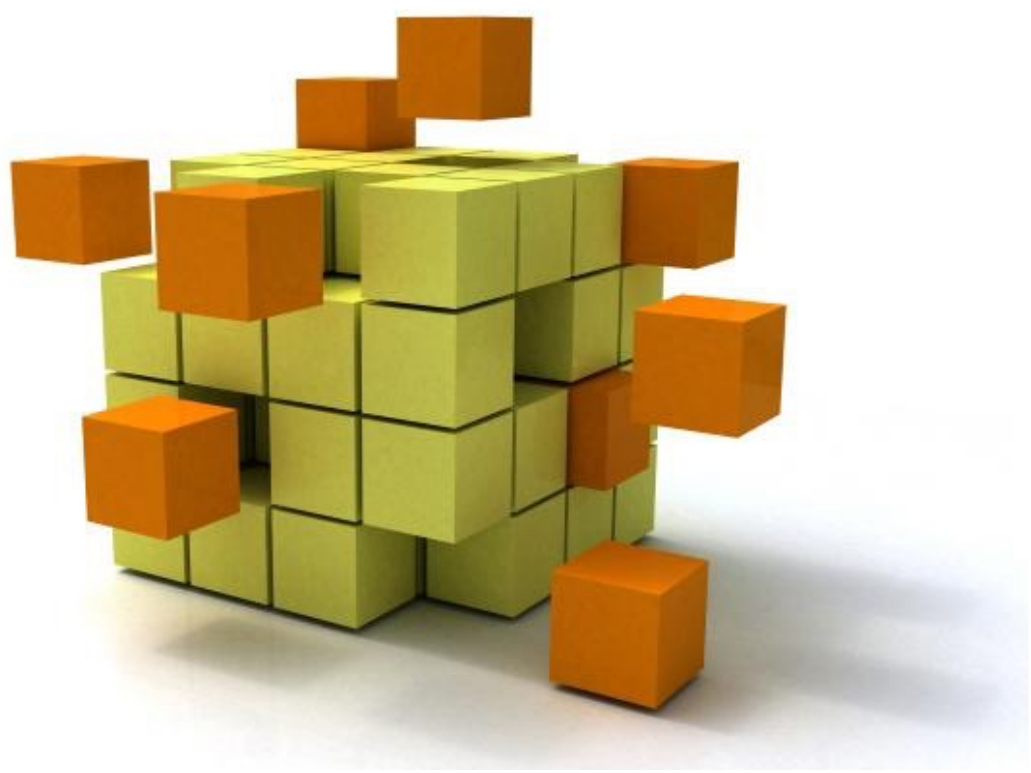


Applications concurrentes, mobiles et réparties en java

S7 Appro – Inf A3 EG

Stéphane Derrode, Alexandre Saïdi, Bât E6, 2ième étage
stephane.derrode@ec-lyon.fr

Organisation de l'AF

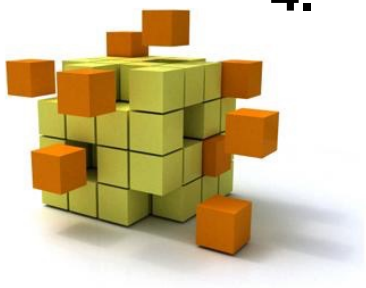
Grands chapitres

1. **Apprentissage de Java** : 4h de cours, 4h de TP, 2h d'autonomie – Stéphane Derrode

2. **IHM en Java** : 4h de cours, 8h de TP, 4h autonomie (BE noté #1) – Stéphane Derrode

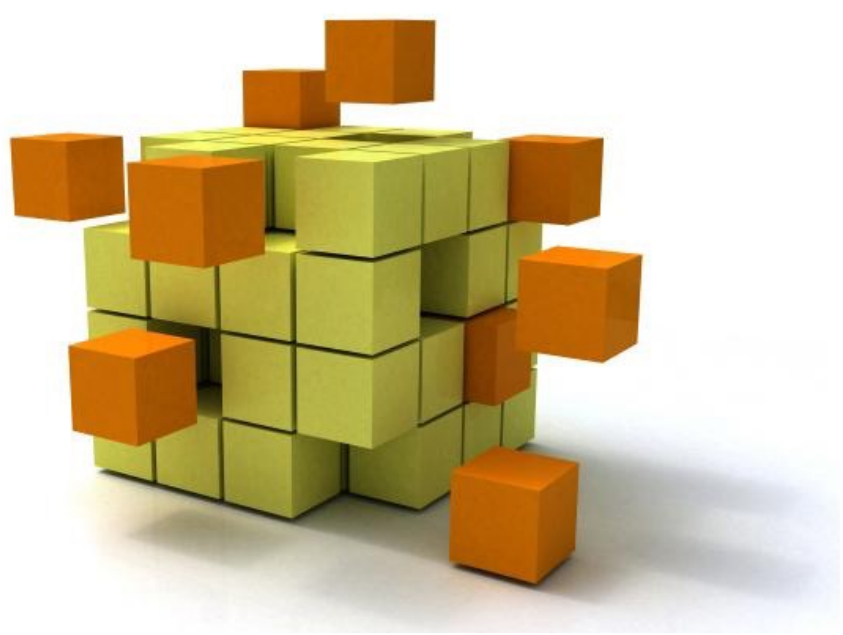
3. **Prog. concurrente et distribuée** : 4h de cours, 4h de TP, 4h d'autonomie (BE noté #2) – Alexandre Saïdi

4. **Prog. mobile** : 4h de cours, 4h de TP, 2h d'autonomie (BE noté #3) – Stéphane Derrode



2- IHM en Java

1. Introduction à la programmation des IHM
2. Les composants
3. Arbre d'instanciation et positionnement
4. Graphiques
5. La gestion des évènements
6. Classe "Bipbip"



1- Introduction

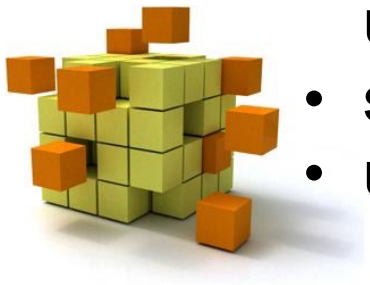
1- Introduction

Principe

- un programme affiche sur l'écran un certain nombre d'objets d'interface (fenêtres, menus, boutons...)
- à chaque objet est lié un traitement à exécuter à l'arrivée d'un événement sur l'objet
- à la suite d'une action de l'utilisateur (souris, clavier), le système fabrique un évènement, détermine l'objet concerné et lui envoie l'événement
- l'objet active le traitement associé

Conséquences

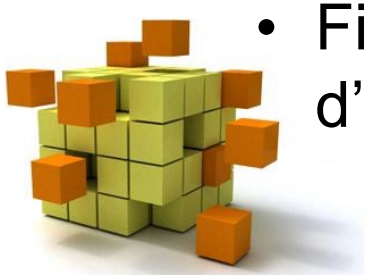
- plus d'enchaînement prédéterminé des actions utilisateur
- structure de programme non linéaire, plus complexe
- utilisation de bibliothèques d'objets d'interface

