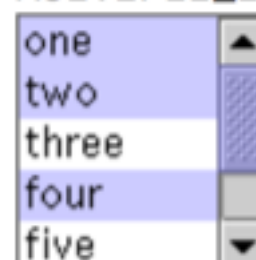
- La plupart des composants Swing (sauf les conteneurs) utilisent un ou plusieurs composants génériques comme MODELE des données
- `JList` :
 - classe **ListModel** pour les données
 - classe **ListSelectionModel** pour gérer les sélections
- `JTable` :
 - classe **TableModel** pour les données
 - classe **TableColumnModel** pour définir les colonnes
 - classe **ListSelectionModel** pour gérer les sélections

Le composant JList



Caractéristiques de base

- Une **JList** est une présentation des données sous forme de liste
- Permet l'affichage d'une liste d'items
 - Sans pouvoir les modifier
 - Ou avec possibilité de changement des items
- Choix d'une ou de plusieurs valeurs dans une liste prédéfinie
 - Remarque: si une seule valeur est affichée et sélectionnée, ComboBox préférable



Initialisation d'une liste

- Constructeur simple (avec modèle caché)

```
String[] couleurs = {"rouge", "bleu", "gris", "vert",  
                    "jaune", "noir", "orange", "blanc", "rose" };  
JList liste = new JList(couleurs) ;
```



- Forcer la sélection d'un élément dans une liste
 - Les indexes démarrent à 0
 - Exemple: sélection de l'élément de rang 2

```
liste.setSelectedIndex(2);
```

Affichage d'une liste

- Ajout d'une barre de défilement à une liste
 - Par défaut, la liste affichera 8 valeurs avec une présentation verticale
 - La barre de défilement n'apparaît pas si la liste comporte moins de valeurs

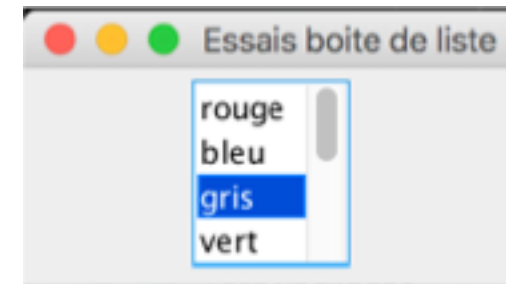
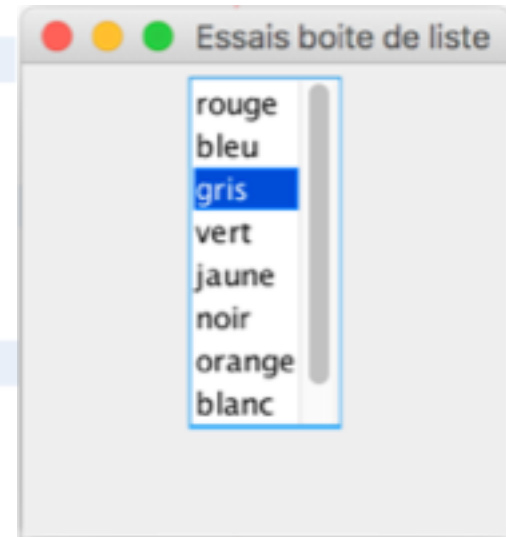
```
JScrollPane jsp = new JScrollPane(liste);  
getContentPane().add(jsp); // ajouter le jsp au content pane
```

OU

```
JScrollPane jsp = new JScrollPane();  
jsp.getViewPort().setView(liste);  
getContentPane().add(jsp); // ajouter le jsp au content pane
```

- Choisir le **nombre d'items** à afficher avec barre de défilement
 - Exemple: afficher seulement 4 valeurs à la fois

```
liste.setVisibleRowCount(4);
```



Mode d'affichage des items d'une liste

- Méthode: `liste.setLayoutOrientation(orientation);`
- 3 modes: VERTICAL, VERTICAL_WRAP, HORIZONTAL_WRAP

OU

```
liste.setLayoutOrientation(JList.VERTICAL);
```

```
liste.setLayoutOrientation(0);    (par défaut)
```

OU

```
liste.setLayoutOrientation(JList.VERTICAL_WRAP);
```

```
liste.setLayoutOrientation(1);
```

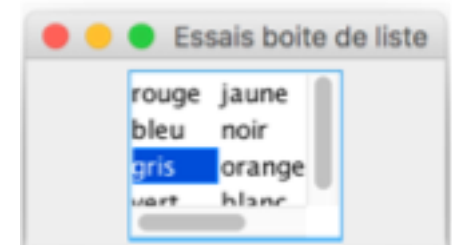
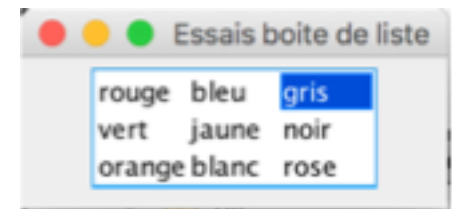
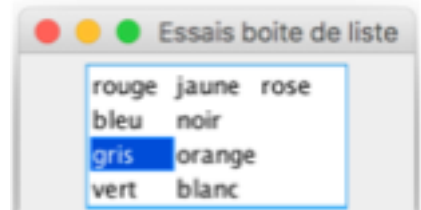
OU

```
liste.setLayoutOrientation(JList.HORIZONTAL_WRAP);
```

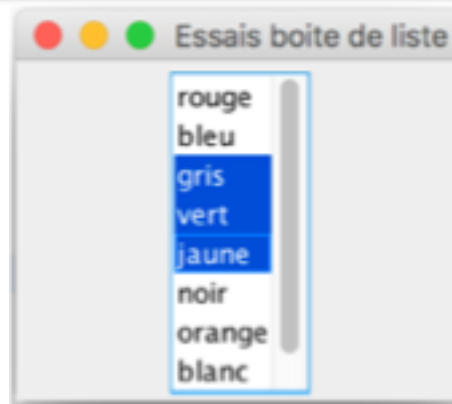
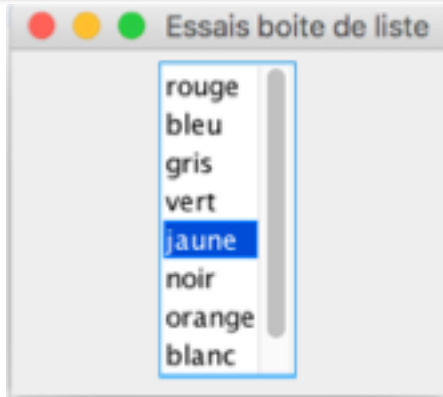
```
liste.setLayoutOrientation(2);
```

