

Swing

Table des matières

| | |
|------------------------------------|----------|
| Swing | 1 |
| Une application swing | 2 |
| La classe principale | 2 |
| Déclarations..... | 2 |
| Constructeur..... | 2 |
| La méthode panelPrincipal..... | 2 |
| Ecouteurs d'évènements..... | 3 |
| Méthode main | 3 |
| Style des composants swing..... | 4 |
| La classe UIManager | 4 |
| Mise en pratique | 4 |
| Composant JTable | 5 |
| La classe DataFileTableModel | 5 |
| Déclaration et variables..... | 5 |
| Constructeur..... | 6 |
| Méthodes à implémenter..... | 6 |
| L'application | 8 |
| Déclarations..... | 8 |
| Constructeur..... | 8 |
| Dimensions préférées..... | 9 |
| Méthode main | 9 |
| La classe WindowCloser | 9 |
| Exercices..... | 9 |

Une application swing

Nous allons créer une mini-application swing qui permettra de choisir le style des composants qu'elle contient à l'aide d'un bouton.

La classe principale

Déclarations

La classe principale sera une classe dérivée du type `Object`. La fenêtre dans laquelle l'application s'exécute ainsi que les composants qu'elle contient seront des champs de cette classe.

```
import javax.swing.*;           //package swing
import java.awt.*;
import java.awt.event.*;

public class SwingApplication {

    JFrame fen;                 //fenêtre de l'application
    JButton changeLAF;          //bouton changement de style
    JLabel nomStyle;            //nom du style utilisé
```

Constructeur

Le constructeur va créer la fenêtre, ajouter les composants qu'elle contient, puis l'afficher. La création et l'ajout des composants seront gérés par la méthode *panelPrincipale* qui fournira un objet de type `JPanel` (`Panel swing`). Il nous suffira donc d'ajouter ce `JPanel` à la fenêtre. Cette opération ne se fait plus directement par la méthode `add` comme avec AWT. Nous devons passer par un container fourni par la méthode **`getContentPane`** de la fenêtre. On obtient le code suivant:

```
public SwingApplication() {
    //création de la fenêtre
    fen=new JFrame("Application Swing");
    //ajout des composants
    fen.getContentPane().add(panelPrincipale(), BorderLayout.CENTER);
    //écouteur pour fermeture de la fenêtre
    fen.addWindowListener(new Fermeture());
    //mise en place et affichage
    fen.pack();
    fen.setBounds(50,50,200,150);
    fen.setVisible(true);
}
```

La méthode *panelPrincipale*

Cette méthode fournit un composant de type `JPanel` contenant les composants de l'application : ici un `JLabel` (`Label swing`) et un `JButton` (`Button swing`). On utilisera un gestionnaire de positionnement de type `GridLayout`.

La classe **`UIManager`** nous permet de connaître le nom du style d'affichage choisi.

Les composants swing disposent d'une méthode **setBorder** qui permet de leur attribuer une bordure fournie par l'une des méthodes (statiques) de la classe **BorderFactory**.

```
public Component panelPrincipal() {
    //-----label nom de style (look and feel)
    nomStyle=new JLabel("LAF: "+UIManager.getLookAndFeel().getName(),
        SwingConstants.CENTER);
    //-----bouton de changement de style
    changeLAF=new JButton("Change");
    //écouteur du bouton
    changeLAF.addActionListener(new ActionBouton());
    //-----le JPanel qui contient le reste
    JPanel pane=new JPanel();
    //définir une bordure, ici vide avec marge
    pane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
    //ajout des composants
    pane.setLayout(new GridLayout(2,1,10,10));
    pane.add(nomStyle);
    pane.add(changeLAF);
    return pane;
}
```

Ecouteurs d'évènements

Nous avons à créer les classes **Fermeture** et **ActionBouton** qui représentent les écouteurs d'évènements liés à la fenêtre et au bouton.