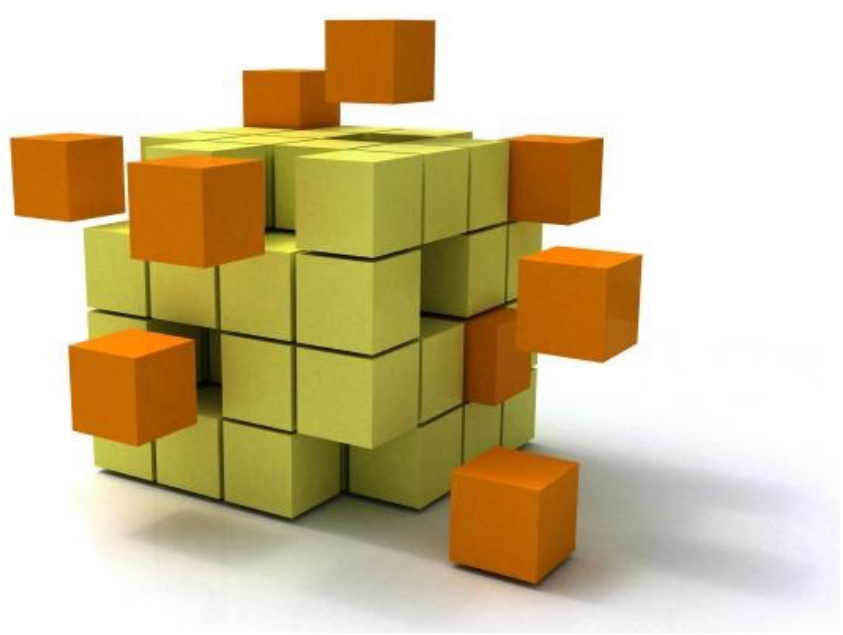


1- Introduction

Structure d'un programme géré par évènements

- Définition d'objets d'interface (instances de classes d'une bibliothèque), de leur arrangement et organisation sur l'écran.
- Définition de traitements d'évènements liés aux objets.
- Définition de la liaison entre les objets d'interface et les traitements.
- Finalement, affichage à l'écran des objets d'interface.





2- Les composants

2- Composants

Objets conteneurs

- objet pouvant contenir d'autres objets
- constitution d'un arbre d'objets graphiques emboités (arbre de scène)
- **JFrame**, **JPanel**, etc

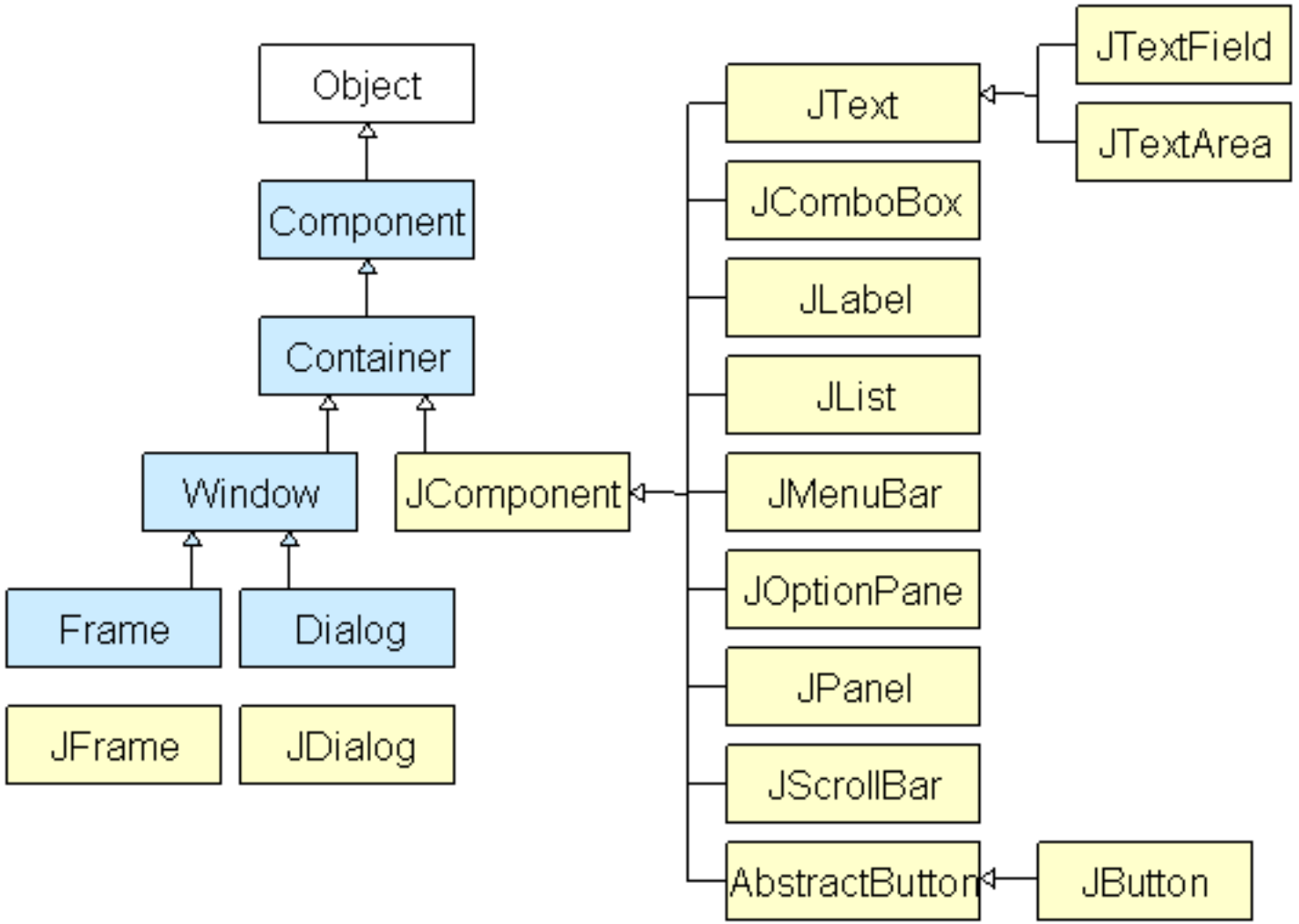
Objets conteneurs racine

- racine d'un arbre d'objets graphiques
- **JFrame**, **JDialog**, **JApplet**
- il existe au moins un objet conteneur racine dans une application
- utilise un objet de positionnement (*layout manager*) pour disposer les objets contenus

Composants (non conteneurs)

