



# Programmation d'IHM - Cours 5

## Le composant JTable

V. DESLANDRES

[veronique.deslandres@univ-lyon1.fr](mailto:veronique.deslandres@univ-lyon1.fr)

# Sommaire de ce cours

- Caractéristiques de base d'une *JTable*
  - Taille des colonnes, redimensionnement : [7](#)
  - Éditeur et rendu de colonnes : [10](#)
- Récupérer les données d'une table : [14](#)
- Exemple d'une table statique (pas d'ajout/suppr de ligne) : [17](#)
- MVC du composant *JTable* : [21](#)
- Créer un modèle de table : [22](#)
- Exemple d'une table dynamique : [25](#)
- Résumé des composants d'une *JTable* : [28](#)
- **Je retiens** : [30](#)
- Exemple d'ajout de ligne pour une *JTable* avec *AbstractTableModel* : [32](#)

# Caractéristiques de base

- Une **JTable** présente les données sous forme de tableau
- Les cellules *ne sont pas* des composants indépendants
- Permet d'afficher, saisir et modifier des '**données ligne**'
- Un **seul type de données par colonne**

Les cellules sont éditables

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>

Les données par colonne sont typées et formatées

On peut ajouter des composants autre que des JLabel

# Un package complet est dédié aux tables: `javax.swing.table`

Ici, édition texte

The screenshot shows a window titled "TableFTFEditDemo" containing a table with the following data:

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"/>

The screenshot shows a window titled "TableRenderDemo" containing a table with a dropdown menu open for the "Sport" column of the first row. The dropdown menu lists the following options:

- Snowboarding
- Rowing
- Knitting
- Speed reading
- Pool
- None of the above

The table data is as follows:

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"/>

édition par comboBox

édition par un  
ColorChooser

The screenshot shows a window titled "TableDialogEditDemo" containing a table with color choosers in the "Favorite Color" column. The table data is as follows:

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
    - (le 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

- Un constructeur possible 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`

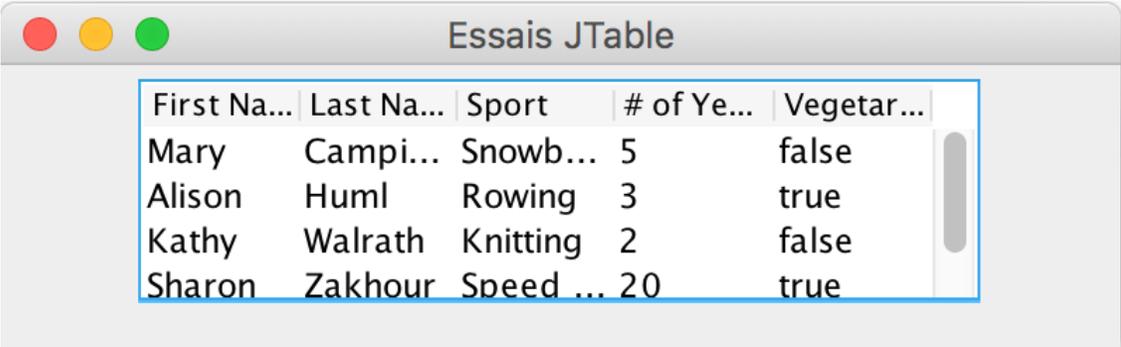
```
JScrollPane jsp = new JScrollPane(table);  
jsp.getContentPane().add(table);
```

- Attention: si on n'utilise pas de `JScrollPane`, **les titres des colonnes ne seront pas affichés**, il faudra le faire soi-même :

```
pan.setLayout(new BorderLayout());  
pan.add(table.getTableHeader(), BorderLayout.NORTH);  
pan.add(table, BorderLayout.CENTER);
```

- Ajout d'une barre de défilement à une table : on peut 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

