Les concepts de la programmation d'une interface graphique

P.O.O.

LMI 2 – Semestre 4 – Option Info

Année 2007-08

Plan du cours

Plan du cours

Les différentes API

- `java.awt`
- `javax.swing`
- `java.awt.event`
- `javax.swing.event`
- `...`

Les composants graphiques

Une interface graphique est constituée de composants :

- des composants simples : *button*, *radiobutton*, *checkbox*, listes déroulantes. . .
- des conteneurs ; *panel*, *frame*, *applet*, *window*. . .

Les composants graphiques élémentaires

Ils héritent tous, directement ou indirectement, de la classe `java.awt.Component`.

- `javax.swing.JButton`
- `javax.swing.JLabel`
- `javax.swing.JTextField`, `javax.swing.JTextArea`
- `java.awt.Canvas`

Les composants graphiques élémentaires

- javax.swing.JCheckBox

```
JCheckBox entree =
    new JCheckBox("Entree");
JCheckBox plat =
    new JCheckBox("Plat principal");
JCheckBox fromage =
    new JCheckBox("Fromage");
JCheckBox dessert =
    new JCheckBox("Dessert");
JCheckBox cafe = new JCheckBox("Cafe");
plat.setSelected(true);
dessert.setSelected(true);
```

Les composants graphiques élémentaires

- javax.swing.JRadioButton

```
JLabel etiquette = JLabel("Civilite :");
JRadioButton mme = new JRadioButton("Mme.");
JRadioButton mr = new JRadioButton("M.");
JRadioButton melle =
    new JRadioButton("Melle.");
ButtonGroup btg = new ButtonGroup();
btg.add(mme);
btg.add(mr);
btg.add(melle);
```

Les composants graphiques élémentaires

- javax.swing.JList

```
String[] data ={"one","two","three","four"};
JList dataList = new JList(data);

// par defaut:
// dataList.setSelectionMode(
//    ListSelectionModel.
          MULTIPLE_INTERVAL_SELECTION);
// on change
dataList.setSelectionMode(
   ListSelectionModel.SINGLE_SELECTION);

dataList.setSelectedIndex(1);//select "two"
dataList.getSelectedValue();//returns "two"
```

Les composants graphiques élémentaires

et d'autres . . .

- `javax.swing.JMenu`
- `javax.swing.JToolTip`
- . . .

Les conteneurs

Une application graphique se déroule dans une instance de `JFrame`.

Sur cette instance sont posés des composants grahiques

- conteneurs
- ou composants graphiques élémentaires

Les conteneurs

Ils héritent tous de la classe `Container` qui hérite de la classe `Component`.

La classe `Container` contient notamment la méthode `add(Component c)` pour poser des composants sur le conteneur.

Petit tour d'horizon des différents conteneurs :

- `JFrame`
- `JWindow`
- `JInternalFrame`
- `JApplet`
- `JPanel`
- `JTabbedPane`
- `JDialog`
- `JOptionPane`

Les gestionnaires de positionnement

Idée : On délègue à un composant la responsabilité du placement des objets graphiques sur le conteneur.

Les gestionnaires de positionnement

Idée : On délègue à un composant la responsabilité du placement des objets graphiques sur le conteneur.

Avantages :

- on ne calcule pas de position ;
- adaptation en fonction de la taille de la fenêtre principale ;
- plus flexible
- code beaucoup plus facile à maintenir

Les gestionnaires de positionnement

Les gestionnaires de positionnement obéissent à une politique.

- affichage autour d'un espace central : `BorderLayout`
- affichage en ligne horizontale : `FlowLayout`
- affichage vertical ou horizontal : `BoxLayout`
- affichage en tableau : `GridLayout`
- affichage paramétrable dans une grille : `GridBagLayout`

Tous ces gestionnaires sont définis dans `java.awt` (sauf `BoxLayout`).

Les gestionnaires de positionnement

Chaque conteneur a un gestionnaire de positionnement par défaut :

- le panneau central d'un `JFrame` est géré par un `BorderLayout`
- un `Jpanel` est géré par un `FlowLayout`
- ...

