D[i] Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Faculté Sciences et Ingénierie

Licence Informatique

Gestion des tableaux et des arbres

Mathieu RAYNAL

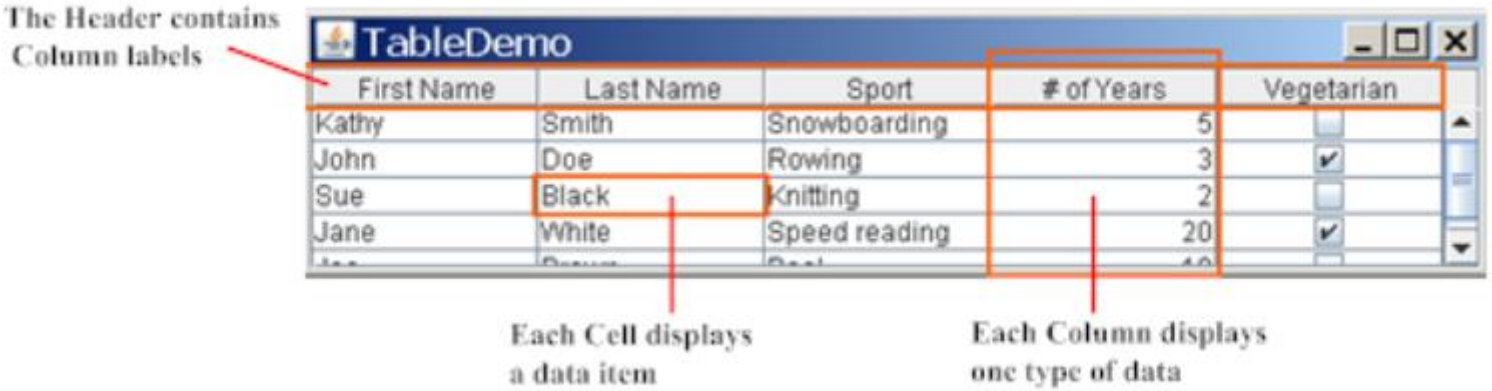
mathieu.raynal@irit.fr

http://www.irit.fr/~Mathieu.Raynal

Les tableaux

Les tableaux

- Un objet graphique : JTable
  - Permet d'afficher un tableau
  - Permet l'édition de son contenu



- Présenter une vision structurée d'un grand nombre de données et permettre leur manipulation

Un composant graphique : **JTable**

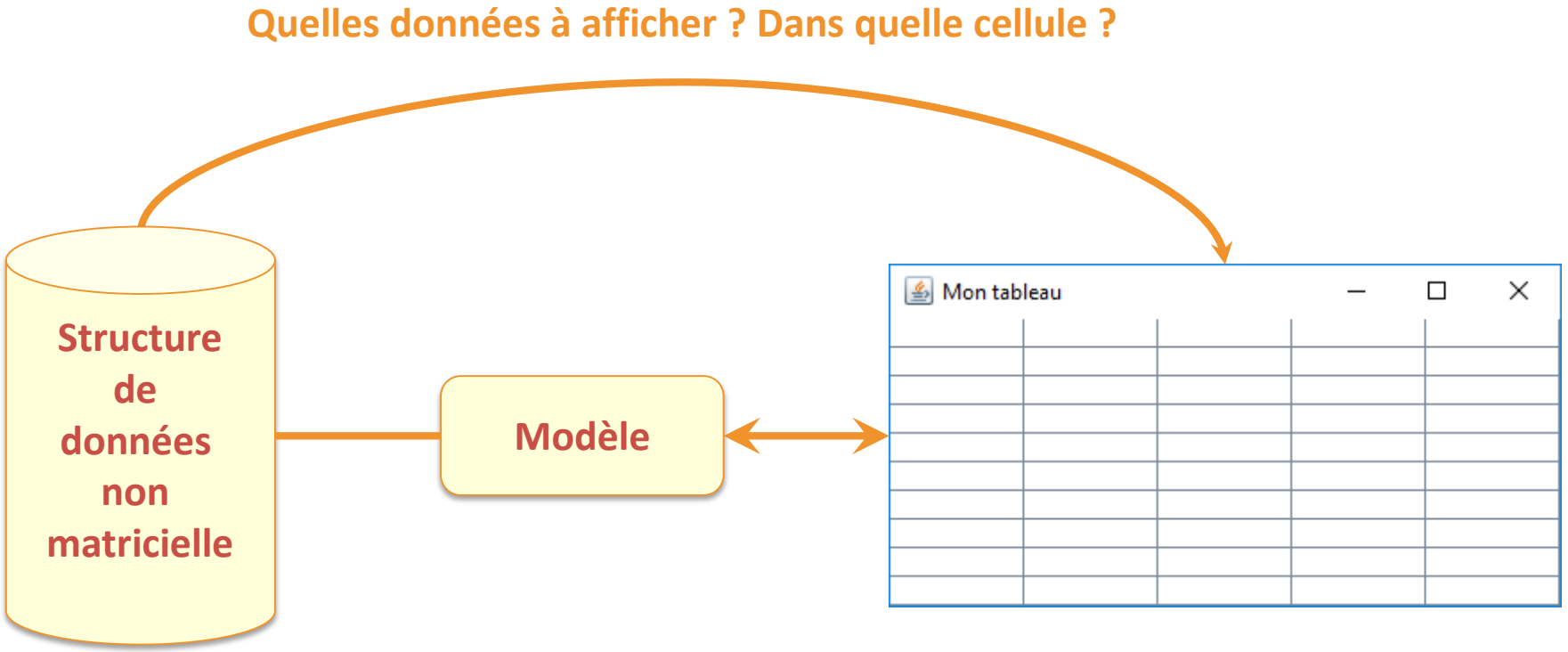
- Création d'un tableau à partir d'une matrice d'éléments
  - Constructeurs