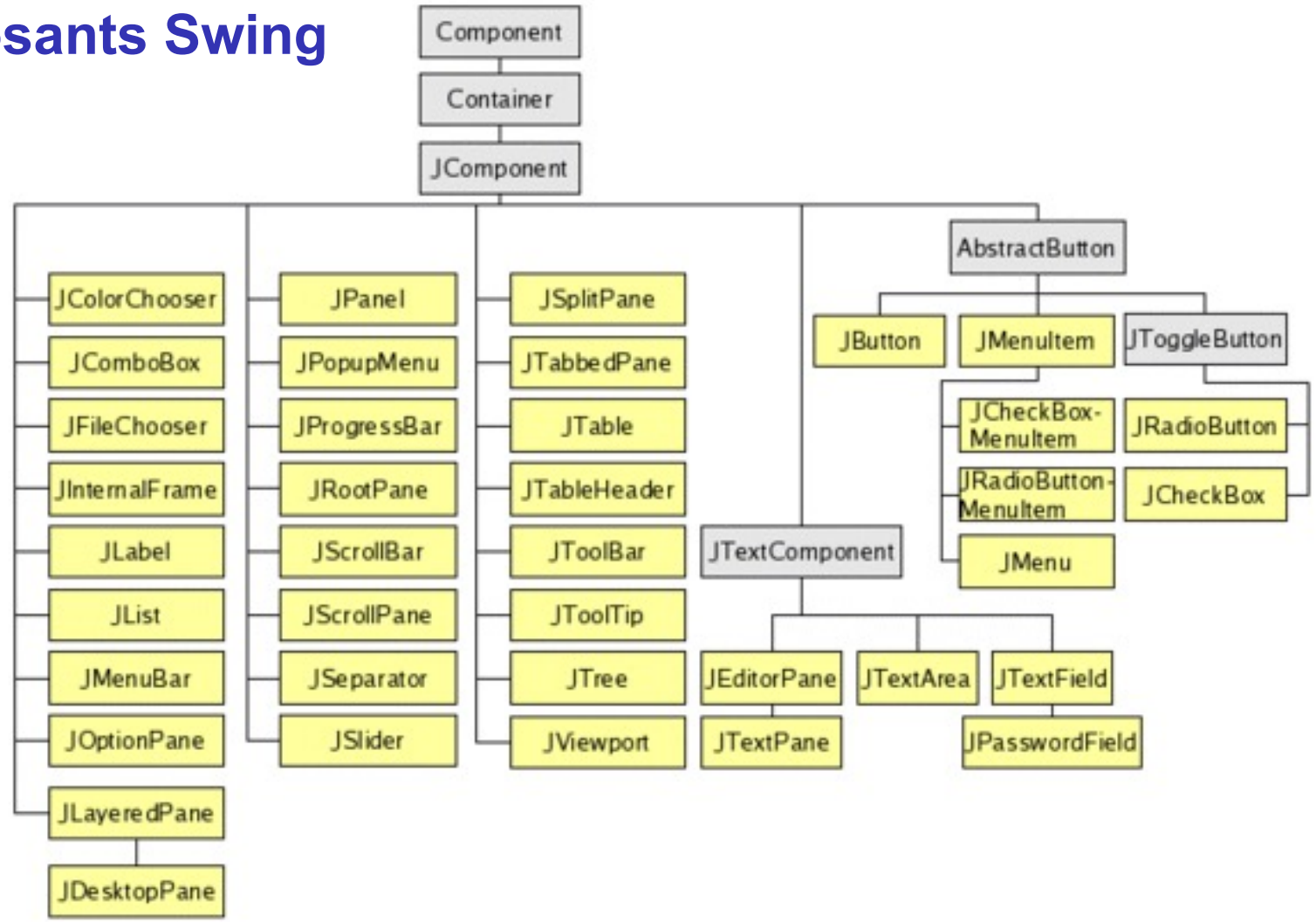
- obligatoirement dans un objet conteneur
- un objet ne peut pas se trouver dans deux conteneurs

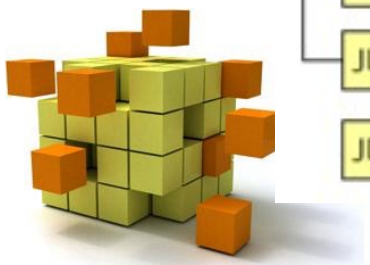
2- Composants



2- Rudiments : interacteurs

Composants Swing





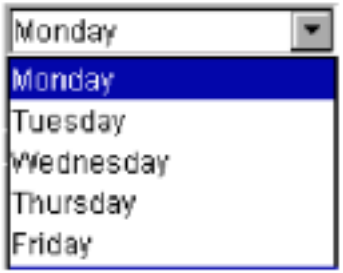
2- Composants : interacteurs

Principaux objets graphiques Swing :

- **JFrame** : fenêtre classique
- **JDialog** : fenêtre de dialogue
- **JPanel** : panneau = élément de structuration
- **JButton** : bouton
- **JRadioButton** : radio bouton
- **JCheckbox** : case à cocher
- **JLabel** : texte
- **JTextField** : champ de saisie
- **JTextArea** : zone de saisie
- **JList** : liste de choix
- **JComboBox** : liste déroulante de choix
- **JScrollBar** : réglette

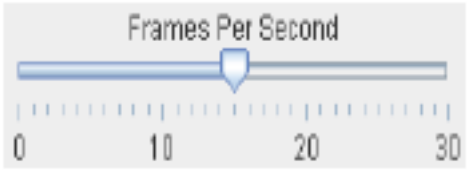
JLabel

JList



JComboBox

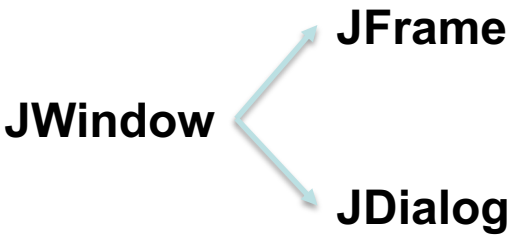
JScrollBar

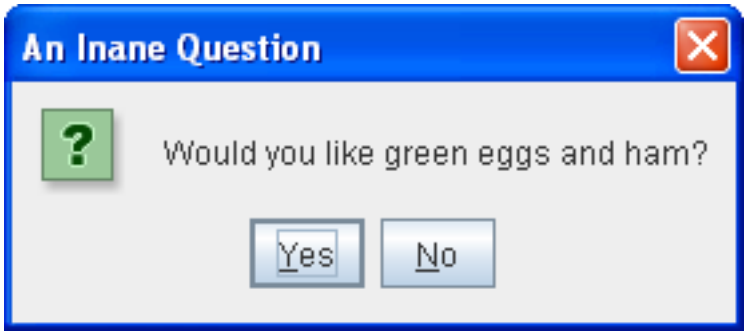


JSlider

Progress bar

2- Composants : fenêtres



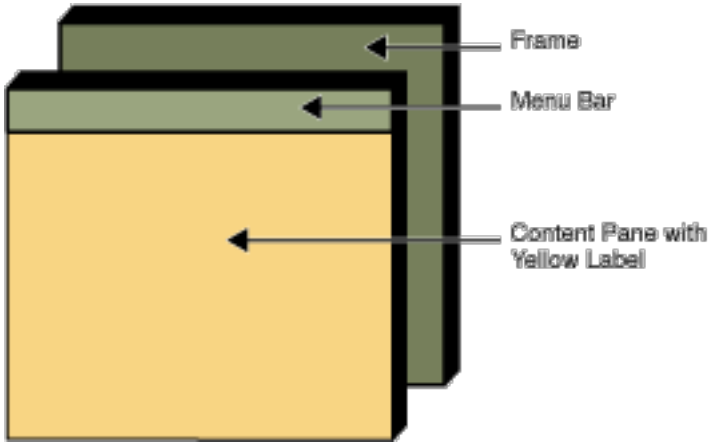


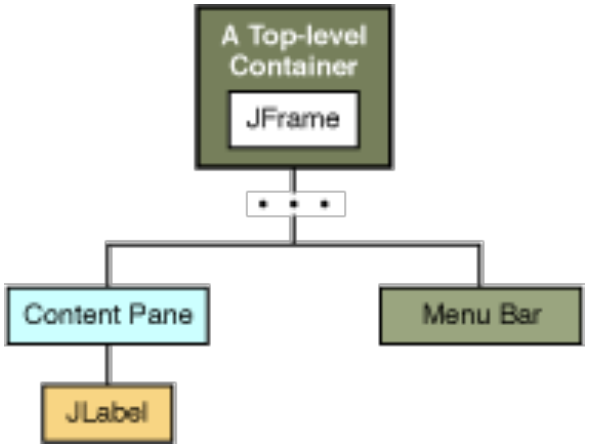
JFrame : fenêtre principale de l'application
JDialog : fenêtre secondaire