- Afficher avec une dimension et des barres de défilement

```
jsp.setPreferredSize(new Dimension(100, 80));
```



Modes de sélection des items d'une liste



- Méthode: `liste.setSelectionMode(mode);`
- 3 modes: SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, MULTIPLE_INTERVAL_SELECTION

OU `liste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);`
`liste.setSelectionMode(0);`

OU `liste.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);`
`liste.setSelectionMode(1);`

OU `liste.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);`
`liste.setSelectionMode(2);` (par défaut)

Accès aux informations de la liste

- Récupérer l'élément à la position i

*Les indices
commencent à 0*



```
//récupérer le modèle de la liste
ListModel myModel=liste.getModel();
String premierelement=myModel.getElementAt(0).toString();
System.out.println("Le premier élément est: "+premierelement);
```

- Récupérer tous les éléments d'une liste : boucle

```
ListModel myModel=liste.getModel();
int size = myModel.getSize();
for (int i = 0 ; i < size ; i++) {
    Object elem = myModel.getElementAt(i);
    System.out.println(elem);
}
```

Accès aux informations de la liste

- Liste à sélection simple : récupérer l'élément sélectionné

- Méthode: **Object** valeur=liste.getSelectedValue();

```
String ch = (String) liste.getSelectedValue();  
System.out.println("Action Liste - La valeur sélectionnée: "+ch) ;
```

- Listes à sélection multiple : récupérer tous les éléments sélectionnés

- Méthode: **List<type>** valeurs = liste.getSelectedValuesList();

```
System.out.println("Action Liste - Les valeurs selectionnees :");  
List<String> valeurs = liste.getSelectedValuesList();  
for (int i = 0; i<valeurs.size(); i++)  
    System.out.println(valeurs.get(i)) ;
```

Accès aux positions des items sélectionnés

- Liste à sélection simple : récupérer la position de la 1^{ère} valeur sélectionnée par l'utilisateur
 - Méthode: `public int getSelectedIndex();`

```
int index = liste.getSelectedIndex();  
System.out.println("Action Liste - Index de la valeur sélectionnée: "+index) ;
```

- Listes à sélection multiple : récupérer les positions de toutes les valeurs sélectionnées
 - Méthode: `public int[] getSelectedIndices();`

```
System.out.println("Action Liste - Les indexes des valeurs selectionnees :");  
int[] indexes = liste.getSelectedIndices();  
for (int i = 0; i<indexes.length; i++)  
    System.out.println(indexes[i]) ;
```

Événements générés

- (Une liste ne génère pas d'événement de type `ActionEvent`)
- Les événements générés par une liste sont des **événements de sélection**
 - de type : `ListSelectionEvent`
- Implémentation de l'interface: `ListSelectionListener`
- L'interface ne comporte qu'une seule méthode :
`public void valueChanged(ListSelectionEvent e)`

Événements de sélection

Méthodes de *ListSelectionEvent*

- `Object getSource()`: objet source de l'événement
- `int getFirstIndex()`: index du 1^{er} item dont la valeur de sélection a changé
- `int getLastIndex()`: index du dernier item dont la valeur de sélection a changé

Evénements générés

- `ListSelectionEvent` est généré :
 - Lors de l'appui sur le bouton de la souris
 - Lors du relâchement du bouton
- Les traitements **sont alors générés 2 fois**
- Pour pallier cette redondance
 - Méthode de l'événement



```
public boolean getValueIsAdjusting();
```

```
public void valueChanged (ListSelectionEvent e) {  
    if ( !e.getValueIsAdjusting() ) {  
        // accès aux informations sélectionnées et traitement  
    }  
}
```

```
import java.awt.* ;
import javax.swing.* ;
//import java.util.List ;
import javax.swing.event.* ; //Utile pour ListSelectionListener
```

```
class JListTest extends JFrame implements ListSelectionListener {
```

```
String[] couleurs = {"rouge", "bleu", "gris", "vert",
                    "jaune", "noir", "orange", "blanc", "rose" } ;
```

```
JList liste ;
```

```
public JListTest() {
```

```
    Container contenu = getContentPane() ;
```

```
    contenu.setLayout(new FlowLayout()) ;
```

```
    //Créer la liste
```

```
    liste = new JList(couleurs) ;
```

```
    //Sélectionner le 3ème élément
```

```
    liste.setSelectedIndex(2);
```

```
    //Afficher 4 valeurs
```

```
    liste.setVisibleRowCount(4);
```

```
    //Choisir le mode d'affichage des items de la liste
```

```
    liste.setLayoutOrientation(JList.VERTICAL);
```

```
    //Choisir le mode de sélection des items de la liste
```

```
    liste.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```

```
    //Ajouter la liste à la barre de défilement
```

```
    JScrollPane jsp = new JScrollPane(liste);
```

```
    //Ajouter le jsp au content pane
```

```
    contenu.add(jsp);
```

```
    //Modifier la taille du scrollPane
```

```
    jsp.setPreferredSize(new Dimension(100, 80));
```

```
    //Associer la liste au listener ListSelectionListener
```

```
    liste.addListSelectionListener(this);
```

```
} //Fin du constructeur
```

La fenêtre est son propre écouteur

Exemple de JList (statique) avec gestion des événements

Ajouter un écouteur
à la liste

Exemple JList statique (suite)

```
//Implémentation de la méthode valueChanged
public void valueChanged (ListSelectionEvent e) {
    if ( !e.getValueIsAdjusting()) {
        //Récupérer le modèle de la liste
        ListModel myModel=liste.getModel();
        //Afficher tous les éléments de la liste
        int size = myModel.getSize();
        for (int i = 0 ; i < size ; i++) {
            Object elem = myModel.getElementAt(i);
            System.out.println(elem);
        }
    }
}

public static void main (String args[]) {
    JListTest fen = new JListTest() ;
    fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fen.setVisible(true) ;
    fen.setTitle("Essais boîte de liste") ;
    fen.setSize(300, 210) ;
}
}
```

Listes dynamiques (modifiables)

- Quand on crée la liste en lui envoyant un vecteur d'objets, Java crée implicitement un `DefaultListModel` mais il est **non modifiable**
 - On ne peut ni **ajouter**, ni **supprimer** les items de la liste
- Quand on veut pouvoir modifier les items de la liste :
 - Il faut créer le modèle **explicitement** avec `DefaultListModel`

Listes dynamiques

- Création du modèle:

```
DefaultListModel monModele = new DefaultListModel ();
```

- Création de la liste:

```
JList liste = new JList(monModele);
```

- Ajout d'un élément à la fin de la liste:

```
monModele.addElement(element);
```

- Ajout d'un élément à la position i:

```
monModele.add(i, element);
```

- Supprimer un élément:

```
monModele.removeElement(element);
```

- Supprimer l'élément à la position i:

```
monModele.remove(i);
```

Exemple avec création de modèle

```
static JList liste;
static DefaultListModel monModele;
TestJListModel(){
    //Utilisation d'un modèle des données (par défaut)
    monModele = new DefaultListModel();
    //Construction de la liste
    monModele.addElement("rouge");
    monModele.addElement("gris");
    monModele.addElement("bleu");
    monModele.addElement("bleu");

    liste = new JList(monModele);

    Container contenu = getContentPane();
    contenu.setLayout(new FlowLayout());

    JScrollPane jsp = new JScrollPane(liste);
    contenu.add(jsp);
}

//Ajout d'un élément à une position donnée
monModele.add(1, "jaune");
//Ajout d'un élément à la fin de la liste
monModele.addElement("rose");
//Suppression d'un élément d'un index donné
monModele.remove(0);
//Supprimer l'élément sélectionné
int index = liste.getSelectedIndex();
monModele.remove(index);
//Suppression d'un item donné
//Supprimer la 1ère occurrence de « bleu » (boolean)
//Retourne vrai si « bleu » était un item de la liste, faux sinon
monModele.removeElement("bleu");
//Pour supprimer toutes les occurrences :
boolean suppr = false;
do
    suppr = monModele.removeElement("bleu");
while (suppr);
```

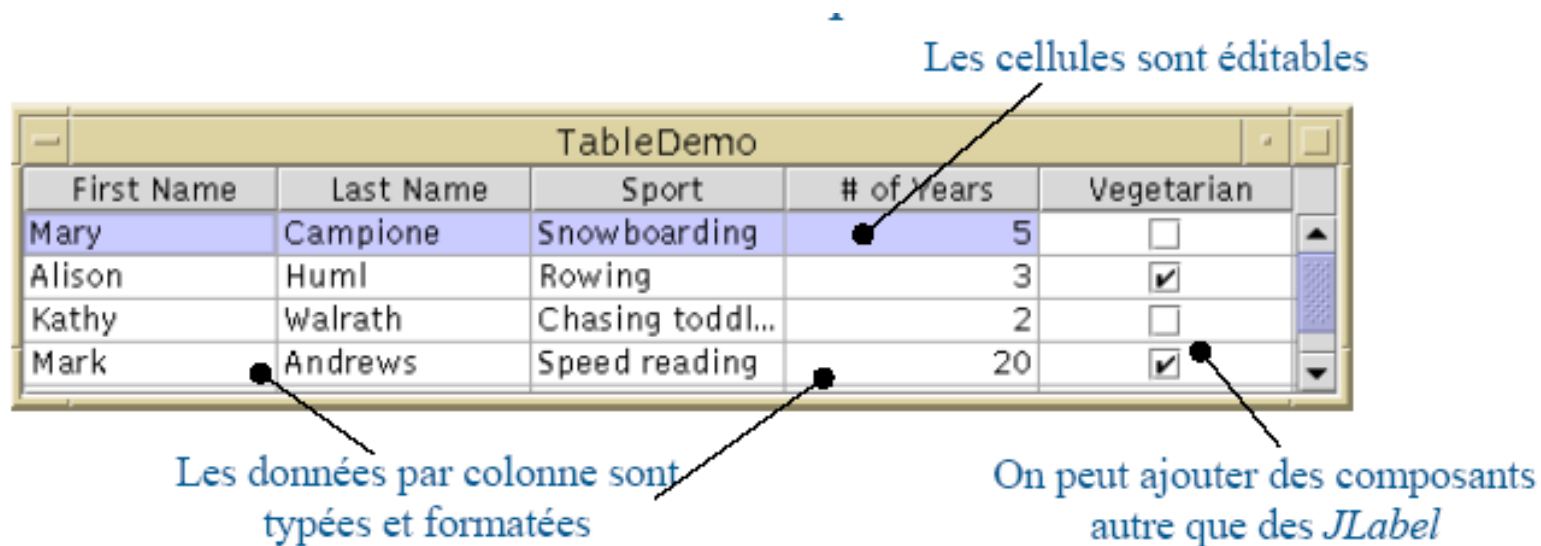
Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34\$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.com	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail.com	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.com	vbAf124%z	Feb 22, 2006

Le composant JTable

`package javax.swing.table`

Caractéristiques de base

- Une **JTable** présente les données sous forme de tableau
- Permet d'afficher, saisir et modifier des 'données ligne'
- Un **seul type de données par colonne**



Un package complet est dédié aux tables:

javax.swing.table

TableFTFEditDemo

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Ste	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Knitting	4	<input checked="" type="checkbox"/>
Sharon	Zakhour	Speed reading	20	<input checked="" type="checkbox"/>

TableRenderDemo

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	9	<input type="checkbox"/>
Alison	Huml	Snowboarding	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Rowing	2	<input type="checkbox"/>
Sharon	Zakhour	Knitting	20	<input checked="" type="checkbox"/>

Speed reading
Pool
None of the above

TableDialogEditDemo

First Name	Favorite Color	Sport	# of Years	Vegetarian
Mary		Snowboarding	5	<input type="checkbox"/>
Alison		Rowing	3	<input checked="" type="checkbox"/>
Kathy		Knitting	2	<input type="checkbox"/>
Sharon		Speed reading	20	<input checked="" type="checkbox"/>

<http://java.sun.com/docs/books/tutorial/uiswing/components/table.html>

Principes des composants JTable

- Une **JTable** possède 3 modèles :
 - un *modèle des données* de la table : instance de la classe **TableModel**, contient les données
 - un *modèle de colonnes* : classe **TableColumnModel**
 - un *mode de sélection* : classe **ListSelectionModel**
 - Simple, par intervalle continu, multiple
 - (C'est la même que pour les JList)
- Chaque élément peut **ne pas être défini**, il existe des éléments par défaut pour chaque modèle :
 - **DefaultTableModel**, **DefaultTableColumnModel**
 - etc.

Initialisation simple

- Le constructeur le plus utilisé est:

`JTable(Object[][] rowData, Object[] columnNames)`

– avec les *valeurs* des cellules `rowData` et les *noms* de colonnes `columnNames`