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

  - Principales méthodes

void **setValueAt**(Object aValue, int row, int column)
Void **revalidate**()

Liaison avec une structure de données dynamique



Utilisation d'un modèle de données

- Constructeur de **JTable** avec un modèle

  JTable(TableModel modele)

- Le modèle indique comment afficher un ensemble de données provenant d'une structure de données
  - Lors de sa création, la **JTable** interroge le modèle pour savoir
    - le nombre de lignes et de colonnes à créer ;
    - Quelle information mettre dans chaque cellule ;
    - La nature de chaque information à afficher;
    - Etc.
  - L'interface **TableModel** définit les méthodes nécessaires pour assurer la liaison avec la **JTable**
  - Toute **JTable** a un modèle par défaut : une instance de **DefaultTableModel**

L'interface **TableModel**

- Elle contient 9 méthodes pour
  - Connaitre le nombre de colonnes à afficher

    int **getColumnCount**()

  - Connaitre le nombre de lignes à afficher

    int **getRowCount**()

  - Connaitre l'élément à mettre dans une cellule

    Object **getValueAt**(int rowIndex, int columnIndex)

  - Connaitre le titre des colonnes

    String **getColumnName**(int columnIndex)

L'interface **TableModel**

- Connaitre le type d'élément dans une colonne

```
Class<?> getColumnClass(int columnIndex)
```

- Savoir si une cellule est éditable ou non

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

- Modifier l'élément contenu dans une cellule

```
void setValueAt(Object aValue, int rowIndex, int columnIndex)
```

- Gérer la liste des listeners du modèle

```
void addTableModelListener(TableModelListener l)
void removeTableModelListener(TableModelListener l)
```

La classe abstraite **AbstractTableModel**

- Implémente **TableModel**
- Nécessité de définir les méthodes abstraites

int **getColumnCount**()

int **getRowCount**()

Object **getValueAt**(int rowIndex, int columnIndex)

Prévenir des changements dans le modèle

- Mise à jour d'une cellule

  void **fireTableCellUpdated**(int row, int column)

- Mise à jour d'une ligne

  void **fireTableRowsUpdated**(int firstRow, int lastRow)

- Mise à jour de la table

  void **fireTableDataChanged**()

- Modification de la structure de la table

  void **fireTableStructureChanged**()