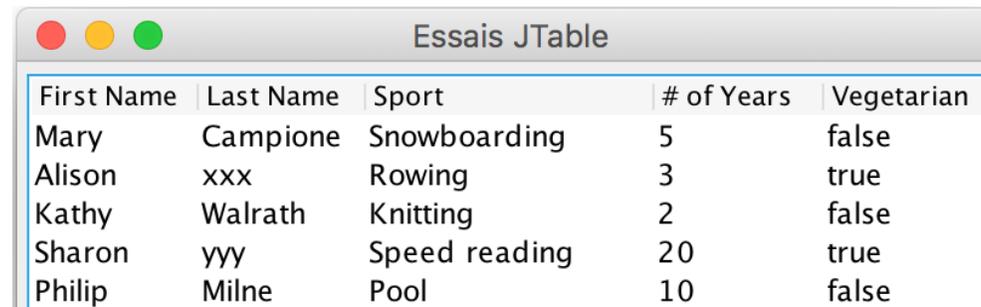
- Par défaut les colonnes ont la **même largeur**
- Changement de la largeur d'une colonne
  - Méthode de **TableColumn** : `public void setPreferredWidth(int larg)`
  - Ex. : ici la 3<sup>ème</sup> colonne fait le double des autres colonnes ; on balaie toutes les colonnes et on fixe la largeur à 50 pour toutes sauf la 3<sup>e</sup> colonne (100)

```
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 JTable **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);
```

Par défaut

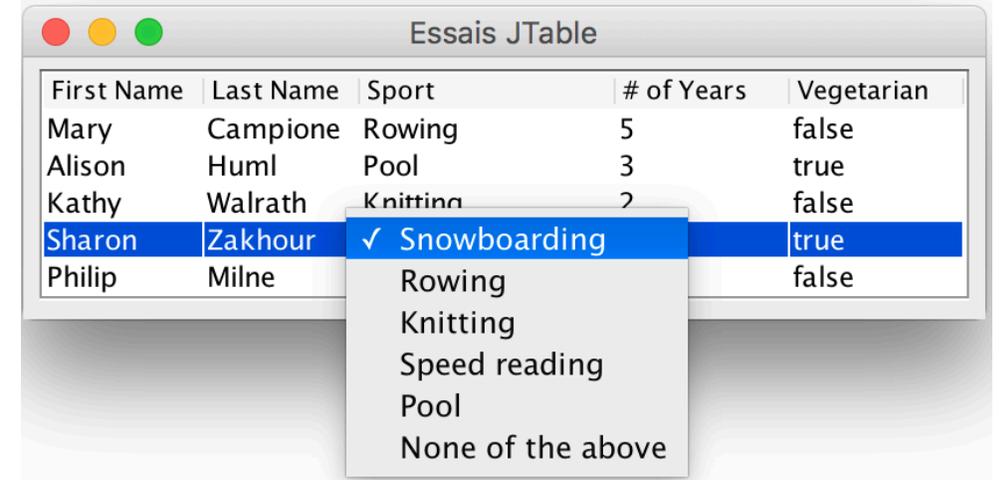
# Editeur de colonne

Chaque colonne peut avoir une façon différente d'être modifiée :

- Éditeur de cellule (classe `TableCellEditor`)
- On peut en définir un seul pour la table, ou un par colonne, avec `setCellEditor()`
- Exemple: utilisation d'une Combo Box