```
String[] nomsColonnes = {"First Name", "Last Name", "Sport", "# of Years", "Vegetarian"};
Object[][] donnees = {
    {"Mary", "Campione", "Snowboarding", new Integer(5), new Boolean(false)},
    {"Alison", "Huml", "Rowing", new Integer(3), new Boolean(true)},
    {"Kathy", "Walrath", "Knitting", new Integer(2), new Boolean(false)},
    {"Sharon", "Zakhour", "Speed reading", new Integer(20), new Boolean(true)},
    {"Philip", "Milne", "Pool", new Integer(10), new Boolean(false)}
};
JTable table = new JTable(donnees, nomsColonnes);
```

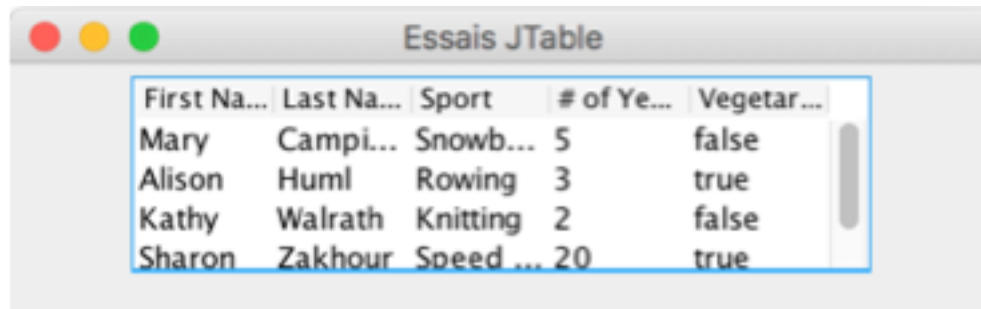
Affichage d'un tableau

- Ajout de la table à la fenêtre avec `JScrollPane`
- Attention: si on n'utilise pas de `JScrollPane`, les titres des colonnes ne seront pas affichés, il faut le faire soi-même

```
JScrollPane jsp = new JScrollPane(table);  
jsp.getContentPane().add(table);
```

- Ajout d'une barre de défilement à une table : redimensionner le `JScrollPane`

```
jsp.setPreferredSize(new Dimension(100, 80));
```



First Na...	Last Na...	Sport	# of Ye...	Vegetar...
Mary	Campi...	Snowb...	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Knitting	2	false
Sharon	Zakhour	Speed ...	20	true

Modes d'affichage des lignes d'un tableau

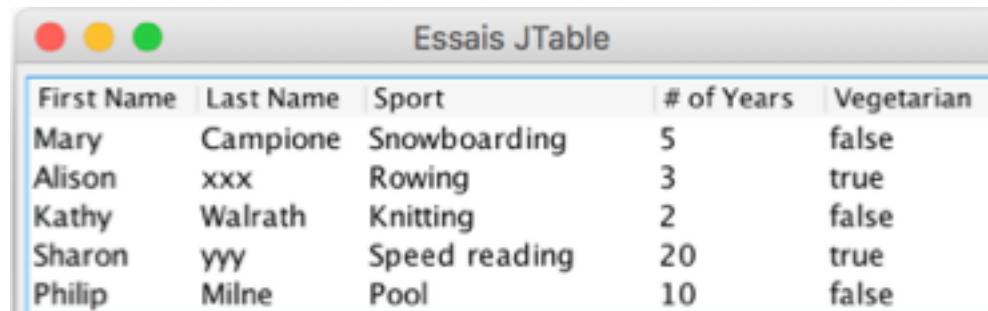
- Changement de la largeur d'une colonne
 - Par défaut les colonnes ont la même largeur
 - Méthode: `public void setPreferredWidth(int larg)`
 - Exemple: modification de la largeur de la 3^{ème} colonne

```
import javax.swing.table.TableColumn;
```

Importer la classe **TableColumn**
du package `table`

```
TableColumn column = null;  
for (int i = 0; i < 5; i++) {  
    column = table.getColumnModel().getColumn(i);  
    if (i == 2) { column.setPreferredWidth(100);} //colonne sport  
    else { column.setPreferredWidth(50); }  
}
```

Récupérer la colonne `i` du
modèle des colonnes de la table



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	false
Alison	xxx	Rowing	3	true
Kathy	Walrath	Knitting	2	false
Sharon	yyy	Speed reading	20	true
Philip	Milne	Pool	10	false

Auto-redimensionnement des colonnes

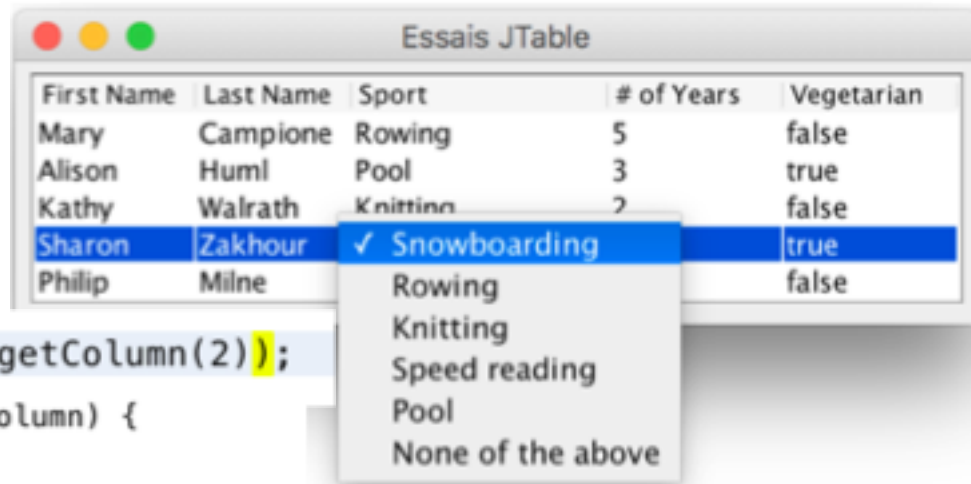
- Auto-redimensionnement en cas de modification de la taille des colonnes ou de la fenêtre
 - Méthode: `public void setAutoResizeMode(int mode)`