- Insertion d'une ligne

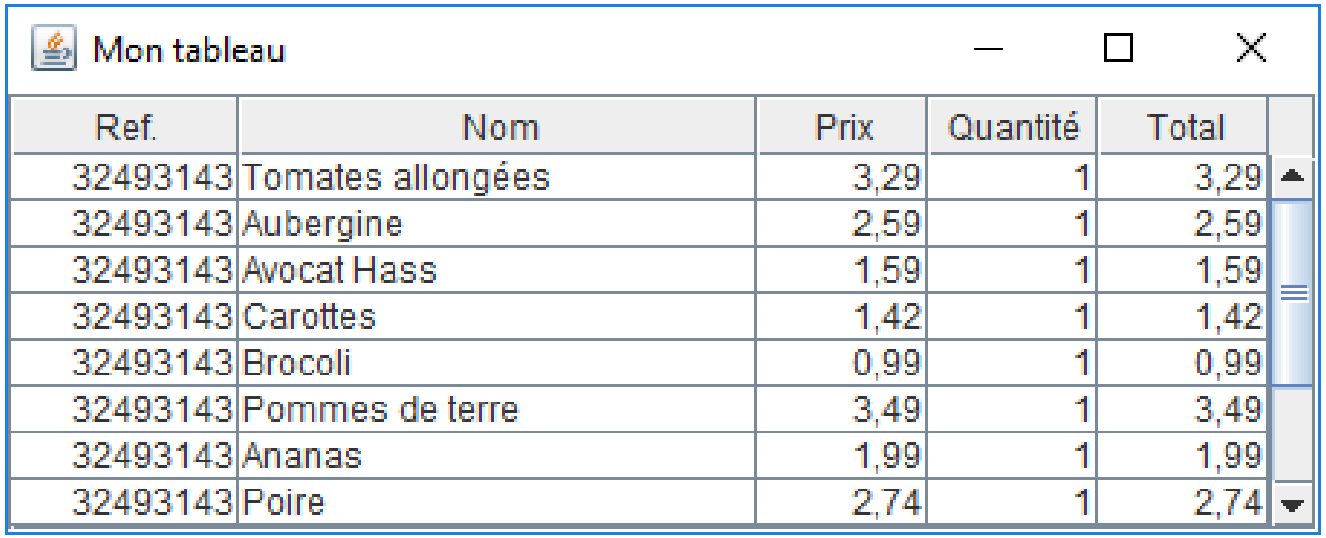
  void **fireTableRowsInserted**(int firstRow, int lastRow)

- Suppression d'une ligne

  void **fireTableRowsDeleted**(int firstRow, int lastRow)

Exercice 1

- A partir de la structure de données ci-dessous, créez le modèle permettant d'afficher les données comme sur ce tableau



Ref.	Nom	Prix	Quantité	Total
32493143	Tomates allongées	3,29	1	3,29
32493143	Aubergine	2,59	1	2,59
32493143	Avocat Hass	1,59	1	1,59
32493143	Carottes	1,42	1	1,42
32493143	Brocoli	0,99	1	0,99
32493143	Pommes de terre	3,49	1	3,49
32493143	Ananas	1,99	1	1,99
32493143	Poire	2,74	1	2,74

Exercice 1

```java
public class Produit
{
    public Categorie categorie;
    public int reference;
    public String nom;
    public float prix;
    public int quantite;

    public Produit(Categorie cat, int ref,
                   String nom, float prix, int qutite)
    {
        categorie = cat;
        reference = ref;
        this.nom = nom;
        this.prix = prix;
        quantite = qutite;
    }

    public float getTotal(){
        return (float)quantite * prix;
    }
}
```

```java
public class ListeCourses {
    ArrayList<Produit> listeProduit;

    public ListeCourses(){
        listeProduit = new ArrayList<Produit>();
    }

    public ArrayList<Produit> getListe(){
        return listeProduit;
    }

    public int getNbProduit(){
        return listeProduit.size();
    }

    public Produit getProduit(int indice){
        return listeProduit.get(indice);
    }
}
```

