Chapitre V
Interfaces graphiques

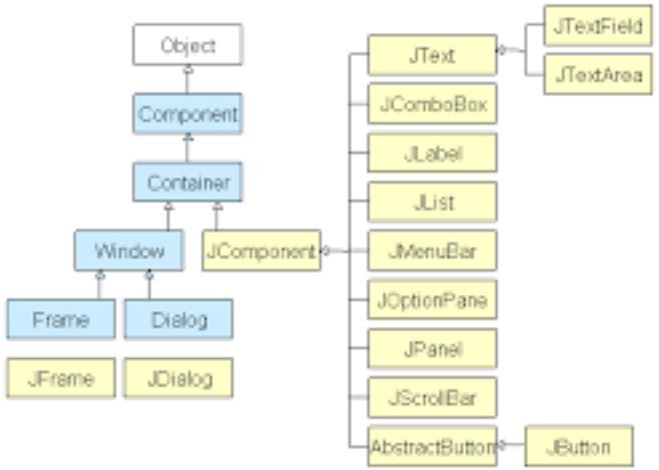
Swing

Principes de base

- Dans un système d'interface graphique, le programme doit réagir aux interactions avec l'utilisateur
- Les interactions génèrent des « événements » qui provoquent l'exécution de code
- Le programme est dirigé par les événements (event-driven)