```
//ne pas ajuster automatiquement la largeur des colonnes,  
//utiliser une barre de défilement horizontale à la place  
table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);  
//lorsqu'une colonne est ajustée dans l'interface utilisateur,  
//redimensionner la colonne suivante dans le sens opposé  
table.setAutoResizeMode(JTable.AUTO_RESIZE_NEXT_COLUMN);  
//lorsqu'une colonne est ajustée dans l'interface utilisateur,  
//redimensionner les colonnes suivantes pour conserver la largeur totale;  
//Ceci est le comportement par défaut  
table.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);  
//redimensionner la dernière colonne seulement  
//pendant les opérations de redimensionnement  
table.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);  
//redimensionner proportionnellement toutes les colonnes  
//pendant toutes les opérations de redimensionnement  
table.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
```

Editeur de cellules

- Chaque cellule peut avoir une façon différente d'être éditée (affichée, modifiée) :
 - Éditeur de cellule (classe `TableCellEditor`)
 - On peut en définir un seul pour la table, ou un par colonne

- Exemple: utilisation d'une Combo Box



```
setUpSportColumn(table.getColumnModel().getColumn(2));
```

```
public void setUpSportColumn(TableColumn sportColumn) {  
    // fixer l'éditeur pour la colonne sport  
    JComboBox comboBox = new JComboBox();  
    comboBox.addItem("Snowboarding");  
    comboBox.addItem("Rowing");  
    comboBox.addItem("Knitting");  
    comboBox.addItem("Speed reading");  
    comboBox.addItem("Pool");  
    comboBox.addItem("None of the above");  
    sportColumn.setCellEditor(new DefaultCellEditor(comboBox));  
} // Fin setUpSportColumn
```


Rendu de cellules

- On peut définir une visualisation différente des cellules ou des colonnes (texte, case à cocher, couleur de fond, etc.)
 - C'est le « rendu »: classe `TableCellRenderer`
- Le « renderer » définit les caractéristiques d'affichage des éléments de la colonne

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Rowing	5	false
Alison	Huml	Pool	3	true
Kathy	Walrath	Knitting	20	false
Sharon	Zakhour	Snowboarding	20	true
Philip	Milne	Pool	10	false

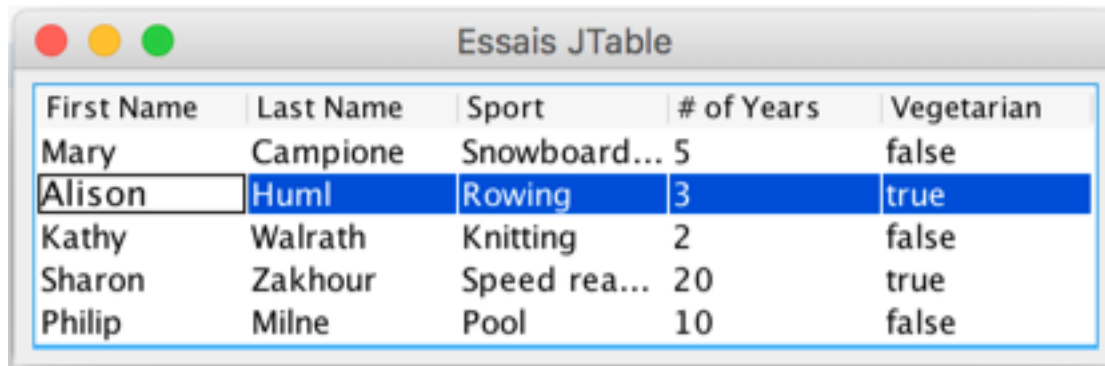
- Exemple: ajouter une bulle d'information

```
import javax.swing.table.DefaultTableCellRenderer;
```

```
//Bulle d'information : utilisation d'un "renderer"  
DefaultTableCellRenderer renderer =  
    new DefaultTableCellRenderer();  
renderer.setToolTipText("Click for combo box");  
sportColumn.setCellRenderer(renderer);
```

Modes de sélection des lignes d'un tableau

- Méthode: `table.setSelectionMode(mode)` ;
- 3 modes comme pour JList



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboard...	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Knitting	2	false
Sharon	Zakhour	Speed rea...	20	true
Philip	Milne	Pool	10	false

```
table.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
```

```
table.setSelectionMode(ListSelectionMode.MULTIPLE_INTERVAL_SELECTION);
```

```
table.setSelectionMode(ListSelectionMode.SINGLE_INTERVAL_SELECTION);
```

Accès aux informations de la table

- `int getColumnCount()` : retourne le **nombre** de colonnes
- `int getRowCount()` : retourne le **nombre** de lignes
- `int getSelectedColumn()` : retourne **l'indice** de la colonne sélectionnée
- `int getSelectedRow()` : retourne **l'indice** de la ligne sélectionnée
- `String getColumnName(int col)` : retourne **le nom** de la colonne `col`
- `Object getValueAt(int numLigne, int numColonne)` : retourne **l'objet** placé dans la cellule de position `(numLigne, numColonne)`

Récupérer les données d'une table

- Chaque table possède un **modèle** qui contient ses données
 - Une instance de l'interface `TableModel`
- Obtention du modèle d'une table:
 - Méthode `getModel()` : retourne une `TableModel`
 - Exemple: Afficher tous les éléments d'une table

```
import javax.swing.table.TableModel;
```

Importer `TableModel`

```
void printTableData(JTable table) {  
    int numRows = table.getRowCount();  
    int numCols = table.getColumnCount();  
    TableModel model = table.getModel();  
  
    System.out.println("Valeur des données: ");  
    for (int i=0; i < numRows; i++) {  
        System.out.print("    ligne " + i + " :");  
        for (int j=0; j < numCols; j++) {  
            System.out.print(" " + model.getValueAt(i, j));  
        }  
        System.out.println();  
    }  
}
```

Événements générés et écouteur

- Les événements générés par une table sont des événements de type : `TableModelEvent`
- Implémentation de l'interface: **TableModelListener**
- L'interface ne comporte qu'une seule méthode:
`public void tableChanged(TableModelEvent e)`

Méthodes des événements d'une table

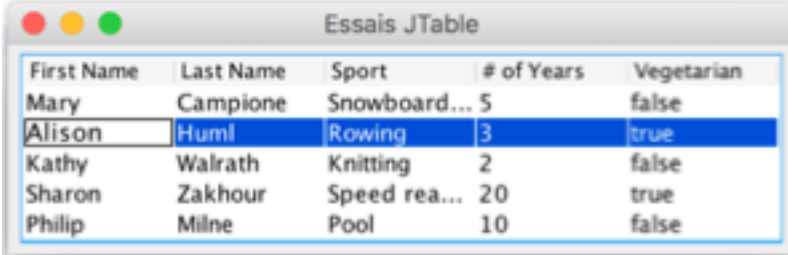
TableModelEvent

- **Object** `getSource()` : objet source de l'événement
- **int** `getFirstRow()` : index de la première ligne qui a changé
- **int** `getLastRow()` : index de la dernière ligne qui a changé
- **int** `getColumn()` : index de la colonne qui a changé

Exemple complet de JTable (statique)