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

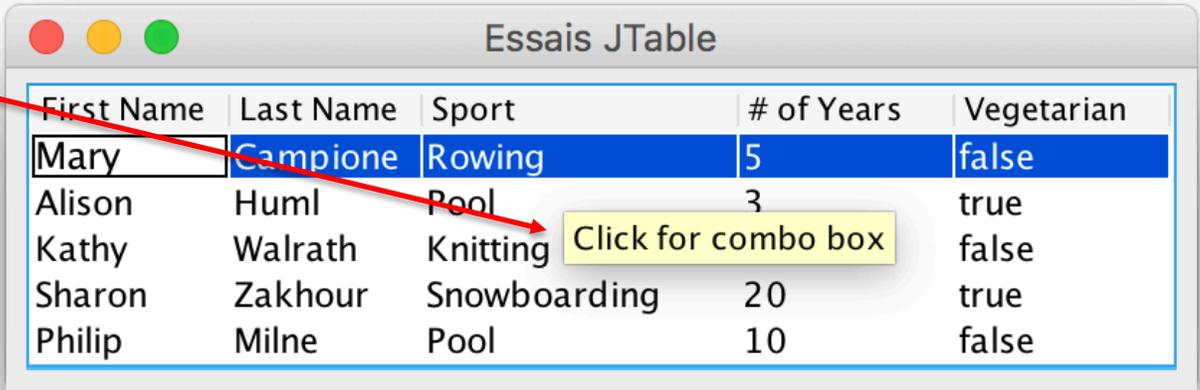
```
public void setUpSportColumn(TableColumn sportColumn) {  
    // fixer l'éditeur pour la colonne sport  
    JComboBox comboBox = new JComboBox();  
    comboBox.addItem("Snowboarding");  
    comboBox.addItem("Rowing");  
    comboBox.addItem("Knitting");  
    comboBox.addItem("Speed reading");  
    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 barré, 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
  - Exemple: ajouter une **bulle d'information**

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

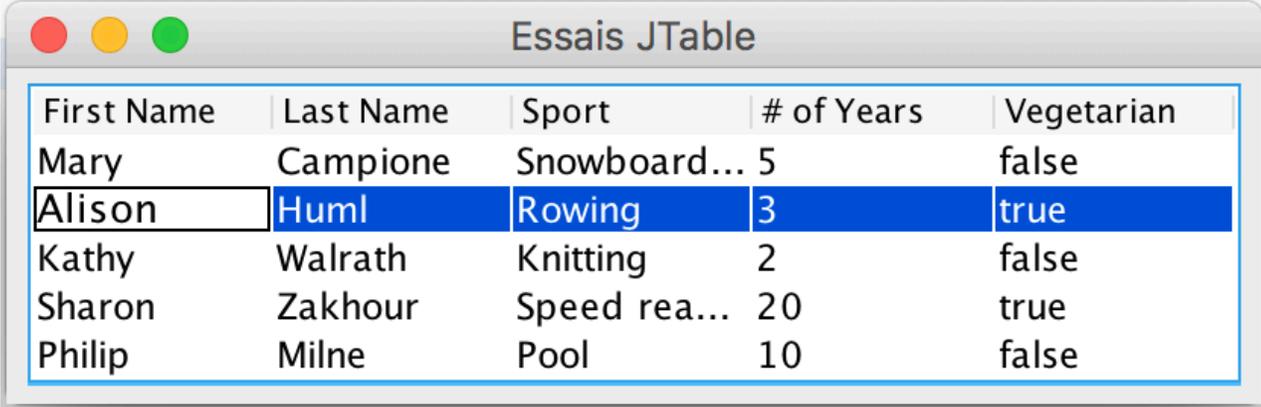


First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Rowing	5	false
Alison	Huml	Pool	3	true
Kathy	Walrath	Knitting		false
Sharon	Zakhour	Snowboarding	20	true
Philip	Milne	Pool	10	false

```
//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 une 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(ListSelectionModel.SINGLE_SELECTION);
```

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

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

# Accès aux informations de la table

## Méthodes de JTable

- `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 en position **visuelle** (`numLigne`, `numColonne`)

# Récupérer les données d'une table

- Même si on en n'a pas créé un, une table possède un **modèle** qui contient ses données
  - C'est une classe qui implémente l'interface `TableModel`
- Obtention du modèle d'une table:
  - Méthode `getModel()` : retourne un `TableModel`
  - Exemple : afficher tous les éléments d'une table

```
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();  
    }  
}
```

`import javax.swing.table.TableModel;`

# Événements générés et écouteur

- Les événements générés par une table sont des événements de type : **TableModelEvent**
- Interface écouteur : **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 généré un événement
- **int** `getLastRow ()` : index de la dernière ligne qui a généré un événement
- **int** `getColumn ()` : index de la colonne de l'événement

```
public class TableOiseauMammiferes extends JFrame {
```

```
private JTable table;
```

```
public TableOiseauMammiferes() {
```

```
Object[][] donnees = {  
    {"Chien", "Mammifère", 16, false},  
    {"Chat", "Mammifère ", 38, false},
```

```
    ...
```

```
    {"Perruche bleue", "Oiseau", 15, true}
```

```
};
```

```
String[] colonnes = {"Animal", "Espèce", "Nombre", "A surveiller"};
```

```
// Création de la JTable :
```

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

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

```
jsp.setPreferredSize(new Dimension(500, 300));
```

```
JPanel pan = (JPanel) this.getContentPane();
```

```
pan.add(jsp);
```

```
// Choix du mode de modification d'affichage :
```

```
table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
// Modifier la largeur des colonnes par défaut :
```