Principes de base

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



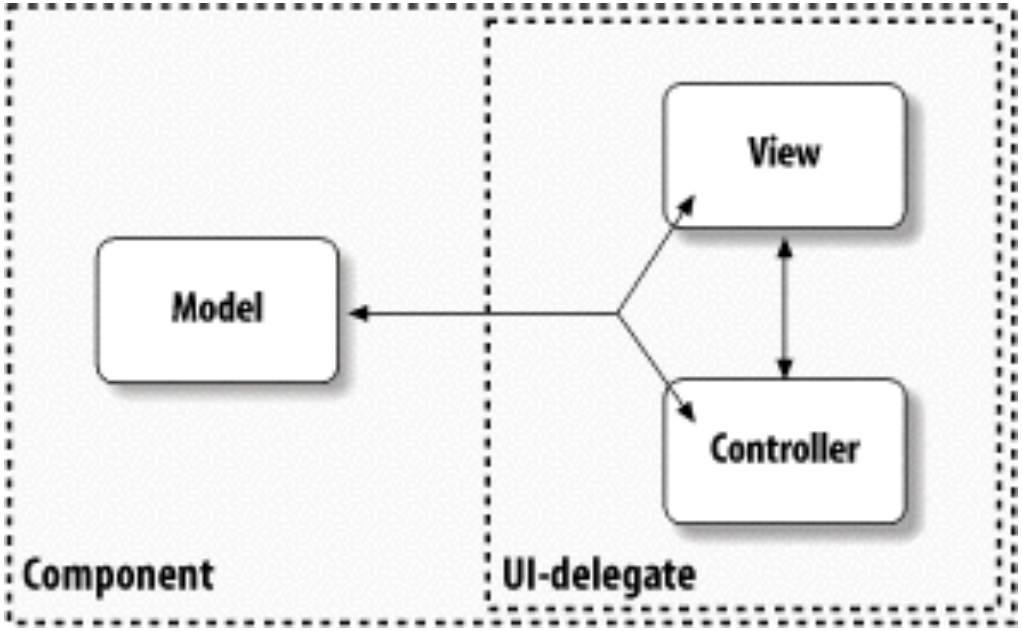
Principes de base

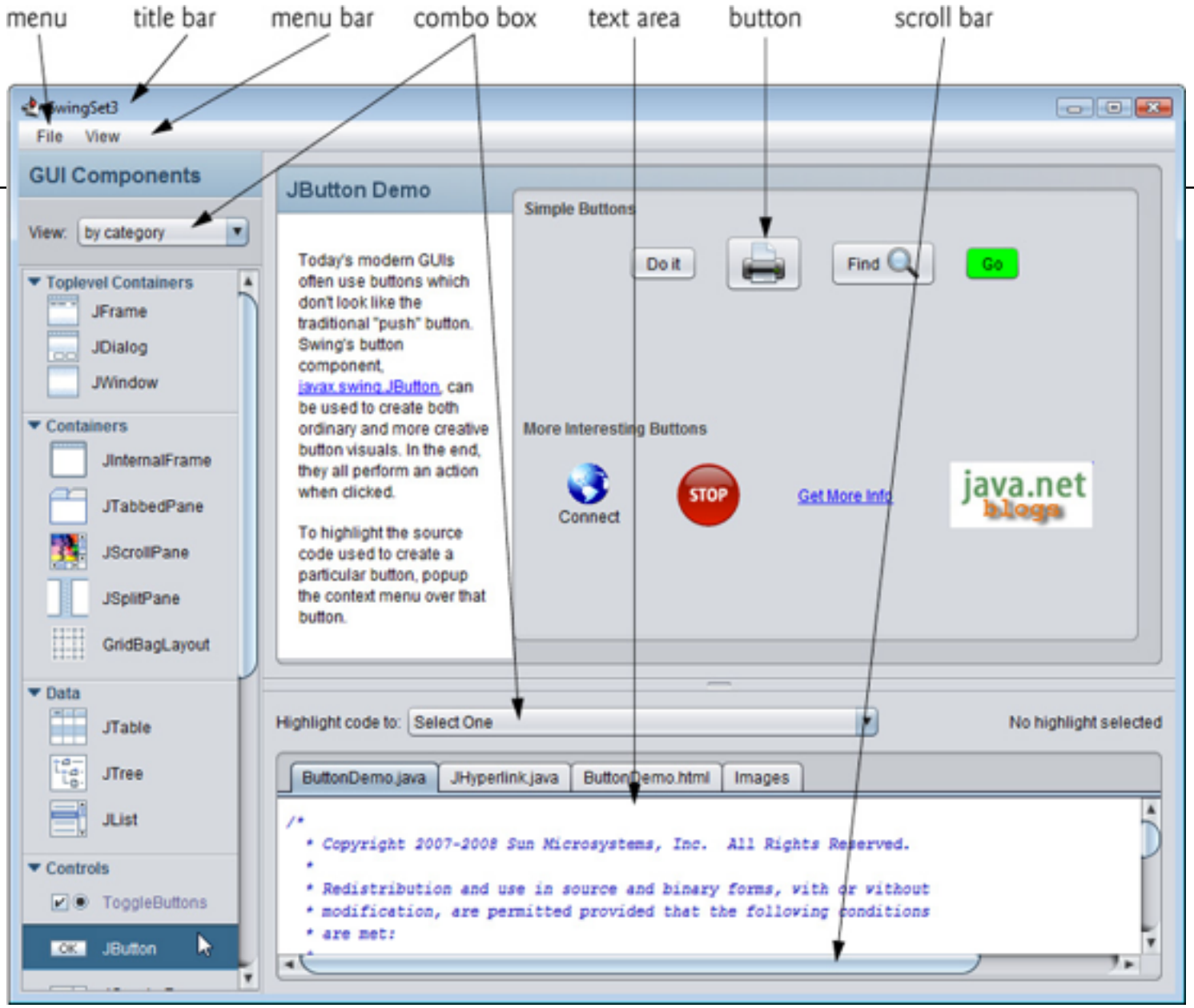
- Définir les composants (instance de classes)
- Les placer dans un JPanel ou un « content pane »
  - placement « à la main » avec un layout Manager ou en utilisant des outils comme eclipse ou netbeans
- Définir les actions associées aux événements (Listener) et les associer aux composants graphiques