Correction

```
25        ListeCourses liste = new ListeCourses();
26        ModeleTableau modele = new ModeleTableau(liste);
27        tableau = new JTable(modele);
```

```
3  import javax.swing.table.AbstractTableModel;
7
8  public class ModeleTableau  extends AbstractTableModel
9  {
10    ListeCourses listeCourses;
11
12    public ModeleTableau(ListeCourses listeCourses){
13       this.listeCourses = listeCourses;
14    }
15
16    public int getColumnCount()   {
17       return 5;
18    }
19
20    public int getRowCount(){
21       return listeCourses.getNbProduit();
22    }
```

Correction

```
24  public Object getValueAt(int indiceLigne, int indiceColonne){
25      Produit p = listeCourses.getProduit(indiceLigne);
26      switch(indiceColonne)
27      {
28          case 0: return p.reference;
29          case 1: return p.nom;
30          case 2: return p.prix;
31          case 3: return p.quantite;
32          case 4: return p.getTotal();
33          default: return null;
34      }
35  }
36
37  public String getColumnName(int indiceColonne)    {
38      switch(indiceColonne)
39      {
40          case 0: return "Ref.";
41          case 1: return "Nom";
42          case 2: return "Prix";
43          case 3: return "Quantité";
44          case 4: return "Total";
45          default: return null;
46      }
47  }
```

Correction

```
49  public void setValueAt(Object val, int indiceLigne, int indiceColonne){
50      Produit p = listeCourses.getProduit(indiceLigne);
51      if(indiceColonne==3)
52          p.quantite = ((Integer)val).intValue();
53      fireTableDataChanged();
54  }
55
56  public Class getColumnClass(int indiceColonne){
57      return getValueAt(0, indiceColonne).getClass();
58  }
59
60  public boolean isCellEditable(int indiceLigne, int indiceColonne){
61      if(indiceColonne==3)
62          return true;
63      return false;
64  }
```

Modifier la taille des lignes et colonnes

- Hauteur des lignes

  void **setRowHeight**(int row, int rowHeight)
  void **setRowHeight**(int rowHeight)

- Longueur des colonnes
  - Accéder au modèle de colonnes dans la classe **JTable**

    TableColumnModel **getColumnModel**()

  - Accéder à une colonne dans la classe **TableColumnModel**

    TableColumn **getColumn**(int indice)

  - Changer sa taille dans la classe **TableColumn**

    void **setPreferredWidth**(int longueur)

Rendu des cellules

- Comment afficher les données dans une cellule ?
  - En fonction du type d'Objet
- Des rendus par défaut
  - Boolean : case à cocher
  - Number : label aligné à droite
  - Object : chaîne de caractères correspondante à l'objet
- Si modèle de données personnalisé, pensez à définir le type d'objet présent dans chaque cellule

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

Personnalisation du rendu des cellules

- Implémenter l'interface **TableCellRenderer**
- ou étendre de la classe **DefaultTableCellRenderer**
  - Qui étend de JLabel

```
public Component getTableCellRendererComponent(
                                    JTable table,
                                    Object value,
                                    boolean isSelected,
                                    boolean hasFocus,
                                    int row,
                                    int column)
```

Personnalisation du rendu des cellules

- Associer le nouveau rendu aux cellules
  - Des colonnes souhaitées

```
void setCellRenderer(TableCellRenderer cellRenderer)
```

  - Des cellules ayant ce type d'objet

```
void setDefaultRenderer(Class<?> columnClass,
                        TableCellRenderer renderer)
```

Exercice 2

- Ecrire la classe permettant d'avoir le même rendu que sur le tableau :
  - Les fruits & legumes en vert
  - Les viandes & poissons en rouge

```
3  public enum Categorie {
4     FRUITS_LEGUMES("Fruits & Légumes"),
5     VIANDES_POISSONS("Viandes & Poissons"),
6     PAINS_PATISSERIE("Pains & Pâtisserie"),
7     PRODUIT_LAITIER("Produits laitiers"),
8     SURGELES("Surgelés"),
9     EPICERIE_SUCREE("Epicerie sucrée"),
10    EPICERIE_SALEE ("Epicerie salée"),
11    BOISSONS("Boissons");
12    private String nom;
13    private Categorie(String n){nom=n;}
14    public String getNom(){return nom;}
15 }
```