- Indépendante de la **JFrame** (toujours au dessus de la **JFrame**)
- Temporaire et modale: bloque l'interaction, impose à l'utilisateur de répondre

2- Composants : fenêtres / JFrame

Panneau de fond (JPanel)

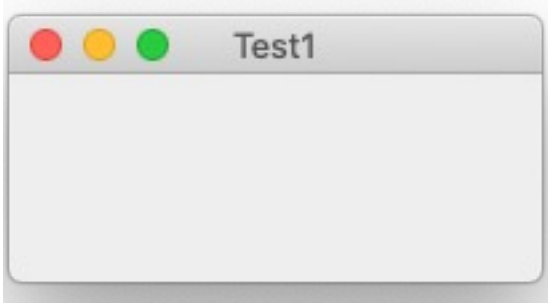




2- Composants : fenêtres / JFrame

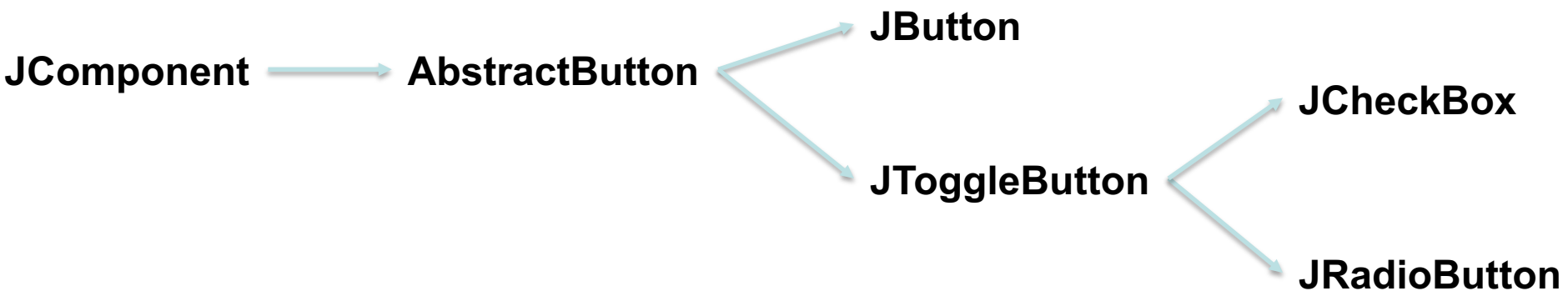
Exemple

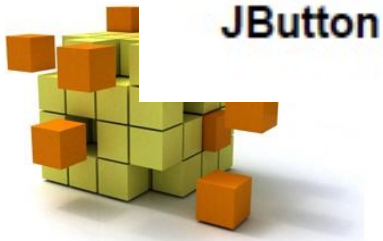
```
package testswing;

import javax.swing.*;

public class Testswing {
    public static void main(String[] args) {
        JFrame f = new JFrame("Test1");
        f.setSize(200,100);
        JPanel p = (JPanel) f.getContentPane();
        f.setVisible(true); // fait apparaître la fenêtre
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



2- Composants : interacteurs

Boutons Swing