Composants: JComponent

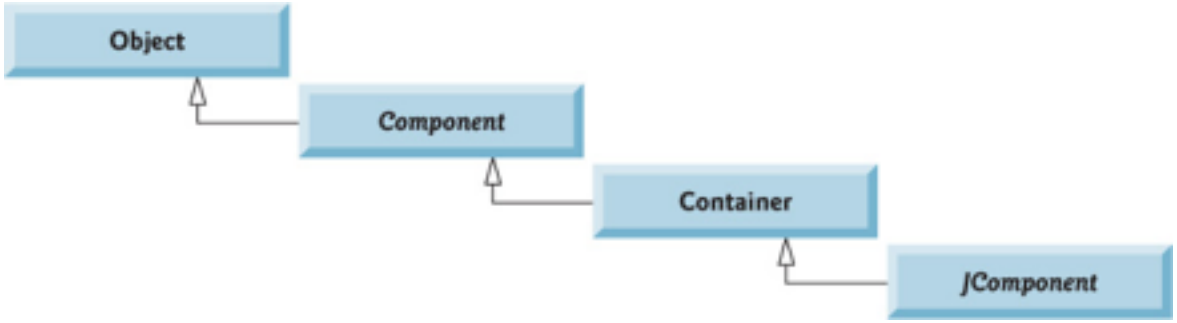
Composants

- Modèle Vue Contrôleur

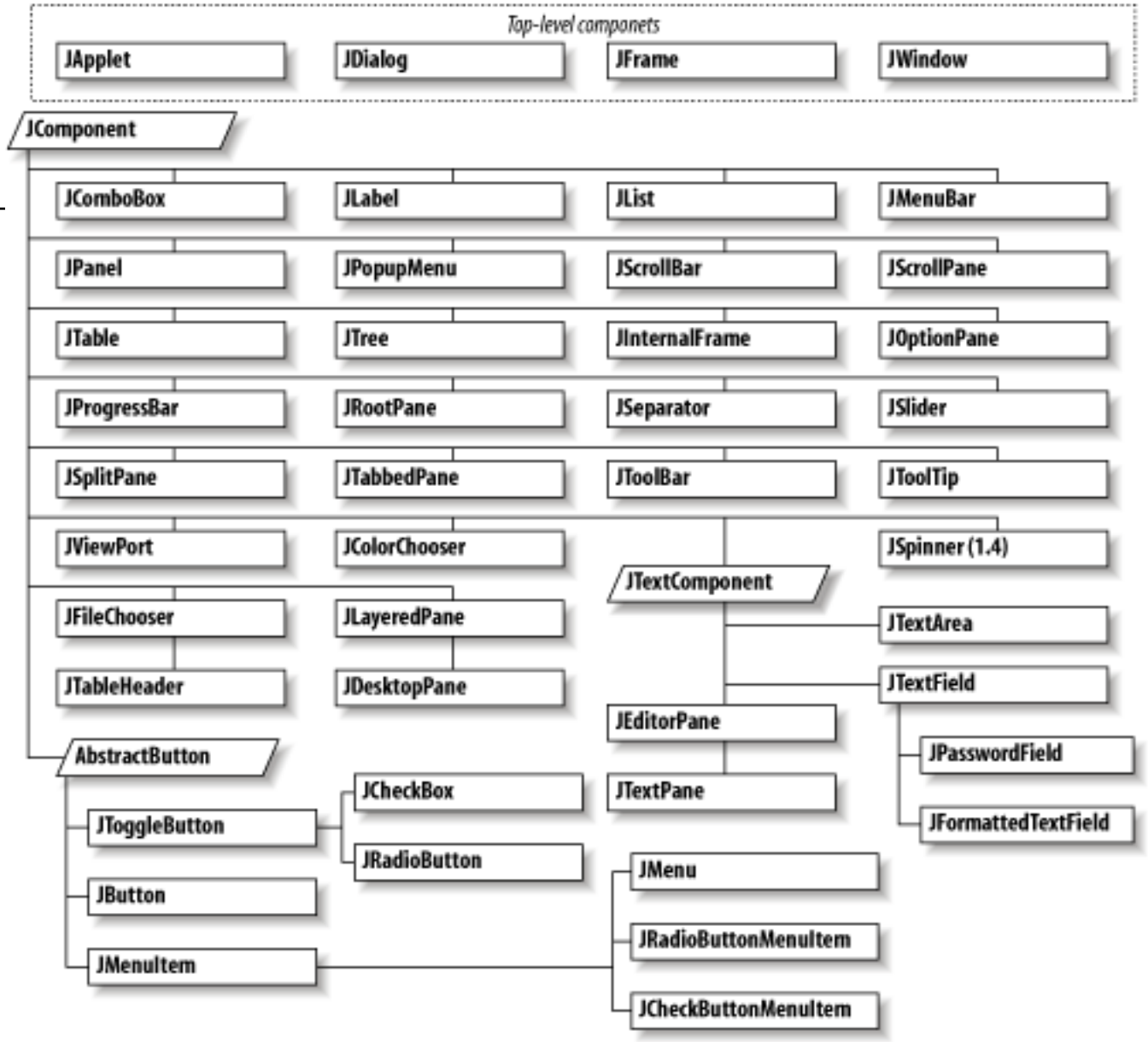



menu
title bar
menu bar
combo box
text area
button
scroll bar
SwingSet3
File View
GUI Components
View: by category
Toplevel Containers
JFrame
JDialog
JWindow
Containers
JInternalFrame
JTabbedPane
JScrollPane
JSplitPane
GridBagLayout
Data
JTable
JTree
JList
Controls
ToggleButtons
JButton
JButton Demo
Today's modern GUIs often use buttons which don't look like the traditional "push" button. Swing's button component, javax.swing.JButton, can be used to create both ordinary and more creative button visuals. In the end, they all perform an action when clicked.
To highlight the source code used to create a particular button, popup the context menu over that button.
Simple Buttons
Do it
Find
Go
More Interesting Buttons
Connect
STOP
Get More Info
java.net blogs
Highlight code to: Select One
No highlight selected
ButtonDemo.java
JHyperlink.java
ButtonDemo.html
Images
/*
* Copyright 2007-2008 Sun Microsystems, Inc. All Rights Reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:

Hiérarchie:



- les composants Swing descendent de JComponent (classe abstraite)

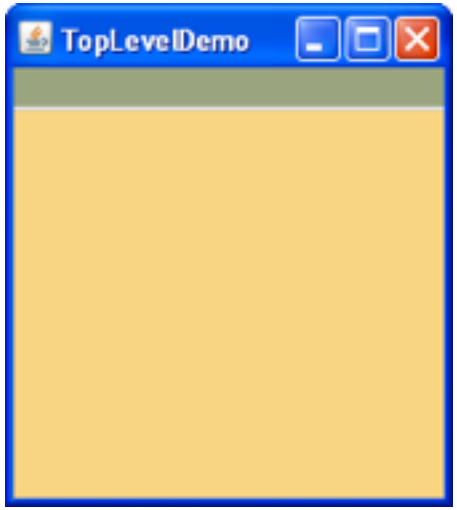

Top-level componets
JApplet
JDialog
JFrame
JWindow
JComponent
JComboBox
JLabel
JList
JMenuBar
JPanel
JPopupMenu
JScrollBar
JScrollPane
JTable
JTree
JInternalFrame
JOptionPane
JProgressBar
JRootPane
JSeparator
JSlider
JSplitPane
JTabbedPane
JToolBar
JToolTip
JViewPort
JColorChooser
JSpinner (1.4)
JFileChooser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

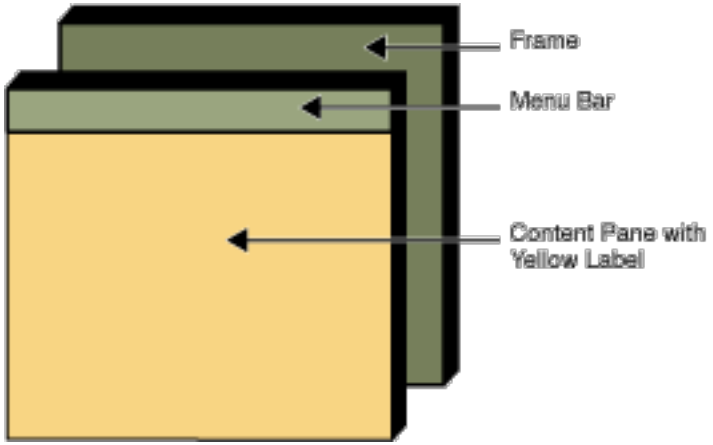
Afficher… top-level

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

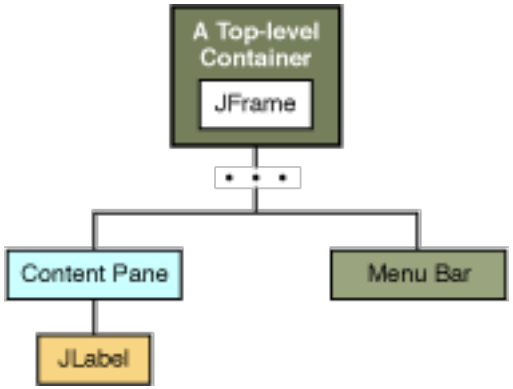
  (JFrame, JDialog et JApplet)
- Hiérarchie des composants: arbre racine top-level

