Interface graphiques

Java Swing

Principes de base

- Des composants graphiques
  (exemple: JFrame, JButton …)
  - Hiérarchie de classes
- Des événements et les actions à effectuer
  (exemple presser un bouton)
- (Et d'autres choses…)

Principes

- Définir les composants (instance de classes)
- Les placer à la main (layout Manager) dans un JPanel ou un content pane ou en utilisant des outils comme eclipse ou netbeans
- Définir les actions associées aux événements (Listener) et les associer aux composants graphiques

Principes

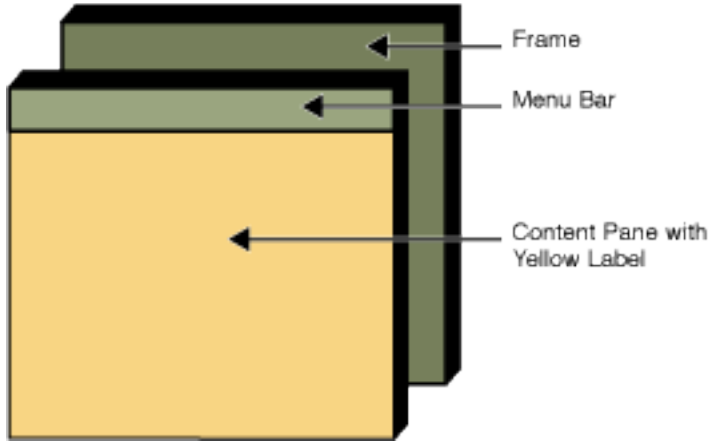
- Dans une interface graphique, le programme réagit aux interactions avec l'utilisateur
- Les interactions génèrent des événements
- Le programme est dirigé par les événements (event-driven)

Afficher…

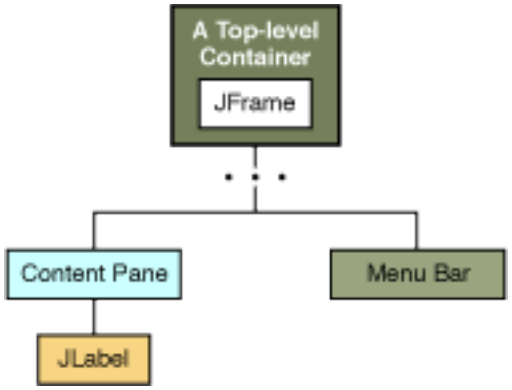
- Pour pouvoir être affiché, il faut que le composant soit dans un top-level conteneur:

  (JFrame, JDialog et JApplet)
- Hiérarchie des composants: arbre racine top-level

Exemple



- Correspond à la hiérarchie



Le code

```
import java.awt.*;
import javax.swing.*;

public class TopLevel {
    /**
     * Affiche une fenêtre JFrame top level
     * avec une barre de menu JMenuBar verte
     * et un JLabel jaune
     */
    private static void afficherMaFenetre() {
            //créer la Jframe
            //créer la JMenuBar
           //créer le Jlabel
           // mettre le JMenuBar et le Jlable dans la Jframe
           //afficher la Jframe
    }
}
```

Le code

```
//Creer la JFrame
  JFrame frame = new JFrame("TopLevelDemo");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  //Créer la JMenuBar
  JMenuBar greenMenuBar = new JMenuBar();
  greenMenuBar.setOpaque(true);
  greenMenuBar.setBackground(new Color(0, 200, 0));
  greenMenuBar.setPreferredSize(new Dimension(200, 20));
  //Créer le JLabel
  JLabel yellowLabel = new JLabel();
  yellowLabel.setOpaque(true);
  yellowLabel.setBackground(new Color(250, 250, 0));
  yellowLabel.setPreferredSize(new Dimension(200, 180));
  //mettre la JmenuBar et position le JLabel
  frame.setJMenuBar(greenMenuBar);
  frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
  //afficher...
  frame.pack();
  frame.setVisible(true);
```

Et le main

```
public class TopLevel {//afficherMaFenetre()
   public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            afficherMaFenetre();
         }
      });
   }
}
```