```
import java.awt.* ;
import javax.swing.* ;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableColumn;
import javax.swing.table.TableModel;

class JTableTest extends JFrame {
    static JTable table;
    String[] nomsColonnes = {"First Name", "Last Name", "Sport", "# of Years", "Vegetarian"};
    Object[][] donnees = {
        {"Mary", "Campione", "Snowboarding", new Integer(5), new Boolean(false)},
        {"Alison", "Huml", "Rowing", new Integer(3), new Boolean(true)},
        {"Kathy", "Walrath", "Knitting", new Integer(2), new Boolean(false)},
        {"Sharon", "Zakhour", "Speed reading", new Integer(20), new Boolean(true)},
        {"Philip", "Milne", "Pool", new Integer(10), new Boolean(false)}
    };
    public JTableTest() {
        Container contenu = getContentPane() ;
        contenu.setLayout (new FlowLayout());
        //Cr er la table
        table = new JTable(donnees, nomsColonnes);
        //Associer la table   la JScrollPane
        JScrollPane jsp = new JScrollPane(table);
        jsp.setPreferredSize(new Dimension(400, 100));
        contenu.add(jsp);
        //Choisir le mode d'affichage
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        //Modifier la taille de la 3 me colonne
        TableColumn column = null;
        for (int i = 0; i < 5; i++) {
            column = table.getColumnModel().getColumn(i);
            if (i == 2) { column.setPreferredWidth(100);}
            else { column.setPreferredWidth(50); }
        }
        //Choisir le mode de s lection
        table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        //Editeur de cellules
        setUpSportColumn(table.getColumnModel().getColumn(2));
    } // fin du constructeur
}
```



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboard...	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Knitting	2	false
Sharon	Zakhour	Speed rea...	20	true
Philip	Milne	Pool	10	false

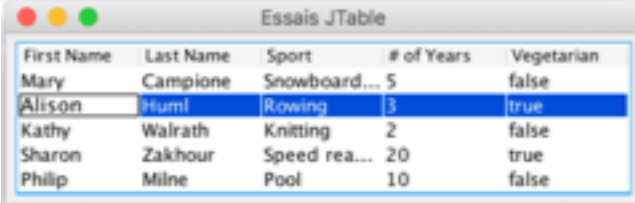
Exemple jTable statique (suite)

```
public static void main (String args[]) {
    JTableTest fen = new JTableTest();
    fen.setTitle("Essai jTable");
    fen.setSize(500, 300);
    fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fen.setVisible(true);
    printTableData(table);
}

public void setUpSportColumn(TableColumn sportColumn) {
    // fixer l'éditeur pour la colonne sport
    JComboBox comboBox = new JComboBox();
    comboBox.addItem("Snowboarding");
    comboBox.addItem("Rowing");
    comboBox.addItem("Knitting");
    comboBox.addItem("Speed reading");
    comboBox.addItem("Pool");
    comboBox.addItem("None of the above");
    sportColumn.setCellEditor(new DefaultCellEditor(comboBox));

    //Bulle d'information : utilisation d'un "renderer"
    DefaultTableCellRenderer renderer =
        new DefaultTableCellRenderer();
    renderer.setToolTipText("Click for combo box");
    sportColumn.setCellRenderer(renderer);
} // Fin setUpSportColumn

static void printTableData(JTable table) {
    //Afficher toutes les données de la table
    //Récupérer le modèle de la table
    TableModel model = table.getModel();
    int numRows = table.getRowCount();
    int numCols = table.getColumnCount();
    System.out.println("Valeur des données: ");
    for (int i=0; i < numRows; i++) {
        System.out.print("    ligne " + i + " :");
        for (int j=0; j < numCols; j++) {
            System.out.print(" " + model.getValueAt(i, j));
        }
        System.out.println();
    }
}
}
```



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboard...	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Knitting	2	false
Sharon	Zakhour	Speed rea...	20	true
Philip	Milne	Pool	10	false

Caractéristiques d'une *JTable* sans modèle défini

Constructeurs

`JTable`(Object[][] **mesDonnees**, Object[] **mesColonnes**)

`JTable`(Vector<T> **mesDonnees**, Vector<String> **mesColonnes**)

Caractéristiques

- *Toutes* les cellules sont **éditables (modifiables)** mais pas d'ajout de ligne ou colonne possible
- Rendu *de base* : les données sont considérées comme des **String** sauf les booléens : apparaissent avec une case à cocher
- Toutes les données des tables doivent être placées dans des tableaux ou des conteneurs
 - inapproprié pour certaines données (ex. données de tables relationnelles)

Si on souhaite mieux → Créer un modèle de table

Tables dynamiques (modifiables)

- Création du modèle:

```
DefaultTableModel monModele = new DefaultTableModel();
```

- Création de la table:

```
JTable table = new JTable(monModele);
```

- Ajout d'une ligne à la fin de la table:

```
monModele.addRow(Object[] rowData);
```

- Ajout d'une ligne à la position i:

```
monModele.insertRow(int position, Object[] rowData);
```

- Supprimer une ligne du tableau:

```
monModele.removeRow(int row);
```

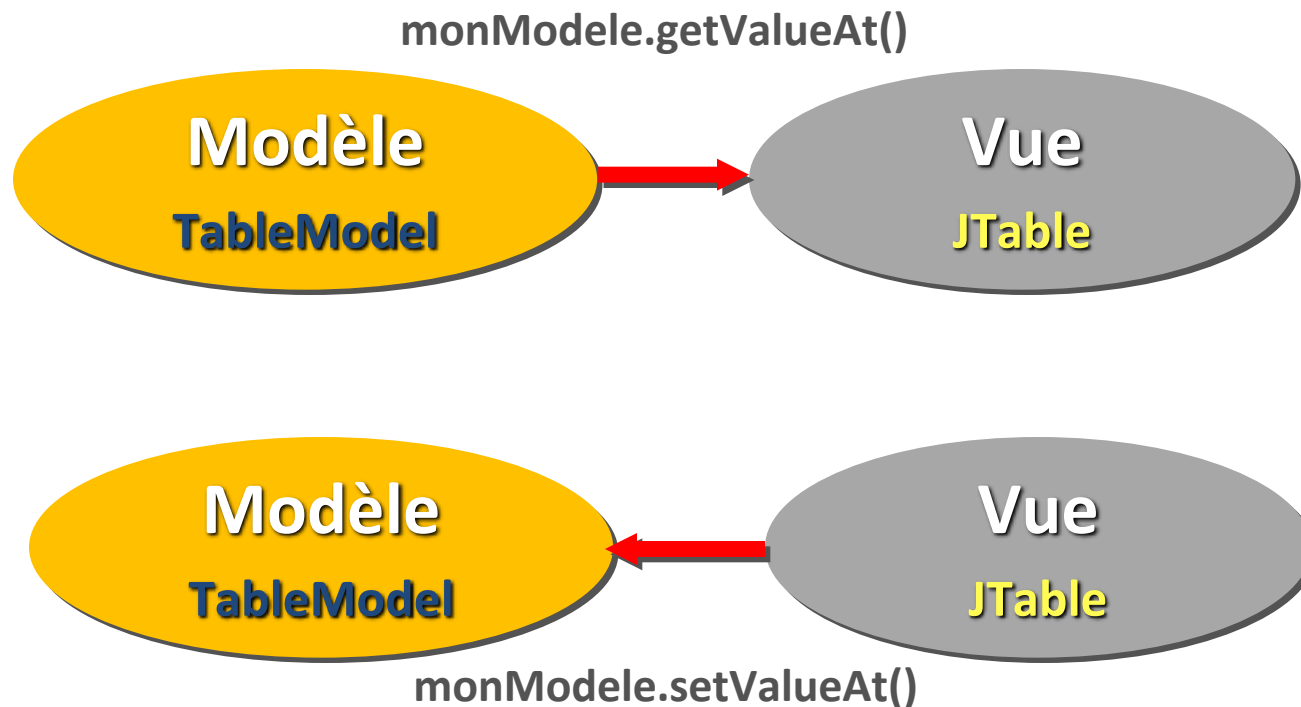
- Modifier le contenu de la cellule du tableau à la position (row, col):

```
monModele.setValueAt(Object value, int row, int col);
```

SetValueAt() de *TableModel*

- Est appelée **automatiquement** à chaque modification de la JTable
 - Pas besoin de gérer les évts avec des écouteurs
- Elle explicite comment on « enregistre » l'objet de la **vue dans le modèle**
- Prend 3 paramètres : un Objet, une ligne, une colonne
- Nécessite ensuite l'appel à une méthode **fireXXX()**
 - *par ex. fireTableCellUpdated(ligne, colonne)*
 - *pour **propager à toutes les vues***
 - *si on ne le fait pas, la vue reprend les anciennes valeurs après un rafraichissement écran !*

Lien Vue / modèle pour une JTable



(appel **automatique** de ces 2 méthodes à chaque modification de la vue)

Rendre des colonnes non modifiables

- Par défaut, les colonnes sont **toutes éditables**
- Sous l'IDE, on peut facilement dire quelles colonnes sont éditables ou non (en mode Design)
- Sans IDE, on utilise la méthode de **TableModel** :

boolean `isCellEditable`(int rowIndex, int colIndex)

```
DefaultTableModel model = new DefaultTableModel() {  
    boolean[] canEdit = new boolean[] {  
        false, false, true, false, false, false, false, true};  
  
    public boolean isCellEditable(int rowIndex, int colIndex) {  
        return canEdit[colIndex];  
    }  
};
```

Ici seules les cellules des 3^e et dernière colonnes sont modifiables, les autres non

Propagation des modifications dynamiques

- Càd. en fonction de la ligne sélectionnée
- On peut par ex. ajouter au modèle les 2 méthodes :
 - `addRow(Object p, int pRow)` : ajoute l'objet `p` à la ligne `pRow`
 - `removeRow(int pRow)` : supprime la ligne `pRow`
- L'action va se faire en fonction de la ligne en cours de sélection
 - Ajout : ligne suivante de celle sélectionnée ou dernière ligne
 - Suppression : ligne en cours de sélection
- A chaque ajout ou suppression d'une ligne il faut notifier les vues des modifications apportées au modèle
 - Avec `monModeleTable.fireTableStructureChanged()` ;

Avec l'IDE, cette méthode est automatiquement gérée

Gérer explicitement les événements de JTable

- On peut aussi vouloir **gérer soi-même et de façon globale** les événements de la table
 - Rappel : par défaut, ils sont gérés par les évts sur le *modèle de la table* et sur le *modèle de colonne*
 - Toute modification de l'utilisateur dans la table fait appel à `leModele.setValueAt()`
 - C'est le transfert Vue → Modèle

Exemple de table dynamique

```
import java.awt.* ;
import javax.swing.* ;
import javax.swing.event.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
```

```
class TestJTableModelEvent extends JFrame implements TableModelListener{
    JTable table;
    DefaultTableModel myModel;
```

```
public TestJTableModelEvent() {
    DefaultTableModel myModel=new DefaultTableModel(new Object[][] {},
        new String [] {"First Name","Last Name", "Sport", "# of Years", "Vegetarian"});
    table=new JTable(myModel);

    Container contenu = getContentPane() ;
    contenu.setLayout (new FlowLayout());
    //Associer la table à la JScrollPane
    JScrollPane jsp = new JScrollPane(table);
    jsp.setPreferredSize(new Dimension(400, 100));
    contenu.add(jsp);
    //Choisir le mode d'affichage
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    //Ajouter les lignes du tableau
    myModel.addRow(new Object[]{"Mary", "Campione", "Snowboarding", new Integer(5), new Boolean(false)});
    myModel.addRow(new Object[]{"Alison", "Huml", "Rowing", new Integer(3), new Boolean(true)});
    //myModel.addRow(new Object[]{"Kathy", "Walrath", "Knitting", new Integer(2), new Boolean(false)});
    myModel.addRow(new Object[]{"Sharon", "Zakhour", "Speed reading", new Integer(20), new Boolean(true)});
    myModel.addRow(new Object[]{"Philip", "Milne", "Pool", new Integer(10), new Boolean(false)});
    //Utilisation de la méthode insertRow pour ajouter la 3ème ligne
    myModel.insertRow(2, new Object[]{"Kathy", "Walrath", "Knitting", new Integer(2), new Boolean(false)});
    //Modifier le nom de la personne à la 2ème ligne et 1ère colonne
    myModel.setValueAt("Katthy", 2, 0);