Mon tableau

Ref.	Nom	Prix	Quantité	Total
32493143	Brocoli	0.99€	1	0.99€
32493143	Pommes de terre	3.49€	1	3.49€
32493143	Ananas	1.99€	1	1.99€
32493143	Poire	2.74€	1	2.74€
32493143	Bananes	0.99€	2	1.98€
32493143	Pavés de boeuf	4.59€	1	4.59€
32493143	Poulet rôti	7.9€	1	7.9€
32556179	Brique de lait 1L	0.88€	1	0.88€

Correction

```
13  public class RenduTableau extends DefaultTableCellRenderer{
14     private static final Color ROUGE = new Color(128,0,0);
15     private static final Color VERT = new Color(0,128,0);
16
17     public Component getTableCellRendererComponent(JTable table, Object value,
18                                                    boolean isSelected, boolean hasFocus,
19                                                    int row, int column){
20        if(column == 1){
21           setText(String.valueOf(value));
22           setHorizontalAlignment(JLabel.LEFT);
23        }else{
24           if(column == 0 || column == 3)
25              setText(String.valueOf(value));
26           else
27              setText(String.valueOf(value)+"€");
28           setHorizontalAlignment(JLabel.RIGHT);
29        }
30
31        Produit p = ((ModeleTableau)table.getModel()).getProduitAt(row);
32        switch(p.categorie){
33           case FRUITS_LEGUMES:setForeground(VERT);break;
34           case VIANDES_POISSONS:setForeground(ROUGE);break;
35           default:setForeground(Color.BLACK);break;
36        }
37        return this;
38     }
39  }
```

Correction

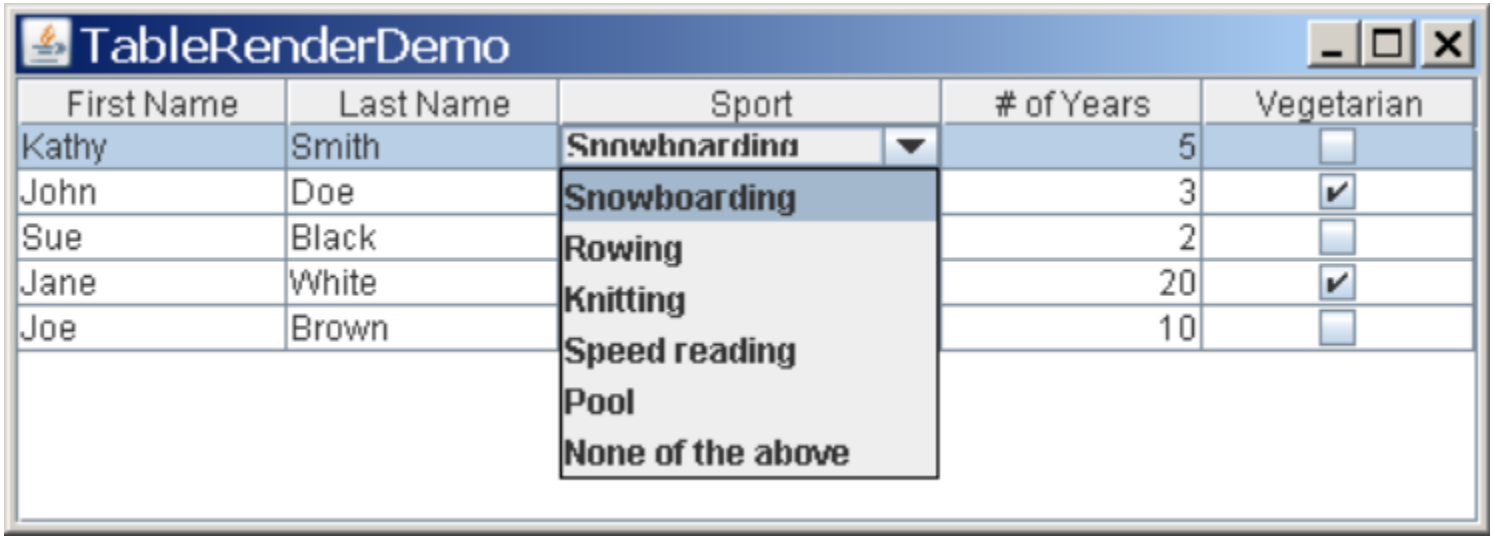
```
38      tableau = new JTable(modele);
39      TableColumnModel modeleColonne = tableau.getColumnModel();
40      modeleColonne.getColumn(0).setPreferredWidth(50);
41      modeleColonne.getColumn(0).setCellRenderer(new RenduTableau());
42      modeleColonne.getColumn(1).setPreferredWidth(150);
43      modeleColonne.getColumn(1).setCellRenderer(new RenduTableau());
44      modeleColonne.getColumn(2).setPreferredWidth(30);
45      modeleColonne.getColumn(2).setCellRenderer(new RenduTableau());
46      modeleColonne.getColumn(3).setPreferredWidth(30);
47      modeleColonne.getColumn(3).setCellRenderer(new RenduTableau());
48      modeleColonne.getColumn(3).setCellEditor(new EditeurQuantite());
49      modeleColonne.getColumn(4).setPreferredWidth(30);
50      modeleColonne.getColumn(4).setCellRenderer(new RenduTableau());
```