```
TableColumn col;
```

```
for (int i = 0; i < table.getModel().getColumnCount(); i++) {
```

```
    col = table.getColumnModel().getColumn(i);
```

```
    if (i == 1) {
```

```
        col.setPreferredWidth(200);
```

```
    } else {
```

```
        col.setPreferredWidth(100);
```

```
    }
```

```
}
```

```
//Choisir le mode de sélection :
```

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

```
// Définition de l'editeur de la colonne Espèce :
```

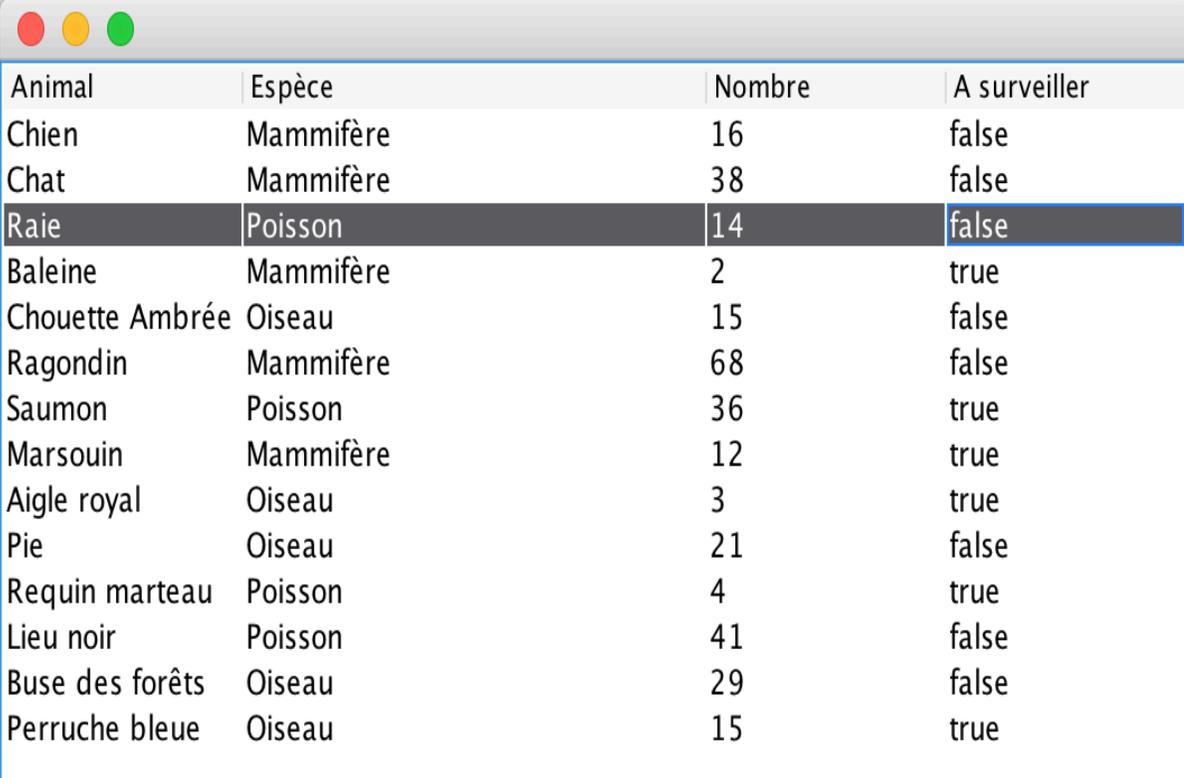
```
this.setEditorColEspece(table.getColumnModel().getColumn(1));
```

```
// Permettre le tri par double-clic sur l'entête colonne :
```

```
table.setAutoCreateRowSorter(true);
```

```
}
```

## Exemple complet de JTable (statique)



Animal	Espèce	Nombre	A surveiller
Chien	Mammifère	16	false
Chat	Mammifère	38	false
Raie	Poisson	14	false
Baleine	Mammifère	2	true
Chouette Ambrée	Oiseau	15	false
Ragondin	Mammifère	68	false
Saumon	Poisson	36	true
Marsouin	Mammifère	12	true
Aigle royal	Oiseau	3	true
Pie	Oiseau	21	false
Requin marteau	Poisson	4	true
Lieu noir	Poisson	41	false
Buse des forêts	Oiseau	29	false
Perruche bleue	Oiseau	15	true

Attention, tri peu performant pour des colonnes de String !