JButton

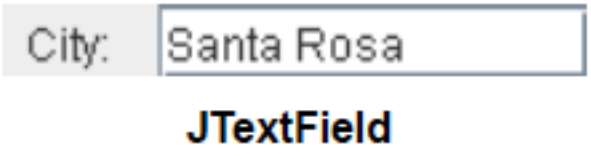
JCheckbox :
choix idépendants

JRadioButton :
choix exclusif : cf. **ButtonGroup**

2- Composants : interacteurs

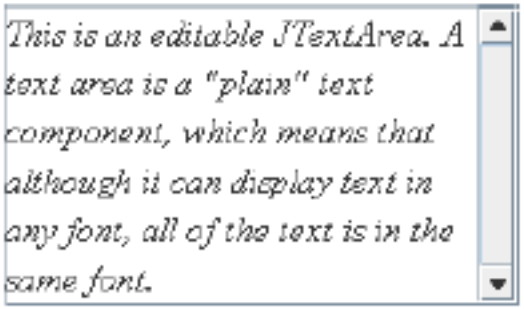
Textes Swing

JComponent → **JTextComponent**
- **JTextField** → **JPassewordField**
- **JTextArea**
- **JEditorPane** → **JTextPane**



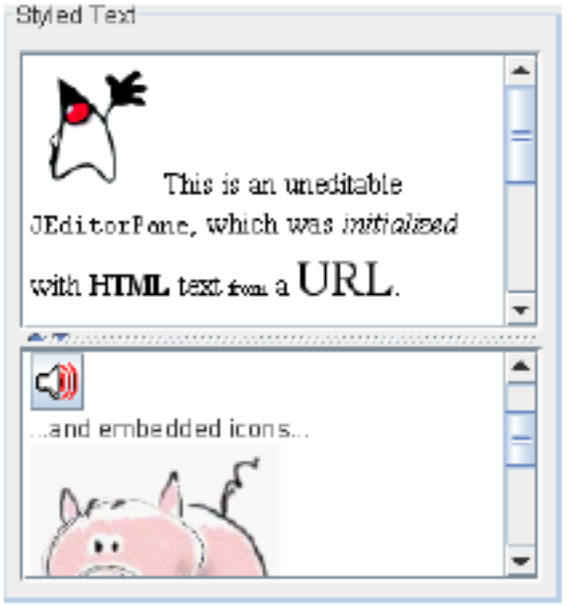
JTextField

JPasswordField



JTextArea :
texte simple multilignes

Ascenseurs :
JScrollPane



JEditorPane : texte avec styles compatible HTML et RTF

2- Composants : interacteurs

JComponent

- **JPanel**
- **JViewPort**
- **JScrollPane**
- **JSplitPane**
- **…**

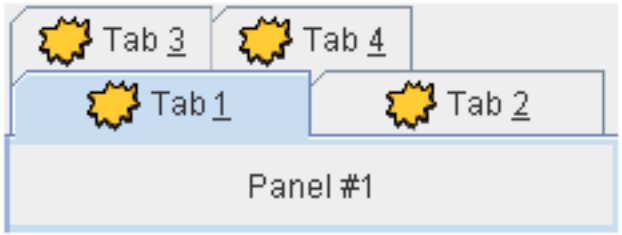
JPanel: conteneur générique

JScrollPane:
avec ascenseurs intégrés