Editeur

- Différentes manières de saisir une donnée dans une cellule
  - Champ texte
  - Case à cocher
  - Liste déroulante
  - Color picker
  - ...

Editeur par défaut

- DefaultCellEditor
  - JTextField, JCheckBox, JComboBox



Edition par liste déroulante - exemple

```
TableColumn sportColumn = table.getColumnModel().getColumn(2);
...
JComboBox comboBox = new JComboBox();
comboBox.addItem("Snowboarding");
comboBox.addItem("Rowing");
comboBox.addItem("Chasing toddlers");
comboBox.addItem("Speed reading");
comboBox.addItem("Teaching high school");
comboBox.addItem("None");
sportColumn.setCellEditor(new DefaultCellEditor(comboBox));
```

Personnalisation de l'édition des cellules

- Définir une classe qui
  - met en oeuvre l'interface **TableCellEditor**

    Component **getTableCellEditorComponent**(JTable table, Object value, boolean isSelected, int row, int column)
  - et étend la classe **AbstractCellEditor**

    Object **getCellEditorValue**()
- Configurer l'éditeur
  - JTable

    void **setDefaultEditor**(Class<?> columnClass, TableCellEditor editor)
  - TableColumn

    void **setCellEditor**(TableCellEditor cellEditor)

Exercice 3

- Ecrire la classe qui permet de modifier les cellules liées à la quantité
  - Au moyen d'un **JSpinner**

Récap. model/rendu/edition

	Classe utilisée par défaut	Personnalisation
Modèle de données	Classe concrète **DefaultTableModel**	Extension de la classe abstraite **AbstractTableModel**
Rendu	Classe concrète **DefaultCellRenderer**	Extension d'un élément graphique (**héritant de Jcomponent**) + mise en oeuvre de l'interface **TableCellRenderer**
Edition	Classe concrète **DefaultCellEditor**	Extension de la classe abstraite **AbstractCellEditor** + mise en oeuvre de l'interface **TableCellEditor**

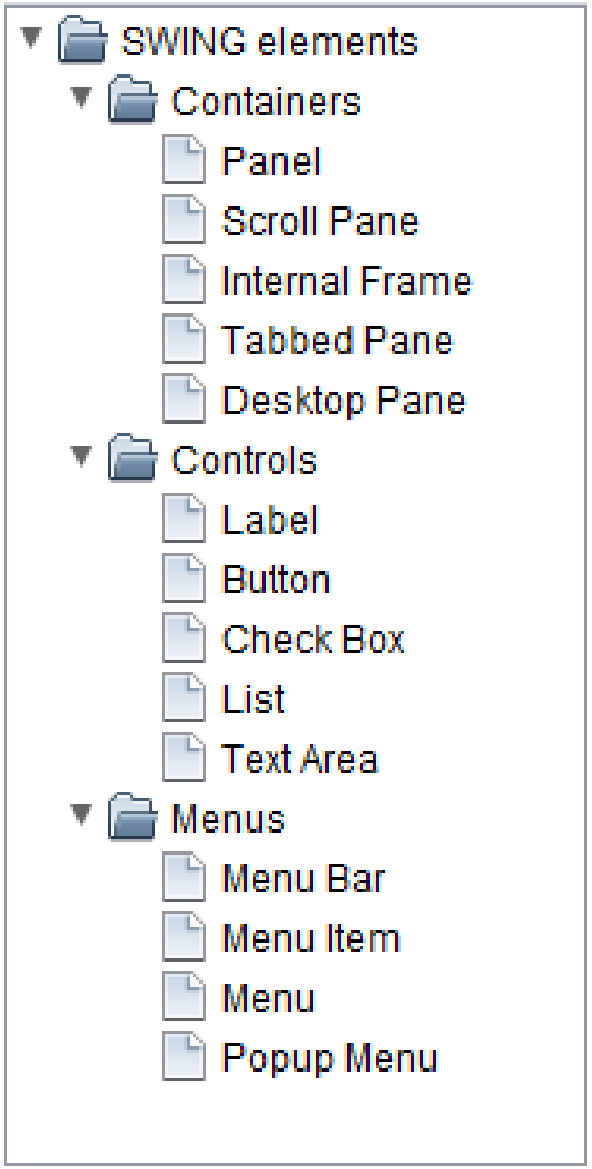
D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse
Faculté Sciences et Ingénierie