La classe **Fermeture** ferme la fenêtre et met fin à l'application.

```
class Fermeture extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        fen.setVisible(false);
        System.exit(0);
    }
}
```

La classe **ActionBouton** fournit la méthode **actionPerformed** que nous laisserons vide dans un premier temps.

```
class ActionBouton implements ActionListener {
    public void actionPerformed(ActionEvent e) {
    }
}
```

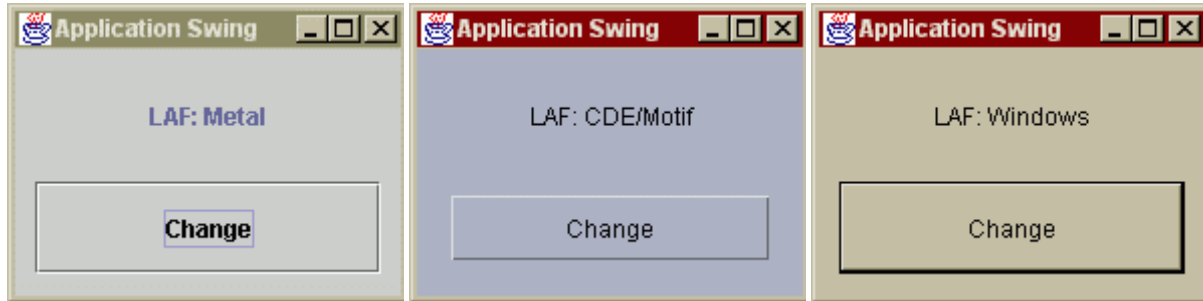
Méthode main

La méthode **main** de l'application se contente de créer une instance de la classe **SwingApplication**.

```
public static void main(String[] args) {
    System.out.println("Chargement en cours...");
    new SwingApplication();
}
```

Style des composants swing

Le package swing permet de choisir parmi plusieurs styles de composants; ainsi l'application précédemment écrite pourra prendre l'une des trois formes suivantes :



La classe UIManager

La classe UIManager fournit les méthodes de gestion des styles d'application. Retenons les méthodes :

- static UIManager.LookAndFeelInfo[] getInstalledLookAndFeels() : elle fournit la liste des styles disponibles sous forme de tableau.
- static LookAndFeel getLookAndFeel() : elle renvoie le style courant.
- static void setLookAndFeel(String className) : elle permet d'installer un style à partir du nom de la classe associée.

Mise en pratique

Nous ajoutons à la classe SwingApplication deux champs supplémentaires :

- un tableau nommé laf qui contiendra les styles disponibles
- un entier lafNo qui est le numéro du style courant

Ceci donne les déclarations suivantes à ajouter:

```
UIManager.LookAndFeelInfo laf[]; //styles disponibles
int lafNo=0; //numéro style courant
```

Le tableau des styles est initialisé dans le constructeur grâce à la méthode fournie par la classe UIManager.

```
//chargement des styles disponibles
laf=UIManager.getInstalledLookAndFeels();
```

Nous pouvons maintenant compléter la classe ActionBouton pour effectuer les changements de styles. A chaque appui sur le bouton change on incrémente le numéro du style courant en revenant à 0 après le dernier, puis on l'installe. La méthode actionPerformed devient donc :

```
public void actionPerformed(ActionEvent e) {
    //on passe au numéro de style suivant ou 0
    lafNo=(lafNo+1) % laf.length;
    //on essaie d'effectuer le changement
```

```

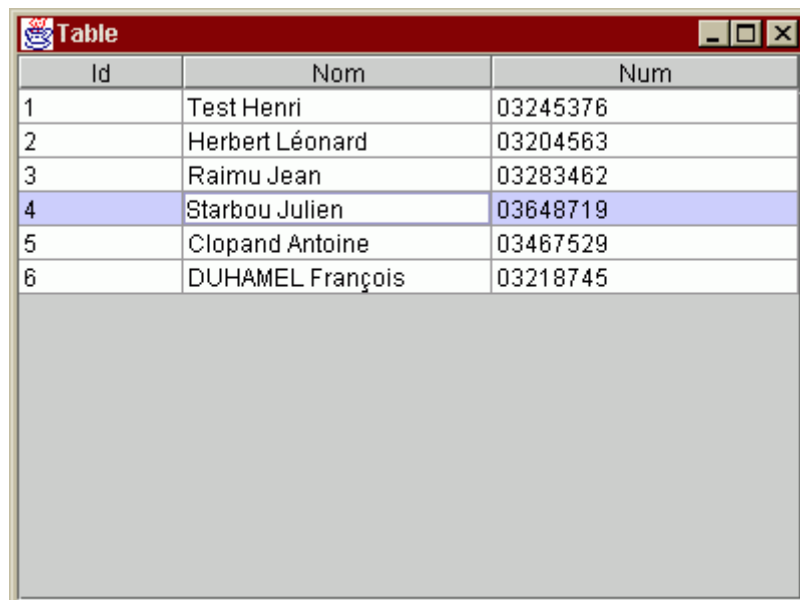
try {
    UIManager.setLookAndFeel(laf[lafNo].getClassName());
    SwingUtilities.updateComponentTreeUI(fen);
    //mise à jour du label contenant le nom du style
    nomStyle.setText("LAF: "+UIManager.getLookAndFeel().getName());
} catch (Exception exc) {}
fen.repaint();
}