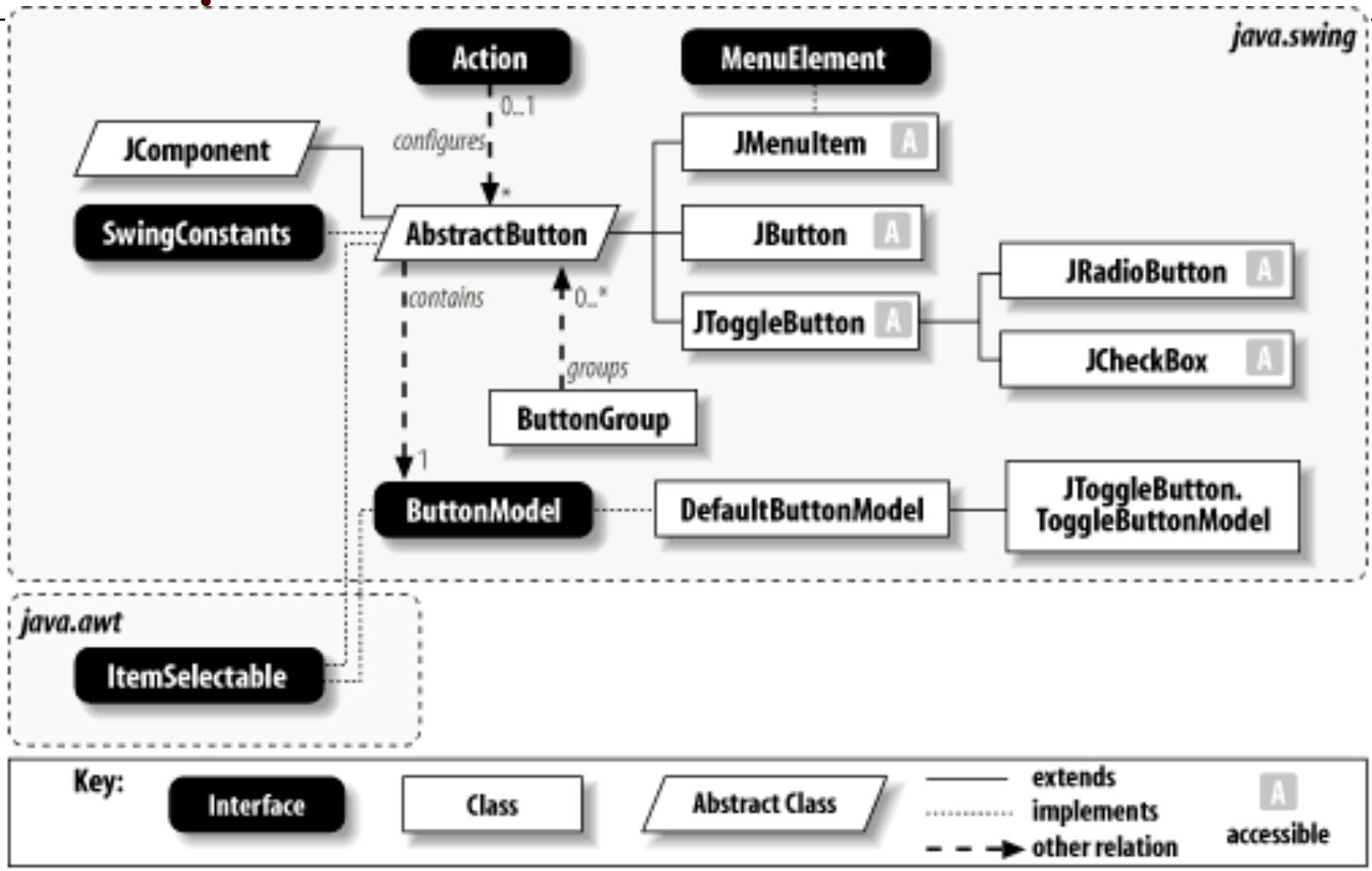
Evénements: principes

- Dans un système d'interface graphique:
  - Quand l'utilisateur presse un bouton, un "événement" est posté et va dans une boucle d'événements
  - Les événements dans la boucle d'événements sont transmis aux applications qui se sont enregistrées pour écouter.