```
private void setEditorColEspece(TableColumn column) {
    JComboBox cbbox = new JComboBox();
    cbbox.addItem("Mammifère");
    cbbox.addItem("Oiseau");
    cbbox.addItem("Poisson");
    cbbox.addItem("Insecte");
    cbbox.addItem("Araignée");
    cbbox.addItem("Autre");
    column.setCellEditor(new DefaultCellEditor(cbbox));

    // ajout d'une bulle de texte :
    DefaultTableCellRenderer rendu = new DefaultTableCellRenderer();
    rendu.setToolTipText("Cliquer pour voir les valeurs possibles");
    column.setCellRenderer(rendu);
}
```

Animal	Espece	Nombre	A surveiller
Chien	Mammifère	16	false
Chat	Mammifère	38	false
Raie	Poisson	14	false
Baleine	Mammifère	2	true
Chouette Amb...	Mammifère	15	false
Ragondin	Oiseau	68	false
Saumon	Poisson	36	true
Marsouin	Insecte	12	true
Aigle royal	Araignée	3	true
Pie	Autre	21	false
Requin marteau	Poisson	4	true
Lieu noir	Poisson	41	false
Buse des forêts	Oiseau	29	false
Perruche bleue	Oiseau	15	true

## Exemple JTable statique (suite)

```
public static void main(String[] args) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            TableOiseauMammiferes fen = new TableOiseauMammiferes();
            TableModel model = fen.table.getModel();

            // affichage de toutes les valeurs de la table :
            for (int i = 0; i < model.getRowCount(); i++) {
                System.out.print("Ligne " + i);
                for (int j = 0; j < model.getColumnCount(); j++) {
                    System.out.print(" " + model.getValueAt(i, j));
                }
                System.out.println();
            }
            fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            fen.setLocationRelativeTo(null);
            fen.pack();
            fen.setVisible(true);
        }
    });
}
```

*Nota : pas de cast possible en DefaultTableModel ici (erreur exécution), car instancié en classe anonyme par la MVJ...*

**retourne un Object (appel à toString())**

```
run-single:
Ligne 0 Chien Mammifère 16 false
Ligne 1 Chat Mammifère 38 false
Ligne 2 Raie Poisson 14 false
Ligne 3 Baleine Mammifère 2 true
Ligne 4 Chouette Ambrée Oiseau 15 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**
- Toutes les données des tables doivent être placées dans des **tableaux** ou des **conteneurs**
  - Peut être inapproprié pour certaines données, par ex. les valeurs lues dans des BD, qu'on n'a pas envie d'instancier en objet

Si on souhaite une autre configuration

→ Créer un modèle de table

# Tables dynamiques (modifiables)

- Création du modèle (ici avec **DefaultTableModel**):

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

- Création de la table:

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

## Toutes les modifications de la table se font SUR LE MODELE :

- 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);
```

# MVC pour la JTable