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

Evénements: principes

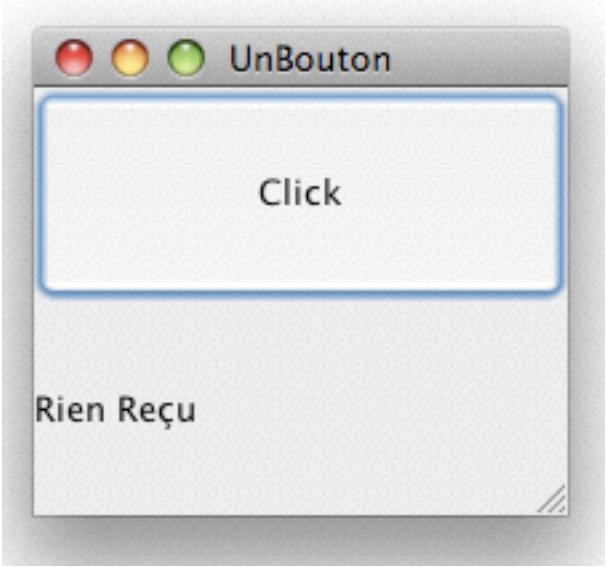
- Dans un système d'interface graphique:
  - Quand l'utilisateur presse un bouton, un "événement" est posté et va dans une boucle d'événements
  - Les événements dans la boucle d'événements sont transmis aux applications qui se sont enregistrées pour écouter.

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)