On peut associer un autre gestionnaire de positionnement :

- via le constructeur du conteneur ;
- via la méthode `setLayout(LayoutManager l)` de la classe `Container`.

Créer une interface graphique

1. dessiner rapidement l'interface souhaitée ;
2. la découper en unité logique de placement : les unités peuvent être côte à côte ou emboitées
3. chaque unité deviendra un conteneur géré par le layout manager qui conviendra ;
4. créer les conteneurs les plus internes ;
5. poser les composants élémentaires sur ces composants ;
6. puis emboiter en remontant vers la fenêtre principale.

La gestion des évènements

Idée :

1. les composants graphiques écoutent les évènements qui se passent sur eux ;
2. ils préviennent le composant à qui est déléguée la gestion de la réaction.

La gestion des évènements

Idée :

1. les composants graphiques écoutent les évènements qui se passent sur eux ;
2. ils préviennent le composant à qui est déléguée la gestion de la réaction.

Avantages :

- un même gestionnaire pour plusieurs contrôles graphiques
- découplage interface graphique / modélisation

La gestion des évènements

À chaque type d'évènement est associée une interface.

La gestion des évènements

À chaque type d'évènement est associée une interface.

Elle définit les messages qu'elle peut recevoir pour être informée des évènements.

La gestion des évènements

À chaque type d'évènement est associée une interface.

Elle définit les messages qu'elle peut recevoir pour être informée des évènements.

On prévient le contrôle grahique que pour tel type d'évènement, il devra prévenir tel objet.

```
monBtn.addActionListener(new GereBouton()) ;
```

La gestion des évènements

À chaque type d'évènement est associée une interface.

Elle définit les messages qu'elle peut recevoir pour être informée des évènements.

On prévient le contrôle grahique que pour tel type d'évènement, il devra prévenir tel objet.

```
monBtn.addActionListener(new GereBouton()) ;
```

Un contrôle graphique peut prévenir plusieurs objets d'un même évènement.

Les différents écouteurs d'évènement

- `ComponentListener`
- `FocusListener`
- `KeyListener`
- `MouseListener`
- `MouseMotionListener`
- `MouseWheelListener`

Ces écouteurs sont supportés par tous les composants.

Les différents écouteurs d'évènement

- `ActionListener`
- `ChangeListener` : `Button`, `RadioButton`, `CheckBox`, `ComboBox`,...
- `ItemListener`
- `CaretListener` : `TextArea`, `TextField`, ...
- `ListSelectionListener`
- `WindowListener`

Exemple 1

```
...
JButton btn = new JButton("OK");
btn.addActionListener(new GereBouton());
...
public class GereBouton
       implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println("Coucou");
  }
}
```

Exemple 2

```
public class UnExemple extends JFrame{
  public UnExemple(String titre){
    super(titre);
    Container cont = getContentPane();
    cont.setLayout(new FlowLayout());
    JButton jaune = new JButton("jaune");
    JButton rouge = new JButton("rouge");
    cont.add(jaune);
    cont.add(rouge);
    Gere2Boutons gb = new Gere2Boutons(this,jaune,rouge);
    jaune.addActionListener(gb);
    rouge.addActionListener(gb);
    setVisible(true);
  }
```

Exemple 2

```
  public void changeCouleur(Color c){
    getContentPane().setBackground(c);
  }
} //UnExemple
```

Exemple 2

```
public class Gere2Boutons
    implements ActionListener{
  private JFrame fenetre;
  private JButton jaune;
  private JButton rouge;
  public Gere2Boutons(JFrame fenetre,
          JButton jaune, JButton rouge){
  ...
  }
```

Exemple 2

```
  public void actionPerformed(ActionEvent e){
    if (e.getSource() == jaune)
      fenetre.changeCouleur(Color.yellow);
    else if (e.getSource() == rouge)
      fenetre.changeCouleur(Color.red);
    else
      System.out.println("y a un probleme!");
  }
}//Gere2Boutons
```

Exemple 2

```
public class Test{
  public static void main(String[] args){
    new UnExemple("Et c'est parti!");
  }
}
```