```
class BooleanTableModel extends AbstractTableModel {  
    String[] columns = {"STUDENT ID", "NAME", "SCORE", "PASSED"};  
    Object[][] data = {  
        {"S001", "ALICE", 90.00, Boolean.TRUE},  
        {"S002", "BOB", 45.50, Boolean.FALSE},  
        {"S003", "CAROL", 60.00, Boolean.FALSE},  
        {"S004", "IGNASIA", 85.80, Boolean.TRUE},  
        {"S005", "MALLORY", 75.80, Boolean.TRUE}  
    };  
}
```

Le modèle

monModele.setValueAt(...);

monModele.getValueAt()

La JTable = la Vue dans MVC

La vue

« Le contrôleur »

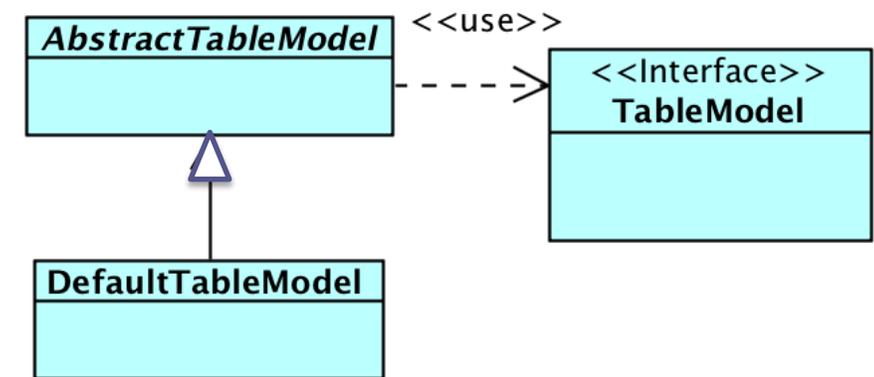
monModele.getValueAt()  
monModele.setValueAt(...);

STUDENT ID	NAME	SCORE	PASSED
S001	ALICE	90	<input checked="" type="checkbox"/>
S002	BOB	45.5	<input type="checkbox"/>
S003	CAROL	60	<input type="checkbox"/>
S004	IGNASIA	85.8	<input checked="" type="checkbox"/>
S005	MALLORY	75.8	<input checked="" type="checkbox"/>

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

## 2 façons de créer un modèle de JTable

- Deux implémentations existent pour faciliter l'utilisation du modèle *TableModel* (interface) :
  - La classe *DefaultTableModel*
  - La classe abstraite ***AbstractTableModel***
- Si on souhaite ajouter un éditeur, un renderer, etc... créer un modèle qui va hériter d'*AbstractTableModel*
- Si on veut juste pouvoir modifier les données, utiliser un *DefaultTableModel* suffit



Si *setValueAt()* non implémentée dans un *AbstractTableModel* : les données seront **non modifiables**

# Création d'un modèle avec `AbstractTableModel`

- Création d'une classe "Modèle" qui hérite de la classe `AbstractTableModel`

```
class MonModeleTable extends AbstractTableModel {
```
- 3 méthodes doivent alors être systématiquement implémentées :
  - `public int getRowCount()`
    - Retourne le nombre de lignes du modèle de table
  - `public int getColumnCount()`
    - Le nombre de colonnes
  - `public Object getValueAt(int i, int c)`
    - Retourne l'objet du modèle correspondant à la cellule (i, c) et l'affiche dans la table

## Méthode `setValueAt()` de *TableModel*

- Est appelée **automatiquement** à chaque modification de la `JTable`
  - Pas besoin de gérer les événements 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
  - Par ex. si le modèle possède un conteneur de données sous la forme `Object[ ][ ] data`
  - le code de `setValueAt(Object valeur, int ligne, int col)` sera :

```
data[ligne][col] = valeur;
this.fireTableCellUpdated(ligne, col);
// pour propager la modification à toutes les vues
```
- On peut aussi vouloir **gérer soi-même et de façon globale** les événements de la table
  - Ajouter d'autres écouteurs (passage de la souris, etc.)

# Propagation des modifications effectuées sur la JTable

- Modification **dynamique**, par exemple en fonction de la ligne sélectionnée :
  - **Ajout** : ligne suivante de celle sélectionnée ou dernière ligne
  - **Suppression** : ligne en cours de sélection
- On utilise les méthodes `fireXXXX()`
- A chaque modification de la table, il faut **notifier les écouteurs** (dont le composant graphique **JTable**) des modifications apportées au modèle
- Avec :

```
monModeleTable.fireTableCellUpdated(row,col);  
monModeleTable.fireTableChanged(evt);  
monModeleTable.fireTableDataChanged();  
monModeleTable.fireTableRowsDeleted(row1, row2)
```

*Avec le Design sous IDE, les méthodes `firexxx()` sont gérées automatiquement*

## Exemple de table avec un écouteur

```
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
}
```

*(cette méthode appelle `fireTableChanged()` pour propager la modification)*

## 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("-----");
}
}
```

# Rendre des colonnes non modifiables

*(possible uniquement pour les tables avec un modèle)*

- 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** :