    //Associer le modèle de la table au listener
    myModel.addTableModelListener(this);
} // fin du constructeur
```

Rappel pour une table statique :

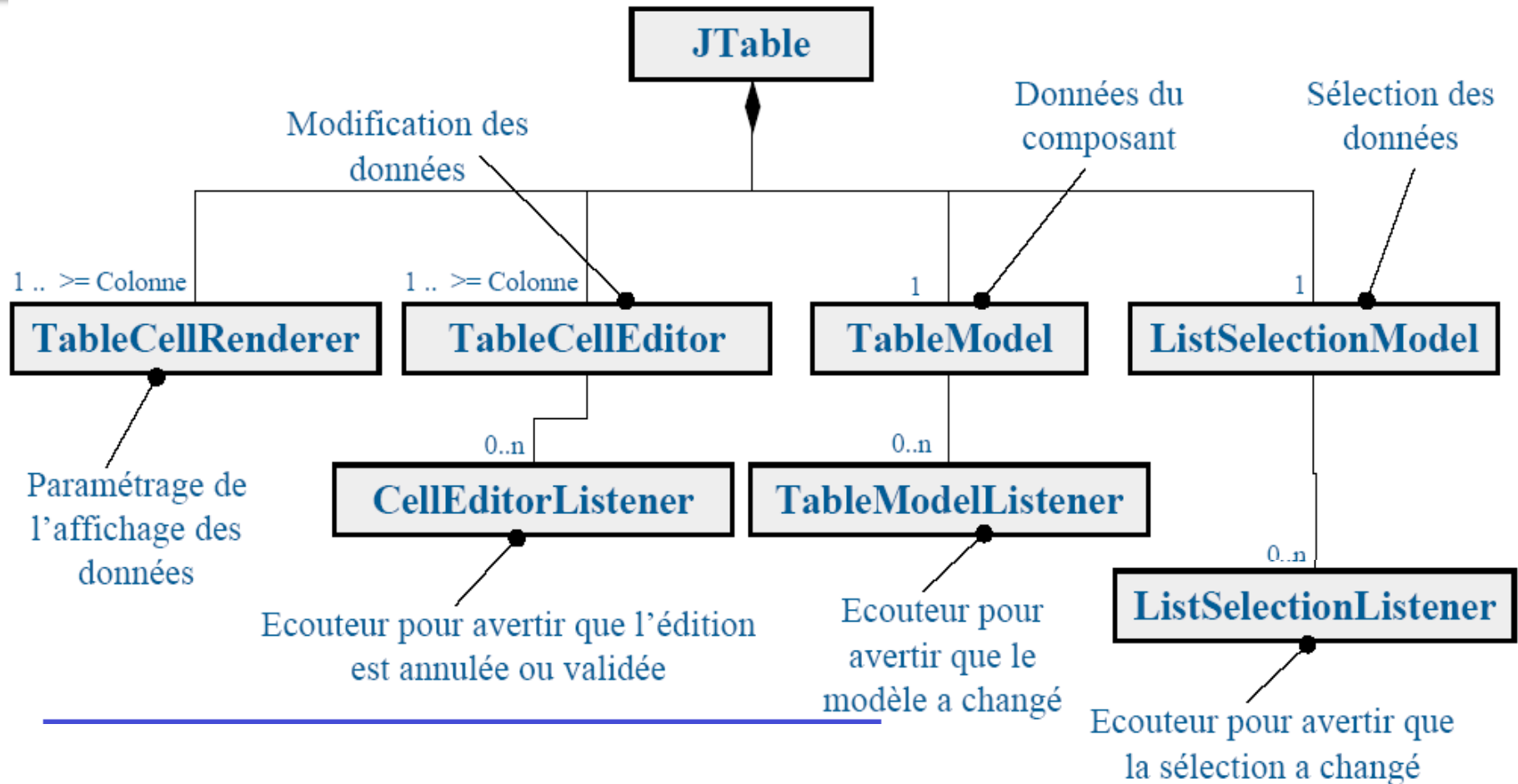
```
table = new JTable(donnees, nomsColonnes);
```


Exemple table dynamique (suite)

```
public static void main (String args[]) {
    TestJTableModelEvent fen = new TestJTableModelEvent() ;
    fen.setTitle("Essai JTable") ;
    fen.setSize(500, 300) ;
    fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fen.setVisible(true);
}

public void tableChanged( TableModelEvent e ) {
    int row = e.getFirstRow();
    int column = e.getColumn();
    TableModel model = (TableModel) e.getSource();
    String columnName = model.getColumnName(column);
    Object data = model.getValueAt(row, column);
    System.out.println("Modification : ");
    System.out.println("ligne : " +row + " Colonne : " +column +
        " Nom colonne : " +columnName + " Nouvelle valeur : " + data);
    System.out.println("-----");
}
}
```

En résumé : JTable



Grâce au pattern MVC de Java, il est possible de « brancher » un modèle existant à une autre vue (*JList* par exemple)

Composants gérés par défaut

- Selon le type de données se trouvant dans le modèle (indiqué par *getColumnClass()*) les objets « Editor » et « Renderer » retournent des composants prédéfinis
- Composant retourné par l'objet « Renderer » :
 - *Boolean* : *JCheckBox*
 - *Number, Double* et *Float* : *JLabel* aligné à droite
 - *ImageIcon* : *JLabel* aligné au centre
- Composant retourné par l'objet « Editor » :
 - *Boolean* : *JCheckBox*
 - Autre : *JTextField*

JTable : je retiens

- Les **composants** qu'on peut définir sur une *JTable*
 - *Modèle des données, de sélection, de colonnes, éditeur, renderer, etc.*
- Les **caractéristiques** d'une *JTable* sans modèle défini
- Ce que retourne *uneTable.getModel()*
- Le mécanisme de **synchronisation** Vue / Modèle avec une *JTable*
- Comment récupérer **la ou les ligne(s) sélectionnée(s)** d'une table
- Les méthodes de *propagation des modifications* effectuées sur une *JTable* dans *setValeurAt()*
 - (**firexxx**, même utilisées automatiquement par l'IDE)



Autres Références

- **JList** tutorial
- <http://fr.slideshare.net/martyhall/java-7-programming-tutorial-advanced-swing-and-mvc-custom-data-models-and-cell-renderers>
- **JTable** : tutoriel très complet d'Oracle
 - <http://docs.oracle.com/javase/tutorial/uiswing/components/table.html>