```

La méthode `updateComponentTreeUI` fournie par la classe `SwingUtilities` permet d'appeler la méthode `updateUI` de chacun des composants contenus dans la fenêtre de façon à les mettre à jour pour le nouveau style actif.

Composant JTable

Le composant `JTable` permet de créer des tableaux. On l'utilise en relation avec une classe dérivée de **`AbstractTableModel`** qui permet de décrire le contenu du tableau (nombre de lignes et de colonnes, contenu des cellules).



| Id | Nom | Num |
|----|------------------|----------|
| 1 | Test Henri | 03245376 |
| 2 | Herbert Léonard | 03204563 |
| 3 | Raimu Jean | 03283462 |
| 4 | Starbou Julien | 03648719 |
| 5 | Clopand Antoine | 03467529 |
| 6 | DUHAMEL François | 03218745 |

Nous allons écrire une application qui affichera dans un tableau le contenu d'un fichier de données au format texte utilisant la virgule comme séparateur. La première ligne du fichier donnera les titres des colonnes du tableau.

La classe `DataFileTableModel`

Cette classe dérivée de la classe `AbstractTableModel` devra implémenter un certain nombre de méthodes déclarées abstraites. Elle utilisera pour ce faire les données contenues dans un fichier.

Déclaration et variables

Nous utiliserons trois variables:

- une chaîne de caractères contenant le nom du fichier de données
- un vecteur contenant les noms de colonnes
- un vecteur contenant les données à afficher dans le tableau

On obtient le code suivant:

```
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;
import java.util.*;

public class DataFileTableModel extends AbstractTableModel {
    protected Vector data;           //données
    protected Vector columnNames ;   //noms de colonnes
    protected String datafile;       //nom du fichier de données
}
```

Constructeur

Le constructeur récupère le nom du fichier de données puis initialise les deux champs de type Vector en lisant le fichier dans la méthode `initVectors`.

```
public DataFileTableModel(String f) {
    datafile=f;
    initVectors();
}

public void initVectors() {
    String ligne;
    data=new Vector();
    columnNames=new Vector();
    try {
        FileInputStream fin=new FileInputStream(datafile);
        BufferedReader br=new BufferedReader(
            new InputStreamReader(fin));
        // lecture des noms de colonnes (1ère ligne)
        ligne=br.readLine();
        StringTokenizer st1=new StringTokenizer(ligne, ",");
        while(st1.hasMoreTokens())
            columnNames.addElement(st1.nextToken());
        // lecture des données
        while ((ligne = br.readLine()) != null) {
            StringTokenizer st2=new StringTokenizer(ligne, ",");
            while(st2.hasMoreTokens())
                data.addElement(st2.nextToken());
        }
        br.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

La première ligne du fichier permet de déterminer le nombre et les noms des colonnes, elle sert donc à initialiser le champ `columnNames`. Les lignes suivantes permettent d'initialiser le champ `data`.

Méthodes à implémenter

Il nous reste à implémenter les méthodes qui permettront de construire et d'afficher le tableau.

Nombre de colonnes

Le nombre de colonnes est le nombre d'éléments du champ `columnNames`.

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

Nombre de lignes

Le nombre de lignes est obtenu en divisant le nombre d'éléments du champ `data` par le nombre de colonnes.

```
public int getRowCount() {  
    return data.size() / getColumnCount();  
}
```

Noms des colonnes

Les noms des colonnes sont contenus dans le champ `columnNames`.

```
public String getColumnName(int columnIndex) {  
    String colName="";  
    if (columnIndex<=getColumnCount())  
        colName=(String)columnNames.elementAt(columnIndex);  
    return colName;  
}
```

Type de contenu d'une colonne

Dans notre exemple, toutes les données sont des chaînes de caractères, donc de classe `String`.