Evénements

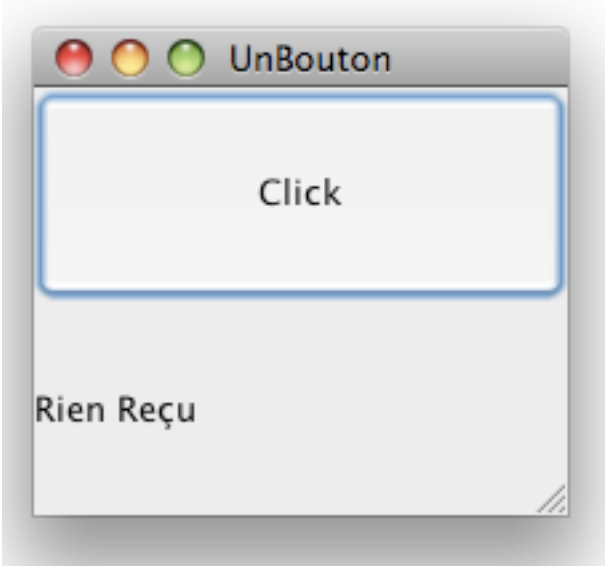
- Chaque composant génère des événements:
  - Presser un JButton génère un ActionEvent (système d'interface graphique)
    - Cet ActionEvent contient des infos (quel bouton, position de la souris, modificateurs…)
  - Un event listener (implémente ActionListener)
    - définit une méthode actionPerformed
    - S'enregistre auprès du bouton addActionListener
  - Quand le bouton est "clické",l'actionPerformed sera exécuté (avec l'ActionEvent comme paramètre)

Exemples Buttons



Un exemple

- Un bouton qui réagit



Le code:

- Un JButton
- Un JLabel
- Implementer ActionListener
  - actionPerfomed définit ce qui se passe quand le bouton est cliqué
- Placer le bouton et le label

Code:

```
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.Jcomponent;
import java.awt.Toolkit;
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;

public class UnBouton extends Jpanel implements ActionListener {
    JButton bouton;
    String contenu="Rien Reçu";
    JLabel label=new JLabel(contenu);
    int cmp=0;
    public UnBouton() { //…}
    public void actionPerformed(ActionEvent e) {//…}
    private static void maFenetre(){//…}
    public static void main(String[] args) {//…}
}
```

Code

```
public UnBouton() {
        super(new BorderLayout());
        bouton = new JButton("Click");
        bouton.setPreferredSize(new Dimension(200, 80));
        add(bouton, BorderLayout.NORTH);
        label = new JLabel(contenu);
                label.setPreferredSize(new Dimension(200, 80));
        add(label,BorderLayout.SOUTH);
        bouton.addActionListener(this);
   }
public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
        label.setText("clické "+ (++cmp)+ " fois");
  }
```