Arbres

Arbre: utilité et composition

- Hiérarchisation des éléments
- Composition d'un arbre
  - Racine
  - Branches
  - Feuilles



Création simple d'un JTree

- Composant graphique – **JTree**

  public **JTree**(TreeNode root)

- Noeuds/Données – **DefaultMutableTreeNode**

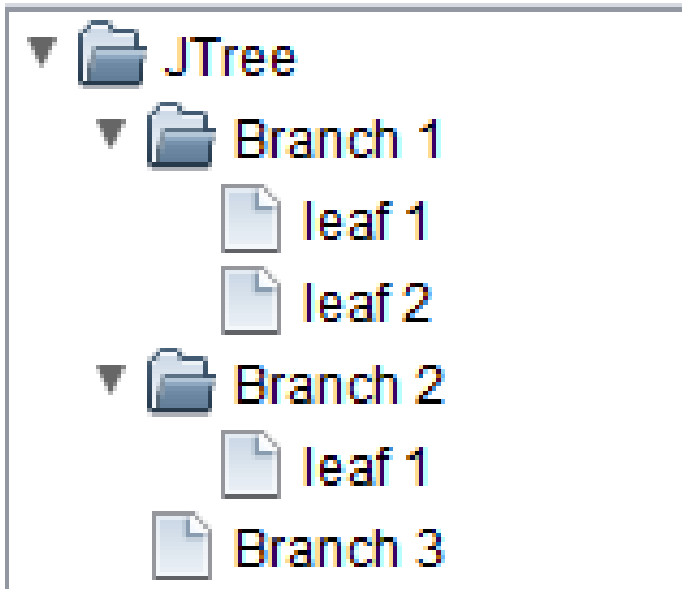
  public **DefaultMutableTreeNode**(Object userObject)

- Ajout d'un sous-noeud

  void **add**(MutableTreeNode newChild)

Exercice 4

- Créer cet arbre



Correction

```
27      DefaultMutableTreeNode racine = new DefaultMutableTreeNode("JTree");
28      DefaultMutableTreeNode noeud1 = new DefaultMutableTreeNode("Branch 1");
29      DefaultMutableTreeNode noeud2 = new DefaultMutableTreeNode("Branch 2");
30      DefaultMutableTreeNode noeud3 = new DefaultMutableTreeNode("Branch 3");
31      racine.add(noeud1);
32      racine.add(noeud2);
33      racine.add(noeud3);
34      DefaultMutableTreeNode feuille1 = new DefaultMutableTreeNode("leaf 1");
35      DefaultMutableTreeNode feuille2 = new DefaultMutableTreeNode("leaf 2");
36      noeud1.add(feuille1);
37      noeud1.add(feuille2);
38      DefaultMutableTreeNode feuille3 = new DefaultMutableTreeNode("leaf 1");
39      noeud2.add(feuille3);
40      arbre = new JTree(racine);
```

Création d'un JTree associé à un modèle

- Constructeur JTree

  public **JTree**(TreeModel modele)