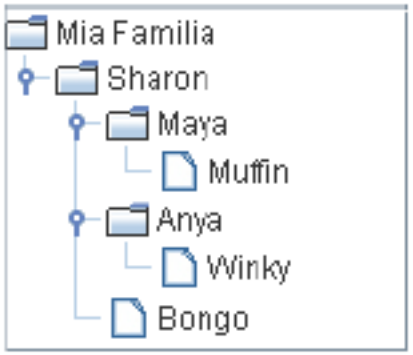
JSplitPane:
avec « diviseur » intégré

2- Composants : conteneurs

JToolBar: barre d'outils
(sous la barre de menus)



JTabbedPane:
onglets

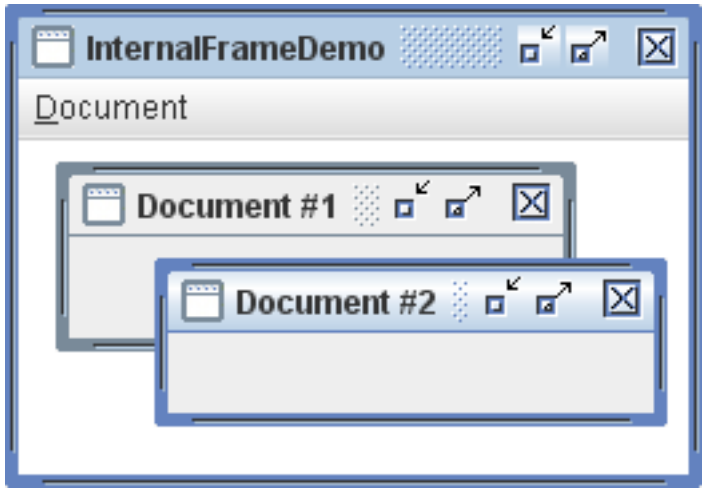


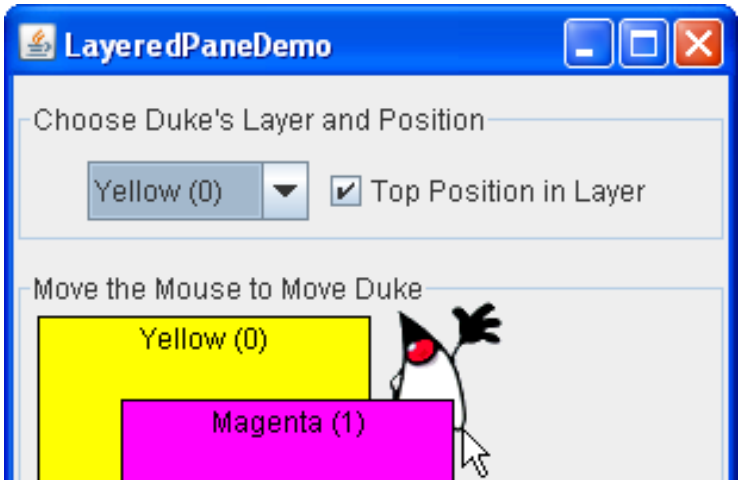
JTree

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

JTable

2- Composants : conteneurs spécifiques





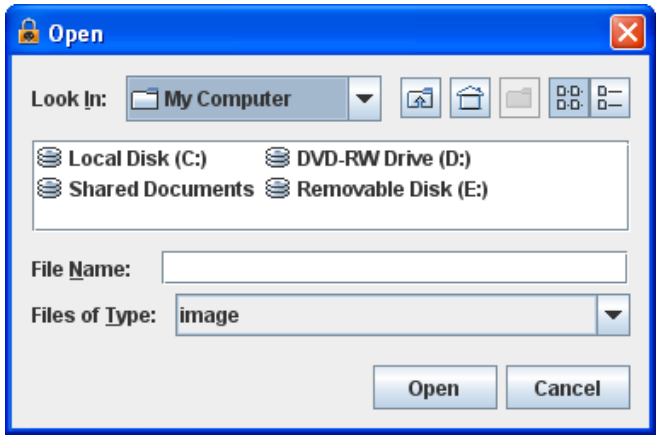
JInternalFrame

- À placer dans un **JDesktopPane**, qui jour un rôle de bureau virtuel.
- **JDesktopPane** hérite de **JLayeredPane**

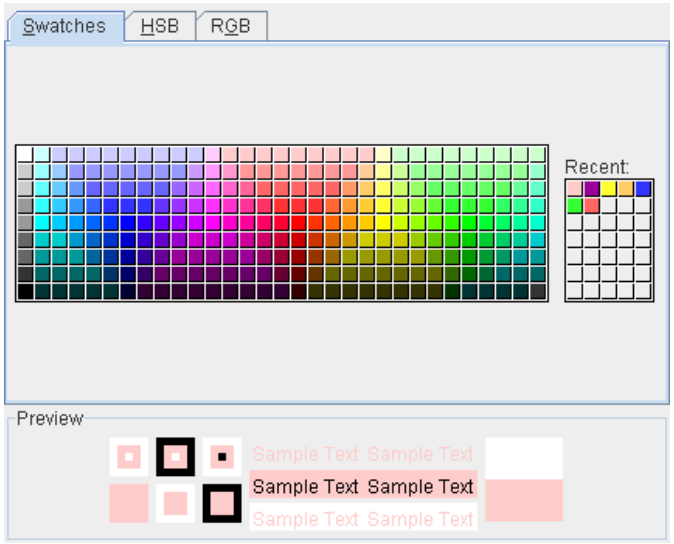
JLayeredPane permet de superposer des composants.

2- Composants : fenêtres / JDialog

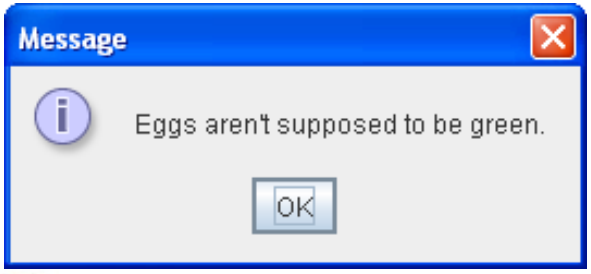
Boîtes de dialogue pré-définies



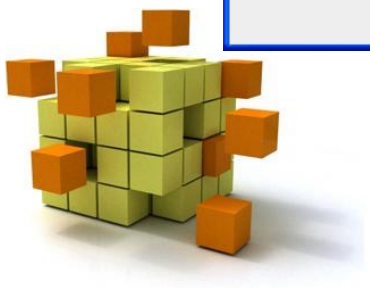