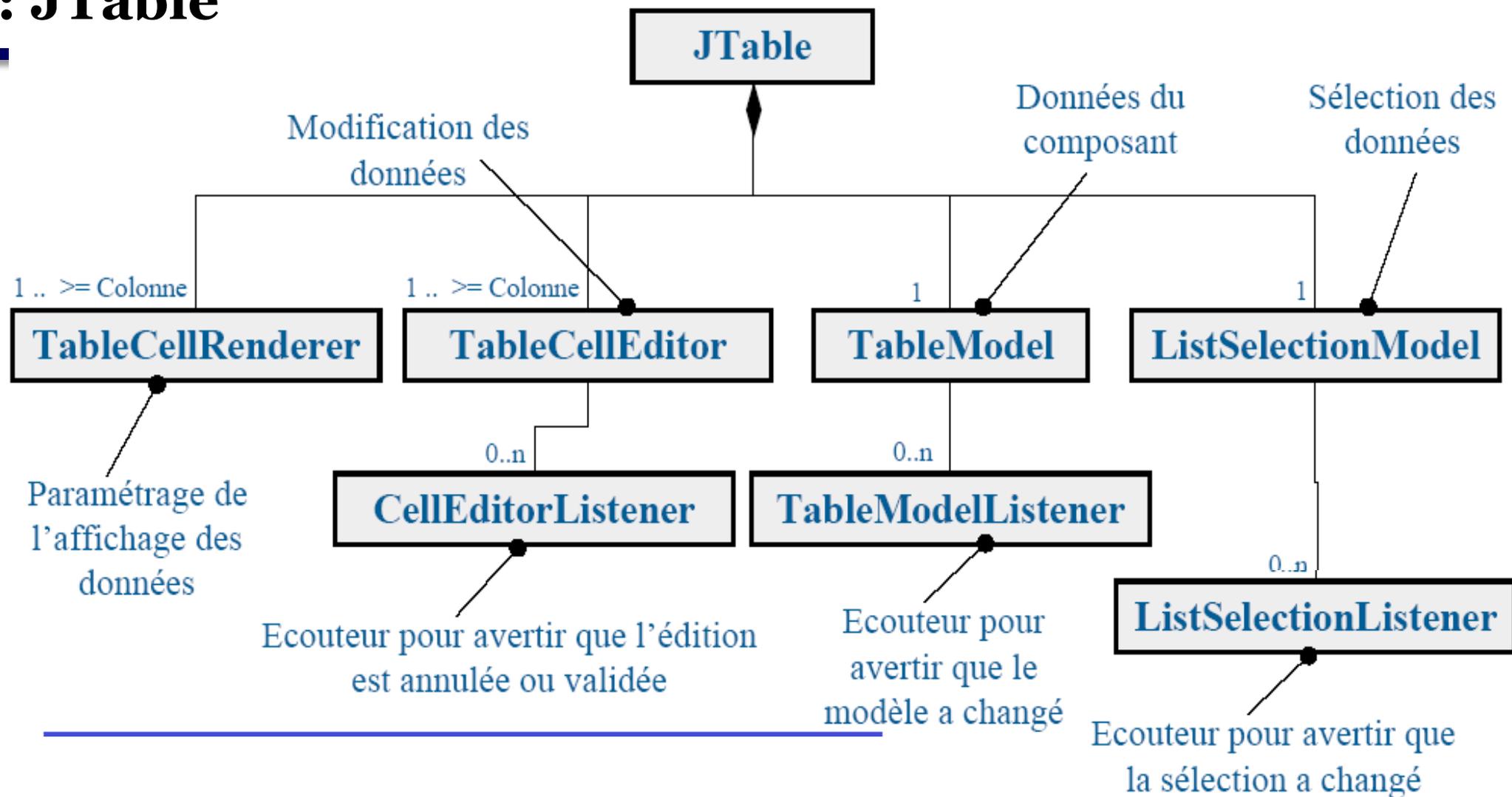
```
boolean isCellEditable(int rowIndex, int colIndex)
```

- Quand `isCellEditable()` est à `FALSE`, `setValueAt()` n'a aucun effet.

```
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<sup>e</sup> et dernière colonnes sont modifiables

# En résumé : JTable



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

# Composants gérés par défaut

- Selon le type de données se trouvant dans le modèle (indiqué par *getColumnClass()* de `TableModel`) les objets `Editor` et `Renderer` retournent des **composants prédéfinis**:

Composant retourné par l'objet « Editor » :

*Boolean* : *JCheckBox*

Autre : *JTextField*

Composant retourné par l'objet « Renderer » :

*Boolean* : *JCheckBox*

*Number, Double et Float* : *JLabel* aligné à droite

*ImageIcon* : *JLabel* aligné au centre

- Il faut pour cela surcharger *getColumnClass()* de `TableModel` en renvoyant la classe de la colonne :