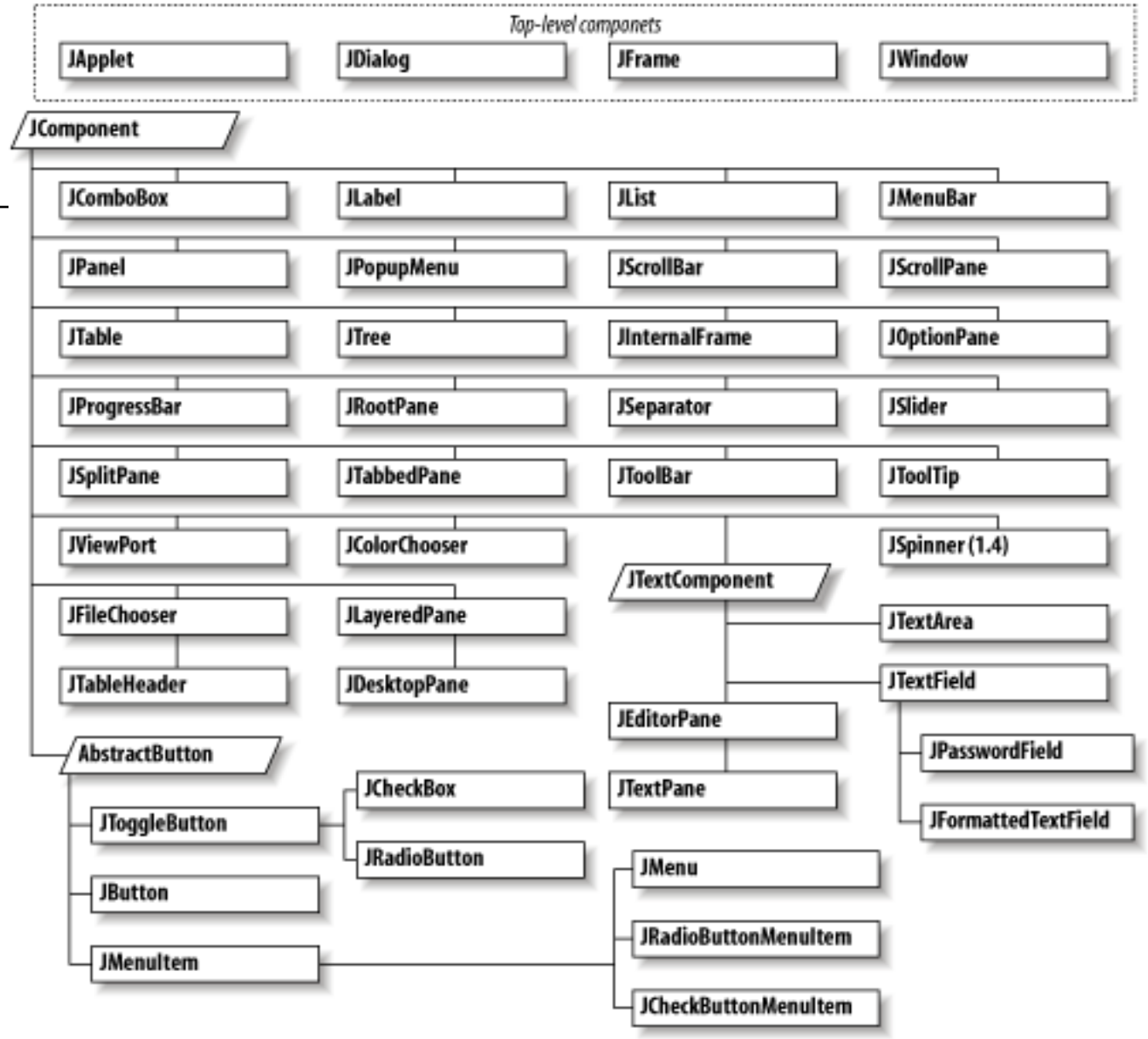
Code

```
private static void maFenetre() {
        JFrame frame = new JFrame("UnBouton");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent newContentPane = new UnBouton();
        newContentPane.setOpaque(true);
        frame.setContentPane(newContentPane);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        //Formule magique
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                maFenetre();
            }
        });
    }
```

Variante

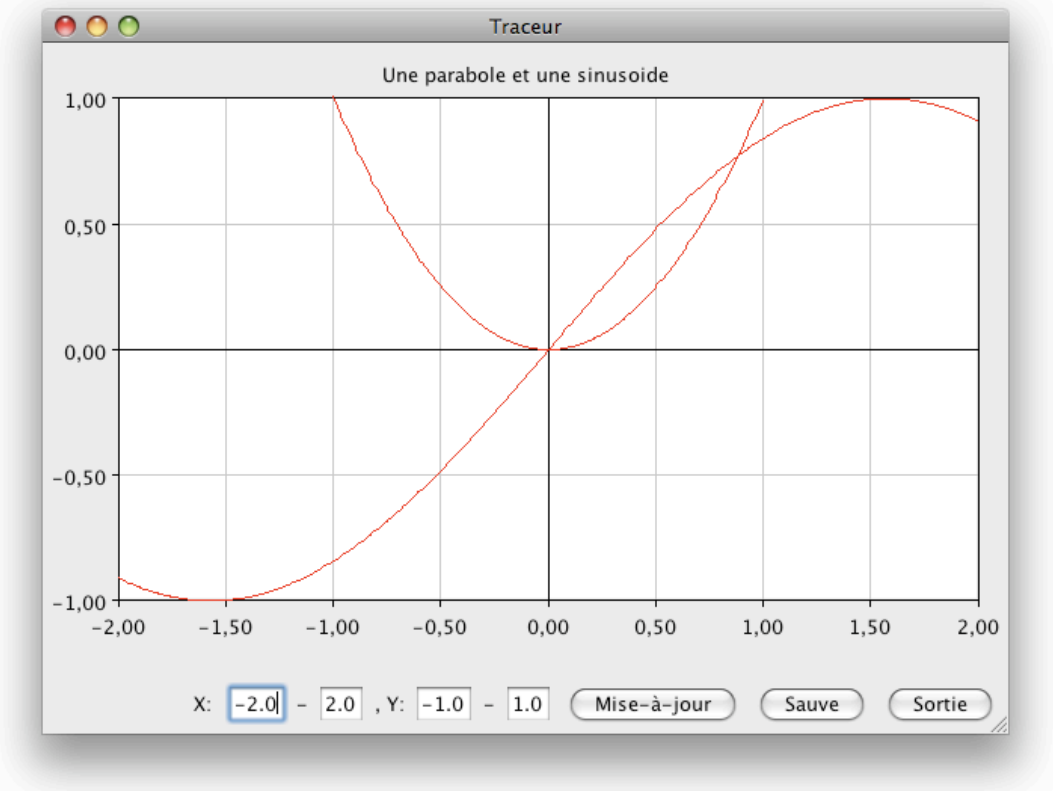
```
public class UnBoutonBis extends JPanel {
//…
 bouton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
              Toolkit.getDefaultToolkit().beep();
              label.setText("clické " + (++cmp) + " fois");
          } });
      }
//…
}
```