```
public Class getColumnClass(int columnIndex){  
    return String.class;  
}
```

Possibilité d'édition des données

On indique ici quelles cellules sont modifiables. Dans notre exemple, aucune cellule ne le sera.

```
public boolean isCellEditable(int rowIndex, int columnIndex) {  
    return false;  
}
```

Contenu d'une cellule

On donne ici le contenu de chaque cellule sous forme d'objet. Dans notre exemple, ce contenu est une chaîne de caractères contenue dans le champ `data`. Il faut tenir compte du fait que les données sont placées les unes derrière les autres et donc recalculer leur position en fonction de la ligne et de la colonne de la cellule.

```
public Object getValueAt(int rowIndex, int columnIndex) {  
    return (String)data.elementAt(  
        (rowIndex*getColumnCount())+columnIndex);  
}
```

Changement du contenu d'une cellule

On indique ici comment prendre en compte les modifications du contenu des cellules effectuées par l'utilisateur. Dans notre exemple, aucune cellule n'est éditable, il n'y a donc rien à faire.

```
        public void setValueAt(Object aValue, int rowIndex, int
columnIndex) {
        }
```

L'application

Nous allons construire un JPanel qui contiendra un élément JTable associé au DataFileTableModel créé précédemment.

Il nous suffira ensuite d'écrire une méthode main qui créera une fenêtre contenant le JPanel créé.

Déclarations

On importe les packages nécessaires et on fait dériver la classe DataFileTable de la classe JPanel.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;

public class DataFileTable extends JPanel {
}
```

Constructeur

On récupère le nom du fichier de données (qui sera déterminé dans la méthode main), puis on crée le modèle de table et la table. La table est placée dans un JScrollPane pour obtenir des ascenseurs lorsque c'est nécessaire.

```
public DataFileTable(String dataFilePath) {
    JTable table;           // le tableau
    DataFileTableModel model; // le modèle
    //fonte
    Font f=new Font("SanSerif",Font.PLAIN,24);
    setFont(f);
    //gestionnaire de positionnement
    setLayout(new BorderLayout());
    //construction du modèle de remplissage à partir du fichier
    model = new DataFileTableModel(dataFilePath);
    //création du tableau
    table=new JTable();
    table.setModel(model);
    table.createDefaultColumnsFromModel();
    //scroller
    JScrollPane scrollpane=new JScrollPane(table);
    add(scrollpane);
}
```



```
}
```

Dimensions préférées

La méthode `getPreferredSize()` permet de fixer les dimensions préférées du tableau.

```
public Dimension getPreferredSize() {  
    return new Dimension(400, 300);  
}
```

Méthode main

La méthode principale permet de récupérer un nom de fichier en paramètre, puis de créer un tableau de type `DataFileTable` et de l'insérer dans une fenêtre.

```
public static void main(String args[]) {  
    //la fenêtre du programme  
    JFrame fen=new JFrame("Table");  
    //le tableau  
    DataFileTable tablo;  
    String nomFichier="datas.txt";  
    if (args.length>0) nomFichier=args[0];  
    tablo=new DataFileTable(nomFichier);  
    //configuration de la fenêtre  
    fen.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);  
    fen.setForeground(Color.black);  
    fen.setBackground(Color.lightGray);  
    fen.getContentPane().add(tablo,"Center");  
    fen.setSize(tablo.getPreferredSize());  
    //affichage  
    fen.setVisible(true);  
    //écouteur pour fermeture  
    fen.addWindowListener(new WindowCloser());  
}
```

La classe WindowCloser

Il reste à créer la classe `WindowCloser` chargée de la gestion des évènements liés à la fenêtre.

```
class WindowCloser extends WindowAdapter {  
    public void windowClosing(WindowEvent e) {  
        Window win=e.getWindow();  
        //effacer la fenêtre  
        win.setVisible(false);  
        //terminer le programme  
        System.exit(0);  
    }  
}
```

Exercices

1. Tableau

Compléter l'exemple d'utilisation du composant `JTable` pour permettre de modifier les données et enregistrer les modifications.

2. Composant Slider

Ecrire une application qui affiche un disque dont le rayon peut être modifié grâce à un composant `Slider`.