- DefaultTreeModel
  - associé par défaut à une instance de classe JTree

    DefaultTreeModel(TreeNode root)

Création d'un JTree associé à un modèle

- Implémentation de l'interface TreeModel

void **addTreeModelListener**(TreeModelListener l)

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

int **getChildCount**(Object parent)

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

Object **getRoot**()

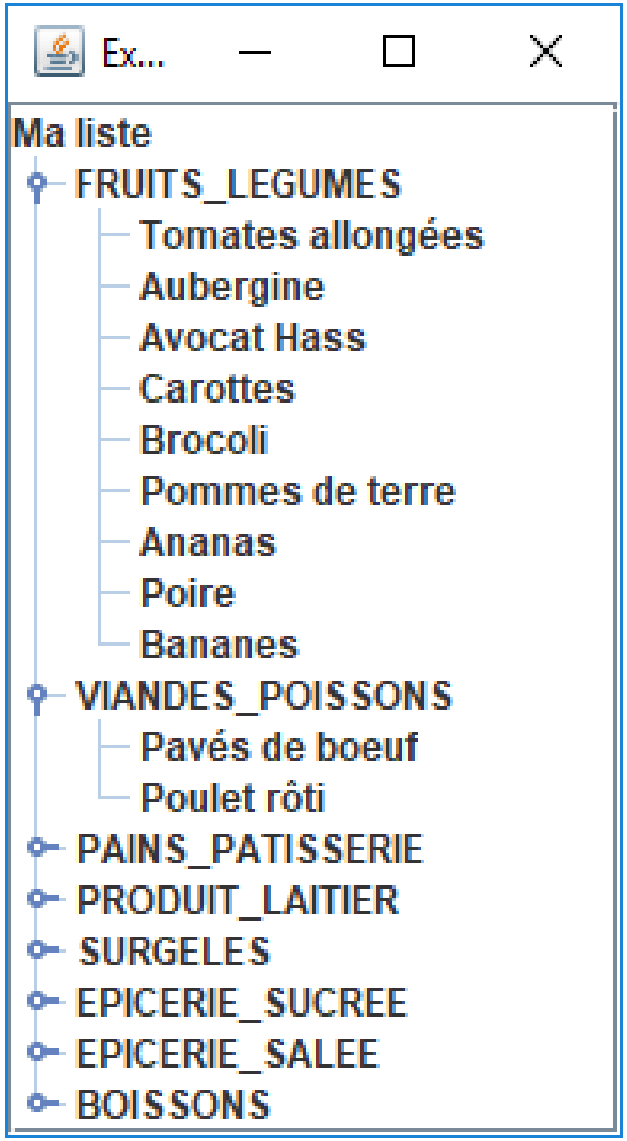
boolean **isLeaf**(Object node)

void **removeTreeModelListener**(TreeModelListener l)

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

Exercice 4

- Toujours à partir de la même structure de données, réalisez le modèle qui permet d'afficher l'arbre ci-contre



Personnalisation du rendu (simple)

- Affichage de l'élément racine

  public void **setRootVisible(boolean rootVisible)**

- Affichage des "poignées de dépliage/repliage"

  public void **setShowsRootHandles(boolean newValue)**

Personnalisation du rendu

- Reconfiguration du renderer associé par défaut DefaultTreeCellRenderer :
  - public DefaultTreeCellRenderer()

- Ou extension de la classe DefaultTreeCellRenderer
  - Implémentation de l'interface de rendu **TreeCellRenderer**
    public void setCellRenderer(TreeCellRenderer x)

DefaultTreeCellRenderer

- Icône associée aux feuilles

  void **setLeafIcon**(Icon newIcon)

- Icône associée nœuds repliés

  void **setClosedIcon**(Icon newIcon)

- Icône associée aux nœuds dépliés

  void **setOpenIcon**(Icon newIcon)

Mise en oeuvre d'une classe de rendu

- Extension de la classe DefaultTreeCellRenderer
- Surcharge de la méthode

```
public Component getTreeCellRendererComponent(
                        JTree tree,
                        Object value,
                        boolean sel,
                        boolean expanded,
                        boolean leaf,
                        int row,
                        boolean hasFocus)
```