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");
        } });
   }
//…
}
```

Compléments

- AWT-Swing
- Multithreading

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 (swing)
    - Les heavyweight composants s'adressent directement à l'interface graphique du système (awt)
  - (certaines caractéristiques dépendent du « look and feel »).

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 composant dans un canevas AWT et utilise le traitement des événements de AWT

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

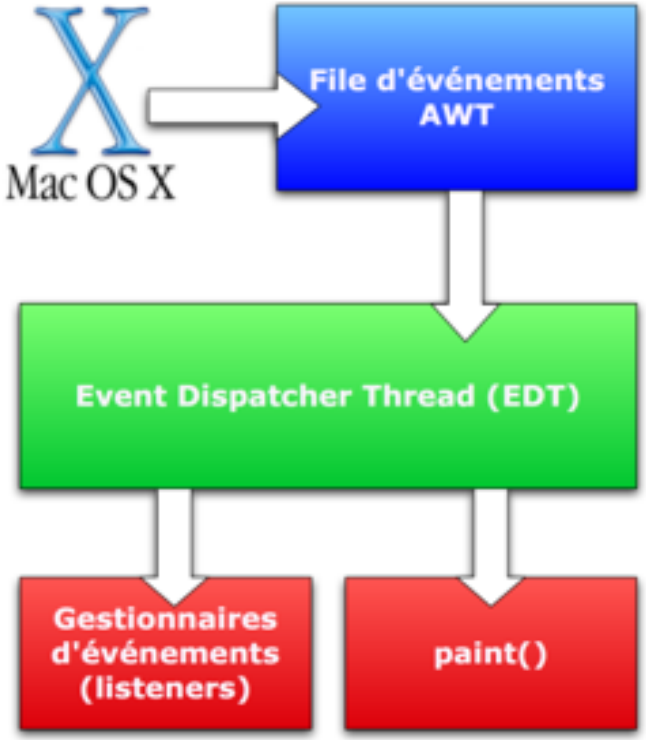
- Attention au « modèle, contrôleur, vue » en cas de multithreading:
  - Tous les événements de dessin de l'interface graphiques sont dans une unique file d'event-dispatching dans une seule thread.
  - La mise à jour du modèle doit se faire tout de suite après l'événement de visualisation dans cette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

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
  - threads initiales (code de l'application de départ)
  - event dispatcher thread: les événements
  - « workers » threads les tâches qui consomment du temps

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 code sera exécuté dans la thread des événements

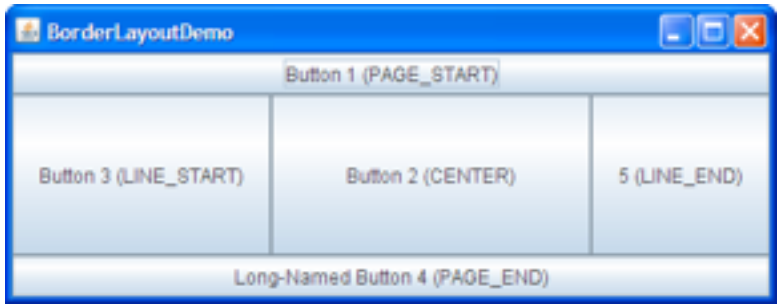
Attendre le résultat:

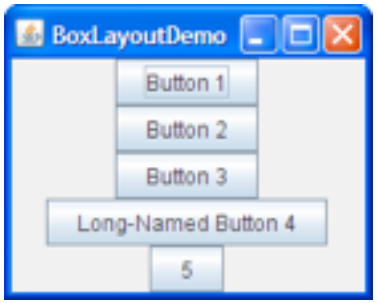
```
try {
  SwingUtilities.invokeAndWait(new Runnable() {
      public void run() {
                  show();
      }
  });
} catch (InterruptedException ie) {}
  catch (InvocationTargetException ite) {}
```

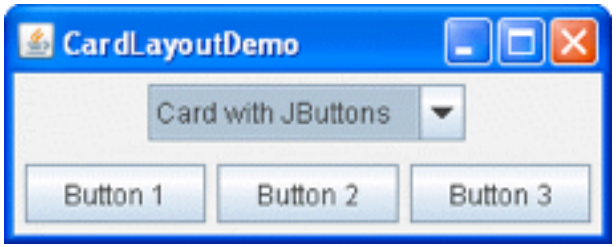
workers

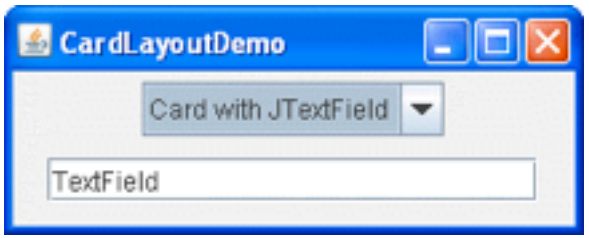
- La classe javax.swing.SwingWorker permet d'exécuter des tâches longues
  - doInBackground(): code à exécuter en background
  - done() sera exécuté dans Event Dispatch Thread quand doInBackground sera terminé
  - (il est aussi possible d'obtenir des résultats intermédiaire)

Layout manager



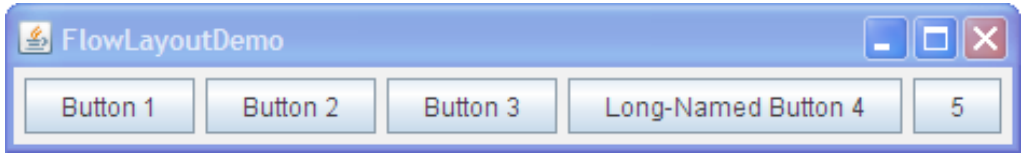


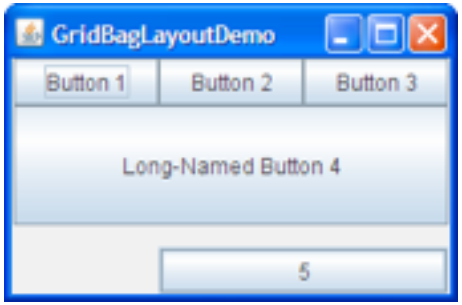


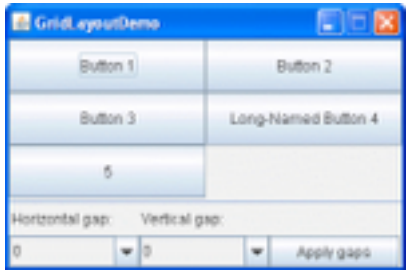


- Borderlayout
- BoxLayout (une ligne ou une colonne)
- CardLayout (les composants peuvent changer)

Layout manager







- FlowLayout (défaut)
- GridBagLayout
- GridLayout

On peut aussi placer directement dans le composant: setLayout(null) et setBounds

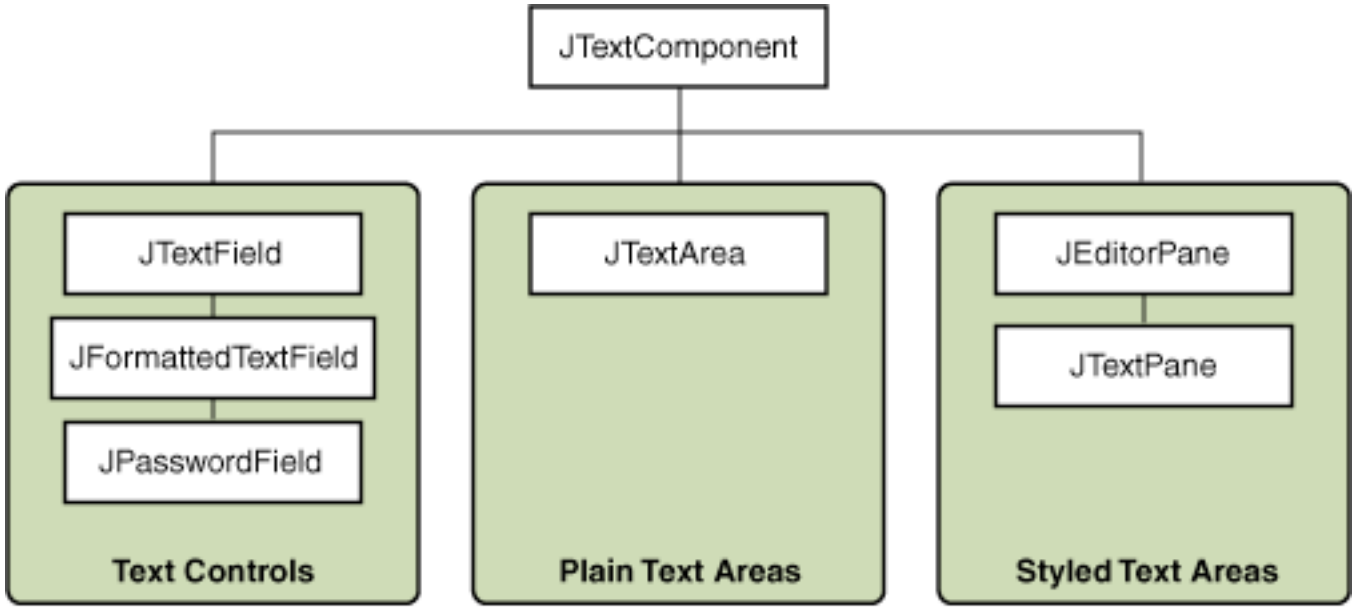
Un premier exemple: éditeur

Exemple JTextComponent

Modèle: document contenu du composant

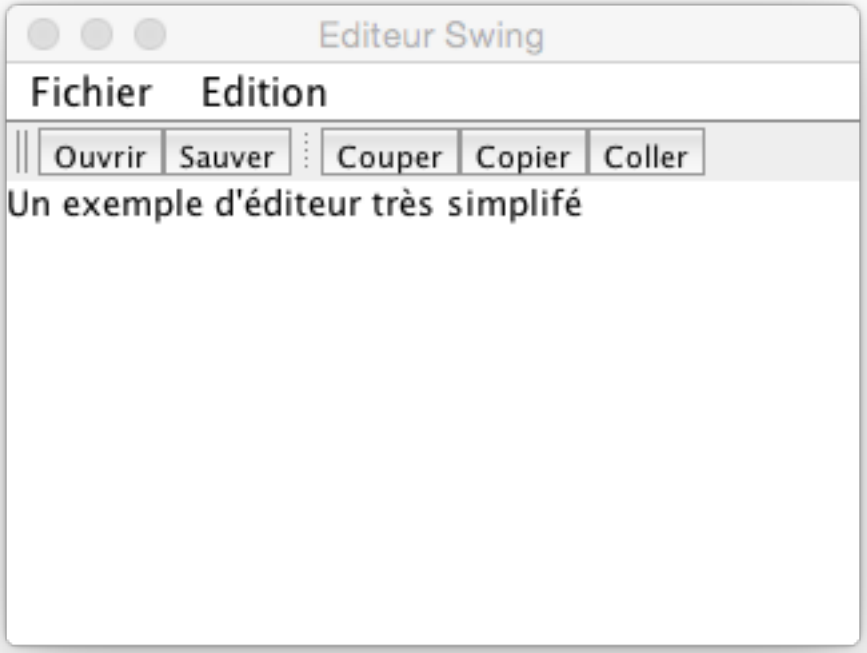
Vue: l'affichage du composant sur l'écran

Contrôleur: editor kit avec actions



Un éditeur simple

- un menu, une barre d'outil,
- actions:
  - couper, coller, copier, tout sélectionner provenant de DefaultEditorKit
  - ouvrir un fichier, sauvegarder un fichier avec un JFileChooser



Structure globale

Créer un JTextArea

Définir les différentes Actions

Créer une barre de menu

- associer les Actions à des boutons de la barre de menu

Créer un menu

- associer les Actions aux éléments de menu

SimpleEditeur