Hiérarchie des classes…


Top-level componets
JApplet
JDialog
JFrame
JWindow
JComponent
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree
JInternalFrame
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JToolBar
JToolTip
JViewPort
JColorChooser
JSpinner (1.4)
JFileChooser
JLayeredPane
JTableHeader
JDesktopPane
JTextComponent
JTextArea
JTextField
JEditorPane
JPasswordField
JTextPane
JFormattedTextField
AbstractButton
JCheckBox
JToggleButton
JRadioButton
JButton
JMenu
JMenuItem
JRadioButtonMenuItem
JCheckButtonMenuItem

Un exemple

- Un traceur de fonctions
  - Une interface graphique swing



Organisation

- GrapheSwing contient un GraphePanel extension de Jpanel
  - GraphePanel méthode paintComponent qui affiche le graphe de la fonction
    - Graphe est la classe contenant le gaphe et définissant une méthode draw pour l'affichage
    - Cette méthode appelle tracer de la classe abstraite Traceur
      - FonctionTraceur étend Traceur

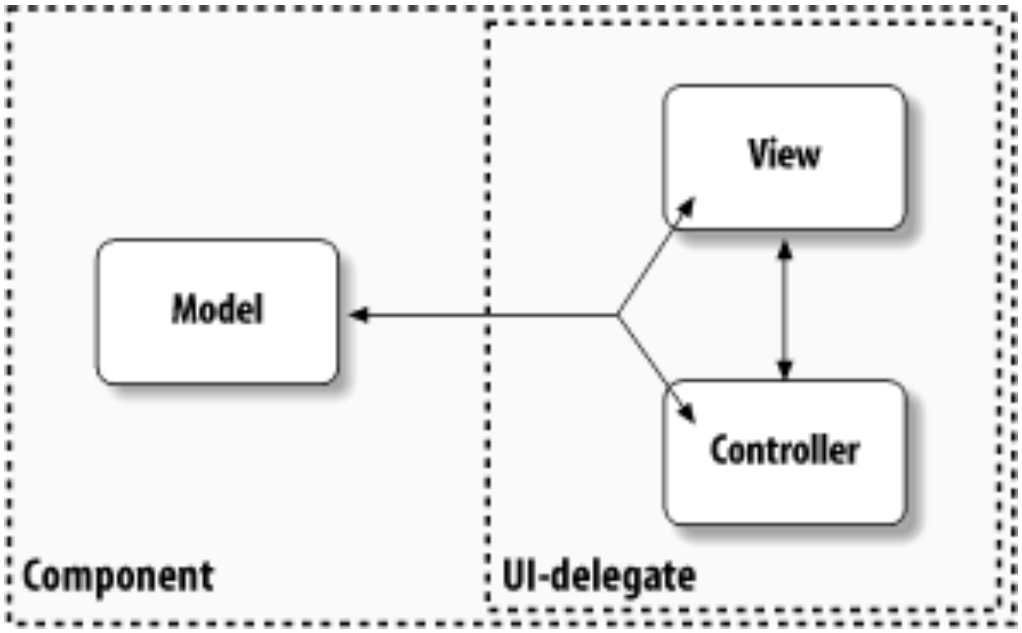
Le main

```
public static void main(String[] args) {  new GrapheSwing(unGraphe());}

  public static Graphe unGraphe() {
     PlotSettings p = new PlotSettings(-2, 2, -1, 1);
     p.setPlotColor(Color.RED);
     p.setGridSpacingX(0.5);
     p.setGridSpacingY(0.5);
     p.setTitle("Une parabole et une sinusoide");
     Graphe graphe = new Graphe(p);
     graphe.functions.add(new Parabole());
     graphe.functions.add(new FonctionTraceur() {
        public double getY(double x) {
           return Math.sin(x);
        }
        public String getName() {
           return "Sin(x)";
        }
     });
     return graphe;
  }
```

Composants

- Modèle Vue Contrôleur



Préliminaires…

- Lightweight et heavyweight composants
  - Dépendent ou non du système d' interface graphique
    - Lightweight écrit en Java et dessiné dans un heavyweight composant- indépendant de la plateforme
    - Les heavyweight composants s' adressent directement à l' interface graphique du système
  - (certaines caractéristiques dépendent du « look and feel »).