```
@Override
```

```
public Class getColumnClass(int col) {
    return getValueAt(0, col).getClass(); }
}
```

- Si *getColumnClass()* n'est pas implémentée, les données sont considérées comme des **String**

# 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
- La définition d'un modèle de table par *DefaultTableModel*
- Le mécanisme de **synchronisation Vue / Modèle** avec une *JTable*
- Comment récupérer *la ou les ligne(s) sélectionnée(s)* d'une table
- Le fonctionnement des méthodes de *propagation des modifications* effectuées sur une *JTable* dans *setValeurAt()* : *firexxx()* (même si utilisées automatiquement par l'IDE)



# Illustration...

## Ajout d'une ligne à une JTable

- via l'IHM
- avec un modèle AbstractTableModel

## Illustration JTable : ajout d'une ligne

- Soit un modèle utilisant cette fois sur un ArrayList pour les données
  - Chaque ligne sera un tableau d'Object :

Dans le modèle :



```
private ArrayList data = new ArrayList();  
  
Object[] d1 = {uneLigne};  
  
Object[] d2 = {uneAutreLigne};  
  
...  
  
data.add(d1); data.add(d2); etc.
```

## Exemple: ajout de ligne (1/2)

```
public class TableAjoutLigne extends JPanel implements java.awt.event.ActionListener  
{
```

```
    MonModeleTable monModele;
```

```
    ...
```

```
    public TableAjoutLigne() {
```

```
        monModele = new MonModeleTable();
```

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

```
        table.setPreferredSize(new Dimension(500, 150));
```

```
        monModele.initialise();
```

```
        ...
```

```
        JButton unButton = new JButton("Ajout Ligne");
```

```
        unButton.addActionListener(this);
```

```
        ...
```

```
    }
```

(classe définie ci-après)

initialise les données de  
l'ArrayList du modèle

Définition du bouton de  
l'IHM pour l'ajout

## Exemple: ajout de ligne (2/2)

```
public void actionPerformed (java.awt.event.ActionEvent e) {  
    Object[] donneeLigne = {"Demi", "Moore", "Step", 3, false};  
    monModele.ajoutLigne(donneeLigne);  
}  
...
```

```
class MonModeleTable extends AbstractTableModel {  
    private String[] columnNames = {"First Name", "Last Name", "Sport", "# of  
Years", "Vegetarian"};  
    private ArrayList lesDonnees = new ArrayList();  
    public void initialise() {  
        Object d1[]={"Mary", "Campione", "Snowboarding", 5, false};  
        ...  
        lesDonnees.add(d1);  
        ...  
    } // d'initialise()  
}
```

```
public int getColumnCount() {
    return columnNames.length; }

public int getRowCount() {
    return lesDonnees.length; }

public Object getValueAt(int row, int col) {
    return lesDonnees[row][col]; } // ou, autre ex.: return new Integer((1+row)*(1+col));
```

## Implémentation des méthodes d'AbstractTableModel

## Définition de méthodes supplémentaires:

```
public void ajoutLigne ( Object[] nouvellesDonnees) {
    lesDonnees.add(nouvellesDonnees);
    fireTableStructureChanged();
}
```

```
public String getColumnName (int col) {
    return columnNames[col]; }

// Utiliser l'éditeur de cellules par défaut et pas du texte :

public Class getColumnClass (int c) {
    return getValueAt(0, c).getClass(); }
```

← Propagation aux vues  
(dont la JTable)

*Si on ne le fait pas, la vue  
reprend les anciennes  
valeurs après un  
rafraichissement écran*

# (fin)

```
// Spécifier quelles sont les cellules modifiables / non modifiables :
```

```
public boolean isCellEditable (int row, int col) {  
    if (col < 2) {  
        return false;  
    } else {  
        return true;  
    }  
}
```

Les cellules des colonnes 0 et 1 sont non modifiables (nom, prénom), les autres oui.

## Autres Références

- **JTable** : tutoriel très complet d'Oracle
  - <http://docs.oracle.com/javase/tutorial/uiswing/components/table.html>