```
public SimpleEditeur() {
    super("Editeur Swing");
    textComp = createTextComponent();
    mesActions();
    Container content = getContentPane();
    content.add(textComp, BorderLayout.CENTER);
    content.add(createToolBar(), BorderLayout.NORTH);
    setJMenuBar(createMenuBar());
    setSize(320, 240);
}

// ici le JTextComponent en un JTextArea: plain text
protected JTextComponent createTextComponent() {
    JTextArea ta = new JTextArea();
    ta.setLineWrap(true);
    return ta;
}
```

Créations des actions

```
protected void mesActions() {
   Action a;
   a = textComp.getActionMap().get(DefaultEditorKit.cutAction);
   a.putValue(Action.SMALL_ICON, new ImageIcon("images/Couper.gif"));
   a.putValue(Action.NAME, "Couper");

   a = textComp.getActionMap().get(DefaultEditorKit.copyAction);
   a.putValue(Action.SMALL_ICON, new ImageIcon(« images/Copier.gif"));
   a.putValue(Action.NAME, "Copier");

   a = textComp.getActionMap().get(DefaultEditorKit.pasteAction);
   a.putValue(Action.SMALL_ICON, new ImageIcon("images/Coller.gif"));
   a.putValue(Action.NAME, "Coller");

   a = textComp.getActionMap().get(DefaultEditorKit.selectAllAction);
   a.putValue(Action.NAME, "Tout Sélectionner");
}
```

Barre d'outils

```
// JToolBar avec boutons
protected JToolBar createToolBar() {
  JToolBar bar = new JToolBar();
   // Ouvrir et Sauver
  bar.add(openAction).setText("Ouvrir");
  bar.add(saveAction).setText("Sauver");
  bar.addSeparator();
   // couper-coller
  bar.add(textComp.getActionMap().get(DefaultEditorKit.cutAction))
                .setText("Couper");
  bar.add(textComp.getActionMap().get(DefaultEditorKit.copyAction))
                .setText("Copier");
  bar.add(textComp.getActionMap().get(DefaultEditorKit.pasteAction))
                .setText("Coller");
       return bar;
}
```

Menu