Look and feel

- Look and feel:

  Possibilité de choisir l'apparence de l'interface graphique.

  UIManager gère l'apparence de l'interface

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
    } catch (Exception e) { }

    new SwingApplication(); //Create and show the GUI.
}
```

Multithreading

- Attention au « modèle, contrôleur, vue » en cas de multithreading:
  - Tous les événements de dessin de l'interface graphiques sont dans une unique file d'event-dispatching dans une seule thread.
  - La mise à jour du modèle doit se faire tout de suite après l'événement de visualisation dans cette thread.

Plus précisément

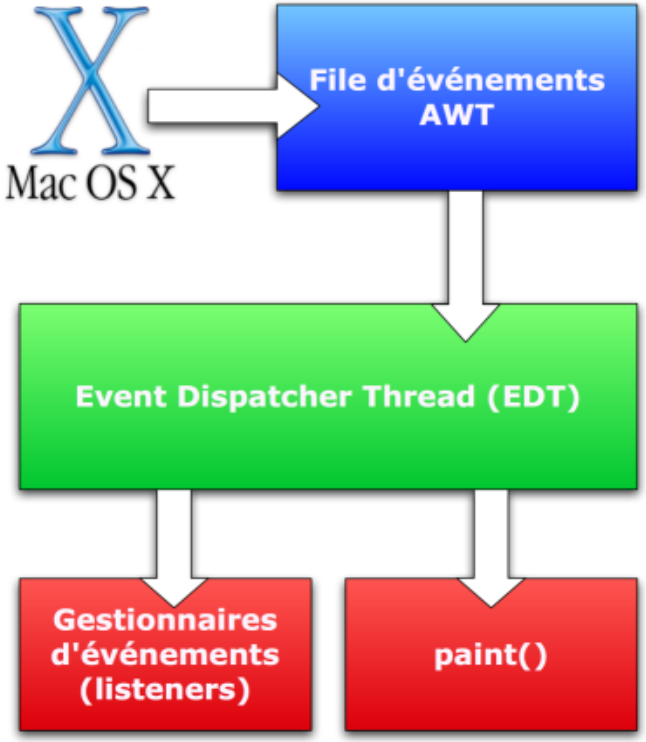
- Swing prend en charge la gestion des composants qui sont dessinés en code Java (lightweight)
- Les composants AWT sont eux liés aux composants natifs (heavyweight)
- Swing dessine le composants dans un canevas AWT et utilise le traitement des événements de AWT

Suite

Les threads

- Main application thread
- Toolkit thread
- Event dispatcher thread

- Toutes Les opérations d'affichage ont lieu dans une seule thread l'EDT



Principes

- Une tâche longue ne doit pas être exécutée dans l'EDT
- Un composant Swing doit s'exécuter dans l'EDT

Exemple

```
public void actionPerformed(ActionEvent e){
   try {
       Thread.sleep(4000);
} catch (InterruptedException e) {    }  }
```

Provoque une interruption de l'affichage pendant 4 secondes

Une solution

```
public void actionPerformed(ActionEvent e){
  try{
    SwingUtilities.invokeLater(newRunnable(
          { public void run() {
                //opération longue
                }
           });
   } catch (InterruptedException ie) {}
     catch (InvocationTargetException ite) {}
   }
}
```

Le main

- Normalement la création d'une fenêtre ne devrait avoir lieu que dans l'EDT:

```
public static void main(String[] args) {
        //Formule magique
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
                                        public void run() {maFenetre(); }
        });
    }
```

invokeLater crée une nouvelle thread qui poste la thread crée dans l'EDT

Attendre le résultat:

```
try {
        SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {
                show();
            }
        });
    } catch (InterruptedException ie) {
    } catch (InvocationTargetException ite) {
    }
```