```
protected JMenuBar createMenuBar() {
    JMenuBar menubar = new JMenuBar();
    JMenu file = new JMenu("Fichier");
    JMenu edit = new JMenu("Edition");
    menubar.add(file);
    menubar.add(edit);
    file.add(openAction);
    file.add(saveAction);
    file.add(new ExitAction());
    edit.add(textComp.getActionMap()
            .get(DefaultEditorKit.cutAction));
    edit.add(textComp.getActionMap()
            .get(DefaultEditorKit.copyAction));
    edit.add(textComp.getActionMap()
            .get(DefaultEditorKit.pasteAction));
    edit.add(textComp.getActionMap()
           .get(DefaultEditorKit.selectAllAction));
     return menubar;
 }
```

OpenAction

```
class OpenAction extends AbstractAction {
 public OpenAction() {
     super("Ouvrir", new ImageIcon("icons/open.gif"));
 }
      // avec un JFileChooser
 public void actionPerformed(ActionEvent ev) {
     JFileChooser chooser = new JFileChooser();
     if (chooser.showOpenDialog(SimpleEditeur.this) != JFileChooser.APPROVE_OPTION) {
          return;
     }
     File file = chooser.getSelectedFile();
     if (file == null) {return;}
     FileReader reader = null;
     try {
         reader = new FileReader(file);
         textComp.read(reader, null);
      } catch (IOException ex) {
       JOptionPane.showMessageDialog(SimpleEditeur.this,
           "Pas de fichier", "ERROR", JOptionPane.ERROR_MESSAGE);
      } finally {
          if (reader != null) {
          try {
              reader.close();
           } catch (IOException x) {}
      }
 }
```

SaveAction

```
class SaveAction extends AbstractAction {
   public SaveAction() {
       super("Save", new ImageIcon("icons/save.gif"));
     }
  public void actionPerformed(ActionEvent ev) {
       JFileChooser chooser = new JFileChooser();
       if (chooser.showSaveDialog(SimpleEditeur.this) != JFileChooser.APPROVE_OPTION) {
            return;
       }
       File file = chooser.getSelectedFile();
       if (file == null) {return;}
       FileWriter writer = null;
       try {
             writer = new FileWriter(file);
             textComp.write(writer);
         } catch (IOException ex) {
             JOptionPane.showMessageDialog(SimpleEditeur.this,
                     "Erreur", "ERROR", JOptionPane.ERROR_MESSAGE);
         } finally {
             if (writer != null) {
                 try {
                     writer.close();
                 } catch (IOException x) {}
             }
         }
       }
 }
```

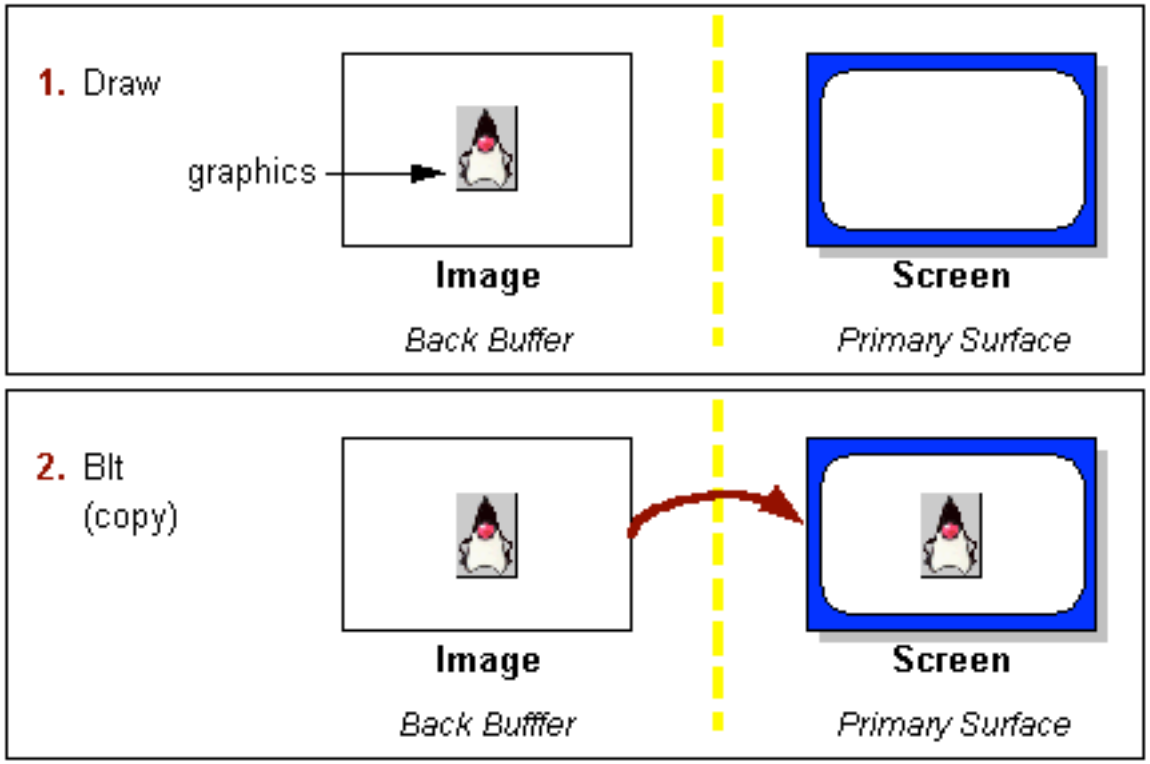
Le reste…

```
public class SimpleEditeur extends JFrame {
    private Action openAction = new OpenAction();
    private Action saveAction = new SaveAction();
    private JTextComponent textComp;
    private static void createAndShowGUI() {
        SimpleEditeur editeur = new SimpleEditeur();
        editeur.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        editeur.setVisible(true);
    }
    //le main
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                UIManager.put("swing.boldMetal", Boolean.FALSE);
                createAndShowGUI();
            }
        });
    }
```

deuxième exemple: painting

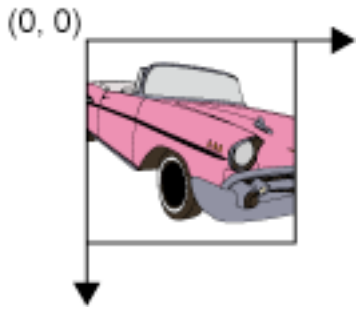
- application graphique:
  - Graphics est la classe des contextes graphiques (paint)
    - contient les méthodes permettant d'afficher des objets graphiques (exemple drawImage, drawLine..)
  - paintComponent(Graphics g) à redéfinir pour réaliser afficher les composants graphiques

Double Buffering



Graphics et Graphics2D

- un sytème de coordonnées: espace de l'utilisateur- espace du device
- les attributs du « rendu »



Graphics et Graphics2D

- Géométrie:
  - points
  - lignes
  - rectangles
  - courbes
  - formes arbitraires
- Texte
- Images
- Print

Painting

- paint est appelée à chaque nouveau « rendu » de d'un composant graphique:
  - public void paint(Graphics g) de awt
- extension dans Jcomponent:
  - protected void paintComponent(Graphics g)
  - protected void paintBorder(Graphics g)
  - protected void paintChildren(Graphics g)

Painting

- chaque composant a son look and feel réalisé par un UI séparé, pour un « paint » d'un composant on aura:
  - appel de paintComponent
  - appel de ui.update()
    - si le composant est opaque, ui.update() remplit le fond avec la couleur de fond de ui.paint()
    - ui.paint() fait le rendu du contenu
  - une réécriture de paintComponent doit en général invoquer super.paintComponent()

Painting

- méthode void paint(Graphics g) de java.awt.component
- paint de JComponent appelle:
  - protected void paintComponent(Graphics g)
  - protected void paintBorder(Graphics g)
  - protected void paintChildren(Graphics g)
- pour afficher des objets graphiques on redéfinira paintComponent (avec appel à super.paintComponent)

Cercle bleu

- un cercle bleu:
- un clic de la souris déplace le cercle vers la position du clic
- un déplacement de la souris (pressée) déplace le cercle

Comme d'habitude

```
public class ExampleDraw {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
    private static void createAndShowGUI() {
        JFrame f = new JFrame("Cercle Bleu");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new MonPanel());
        f.setSize(250,250);
        f.setVisible(true);
    }
}
```

Le panel

```
class MonPanel extends JPanel {
    CercleBleu cercleBleu = new CercleBleu();

    public MonPanel() {
        setBorder(BorderFactory.createLineBorder(Color.red));
        addMouseListener(new MouseAdapter(){
            public void mousePressed(MouseEvent e){
                moveCercle(e.getX(),e.getY());
            }
        });
        addMouseMotionListener(new MouseAdapter(){
            public void mouseDragged(MouseEvent e){
                moveCercle(e.getX(),e.getY());
            }
        });

    }
```

Déplacer le cercle

```
private void moveCercle(int x, int y){
    final int CURR_X = cercleBleu.getX();
    final int CURR_Y = cercleBleu.getY();
    final int CURR_W = cercleBleu.getWidth();
    final int CURR_H = cercleBleu.getHeight();
    final int OFFSET = 1;
    if ((CURR_X!=x) || (CURR_Y!=y)) {
        // le cercle a bougé: le redessiner
        repaint(CURR_X,CURR_Y,CURR_W+OFFSET,CURR_H+OFFSET);
        // mise à jour coordonnées
        cercleBleu.setX(x);
        cercleBleu.setY(y);
        // redessiner dans la nouvelle position
        repaint(cercleBleu.getX(), cercleBleu.getY(),
                cercleBleu.getWidth()+OFFSET,
                cercleBleu.getHeight()+OFFSET);
    }
}
```

MonPanel (fin)

```
    public Dimension getPreferredSize() {
        return new Dimension(250,200);
    }

    public void paintComponent(Graphics g) {
       super.paintComponent(g);
        g.drawString("Un Panel pour le cercle bleu" ,
      10,20);
        cercleBleu.paintCercle(g);
    }
}// fin de MonPanel
```

CercleBleu

```
class CercleBleu{

    private int xPos = 50;
    private int yPos = 50;
    private int width = 20;
    private int height = 20;
    public void setX(int xPos){ this.xPos = xPos;}
    public int getX(){return xPos;}
    public void setY(int yPos){this.yPos = yPos;}
    public int getY(){return yPos;
    public int getWidth(){return width;}
    public int getHeight(){return height;}
    public void paintCercle(Graphics g){
        Graphics2D g2=(Graphics2D)g;
        g2.setPaint(Color.BLUE);
        g2.fill (new Ellipse2D.Double(xPos, yPos, width, width));
    }
}
```