JFileChooser



JColorChooser



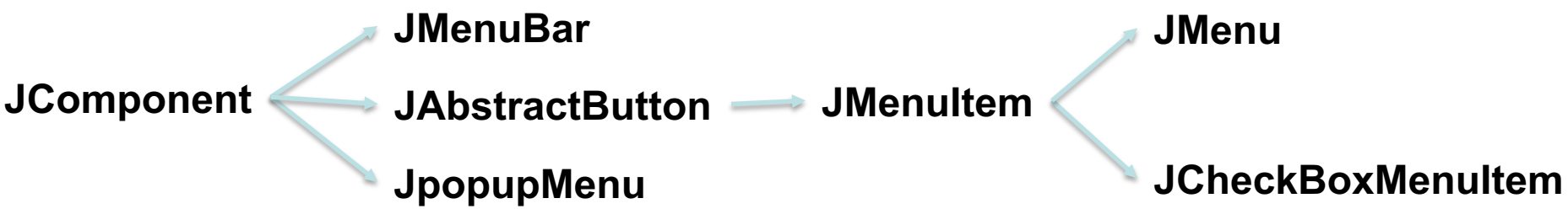
JOptionPane

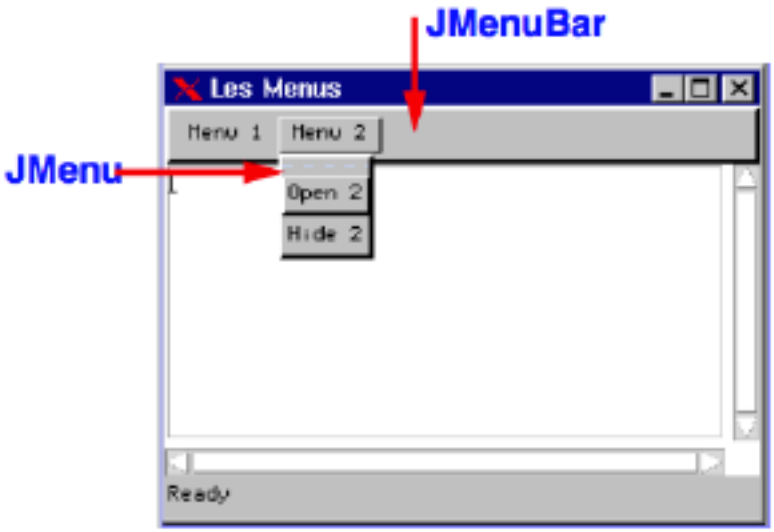


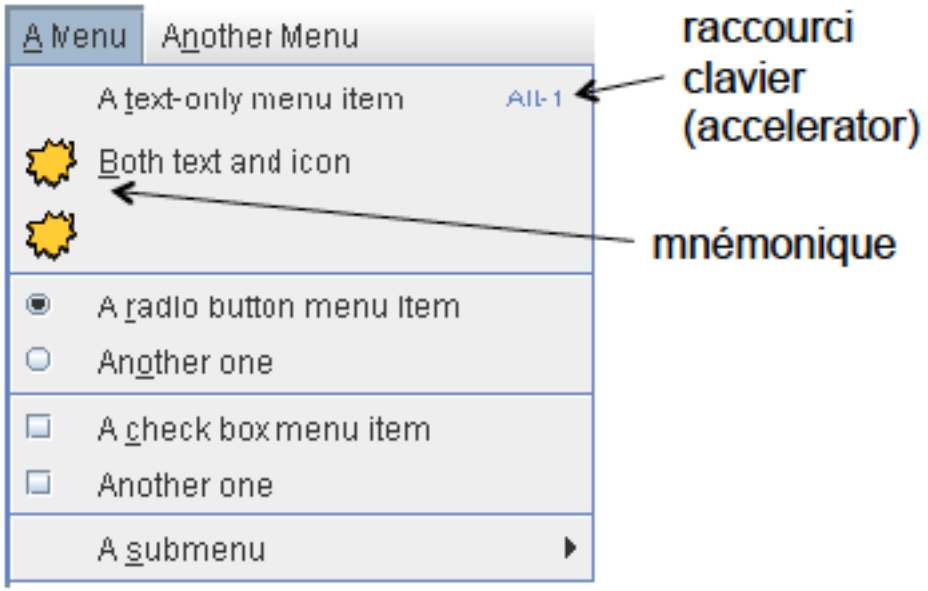
Particularités: peuvent êtres créés

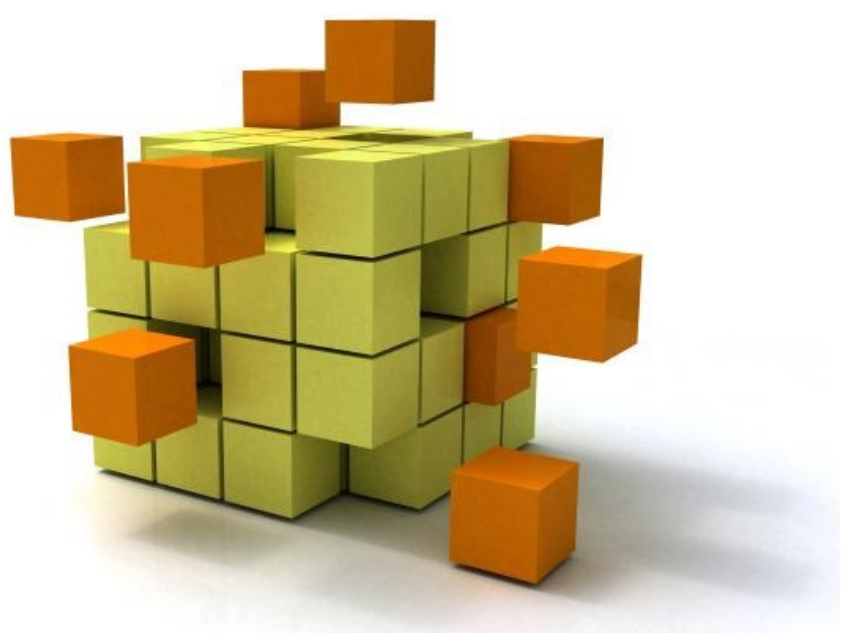
- Comme des composants internes
- Comme des boîtes de dialogue

2- Composants : menus





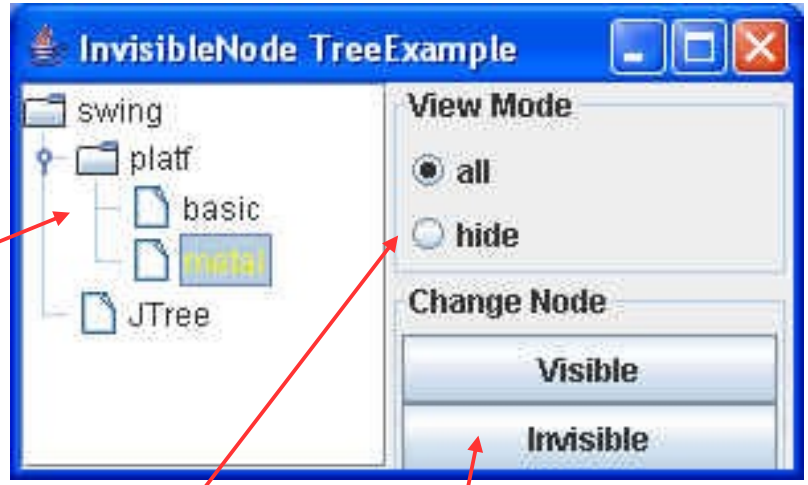


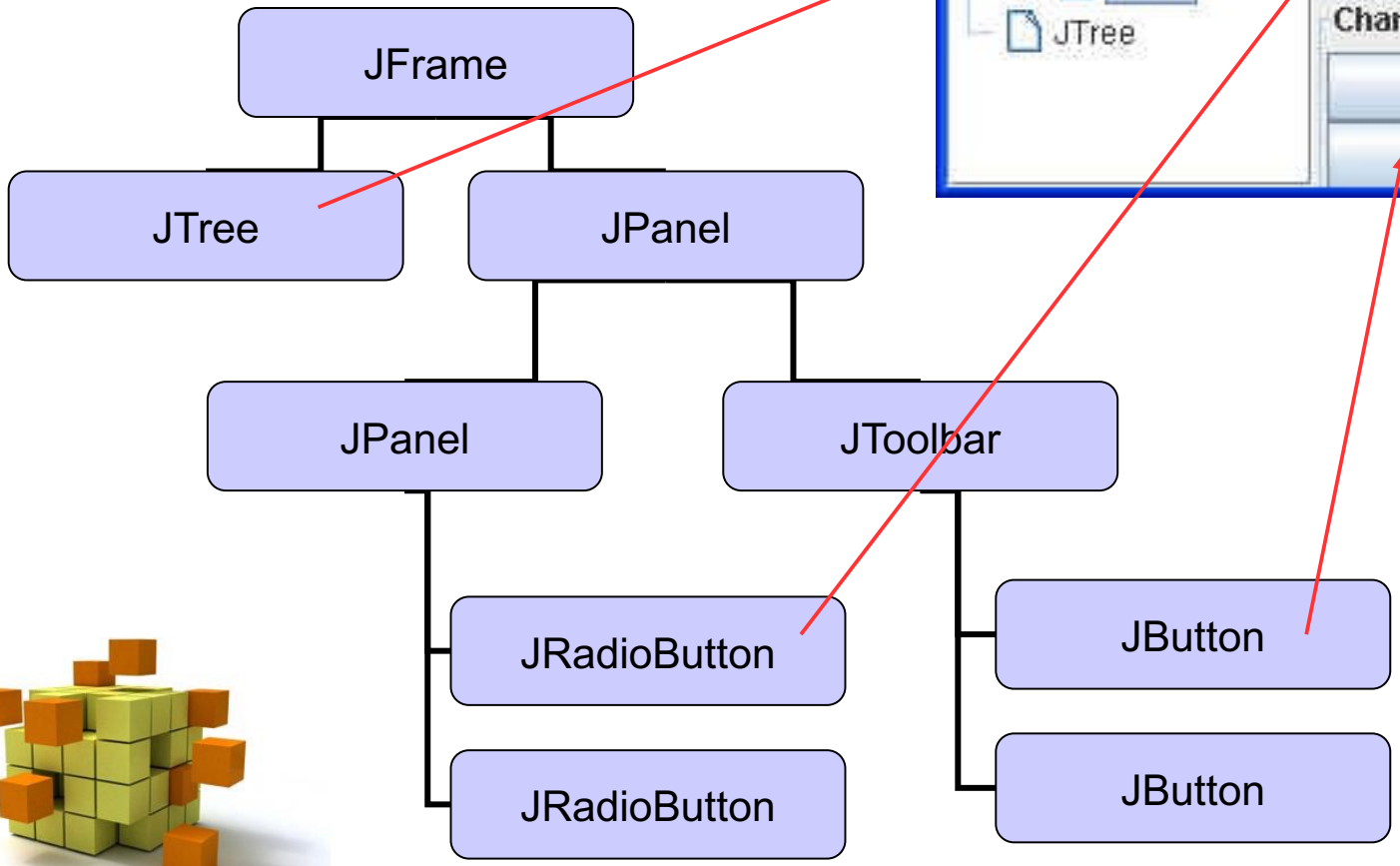


3- Arbre d'instanciation et positionnement

3- Arbre d'instanciation

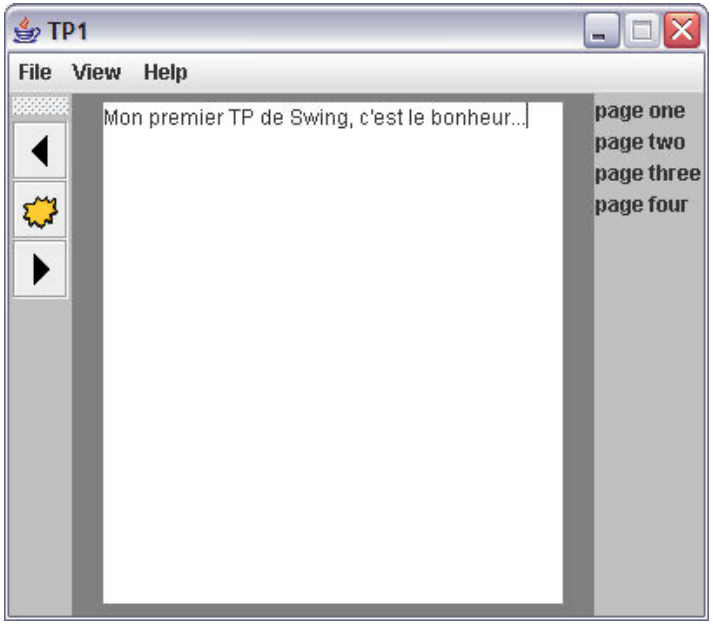
Exemple 1

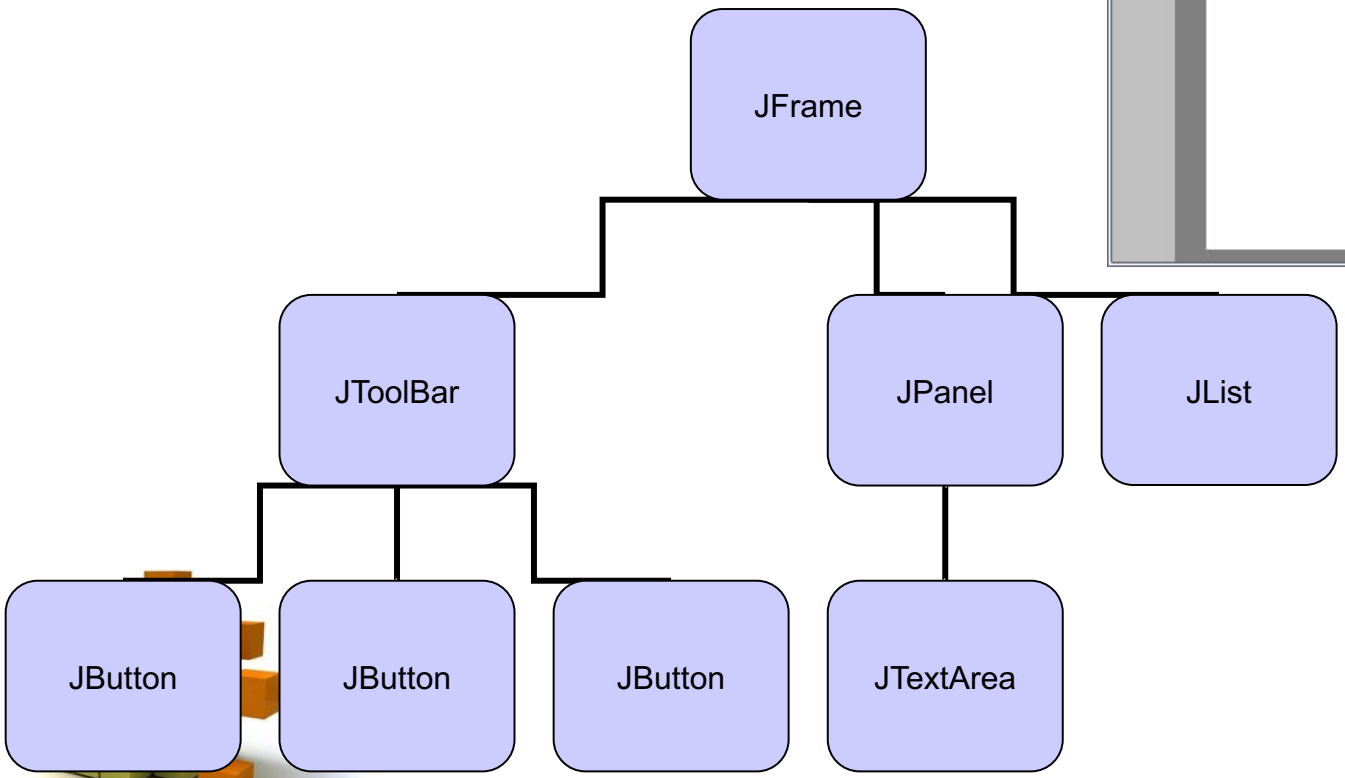




3- Arbre d'instanciation

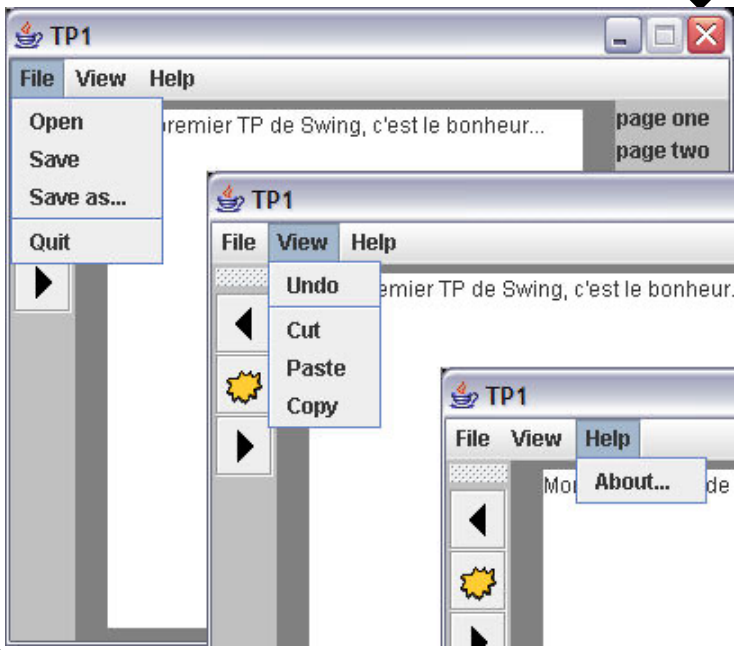
Exemple 2

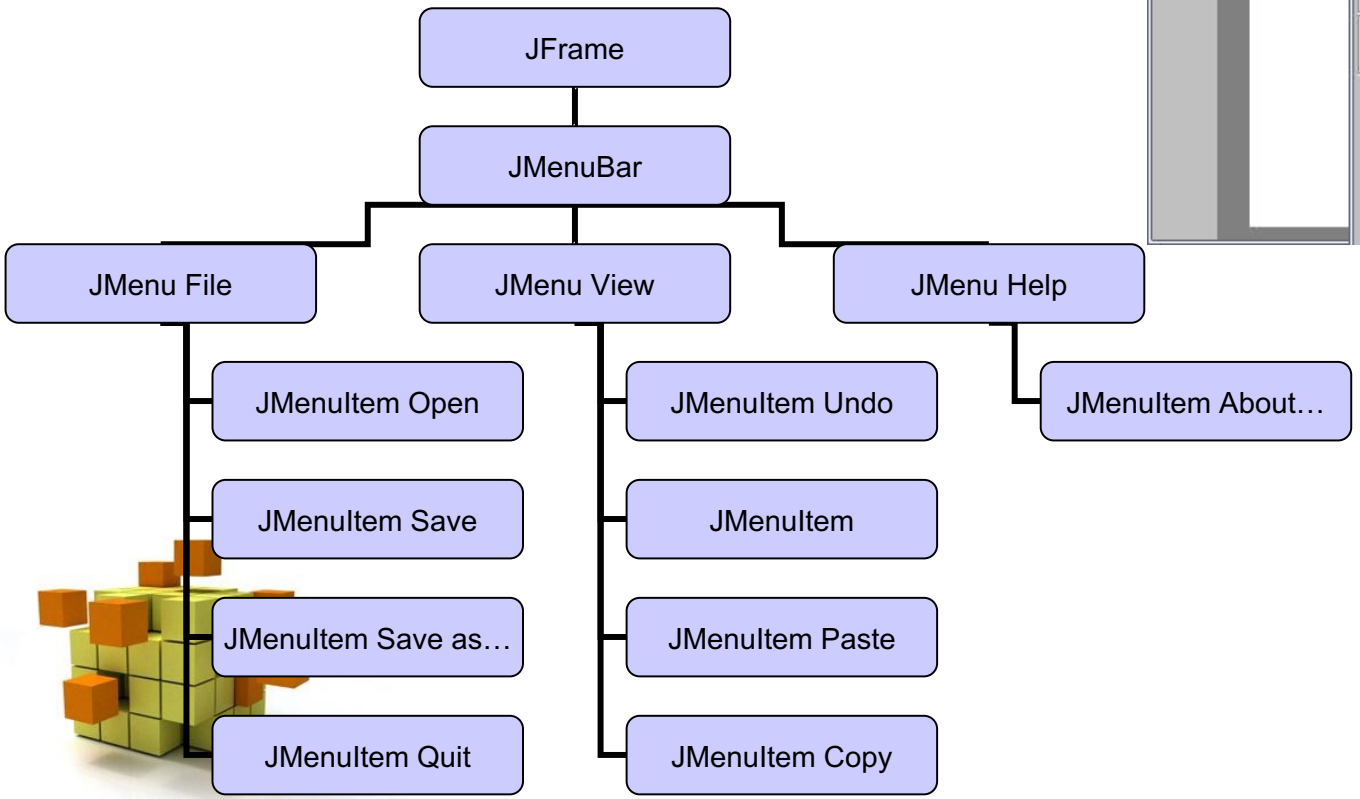




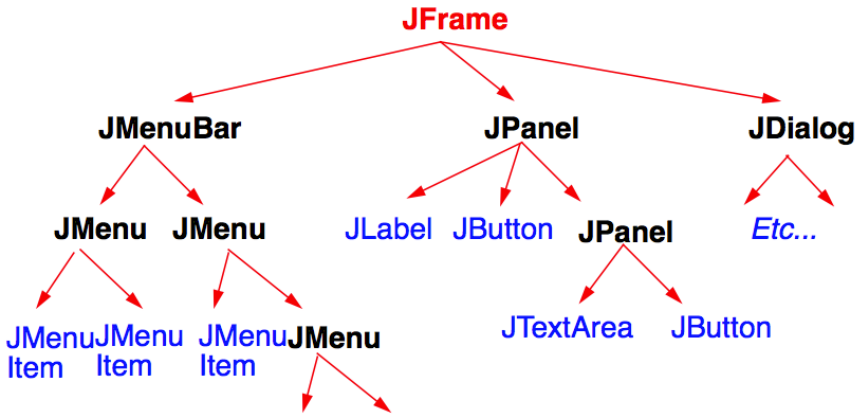
3- Arbre d'instanciation

Exemple 3





3- Arbre d'instanciation



Objet de plus haut niveau de l'arbre
→ **JFrame**

Les conteneurs peuvent être emboités:
→ en particulier les **JPanel**

Les *layout manager* assurent la disposition spatiale:
→ un layout manager par conteneur
→ Défaut pour **JPanel** : *FlowLayout*, pour **JWindow** : *BorderLayout*

Ne pas oublier d'appeler:
→ f.pack() // calcul automatique des boites
→ f.setVisible(true) // fait apparaître la fenêtre

3- Arbre d'instanciation: un exemple

```
import javax.swing.JFrame;
public class Hello {
    public static void main(String[] args) throws Exception {
       Simple f = new Simple();
       f.pack();
       f.setVisible(true);
       f.setDefaultCloseOperation(JFrame. EXIT_ON_CLOSE);
    }
}
```

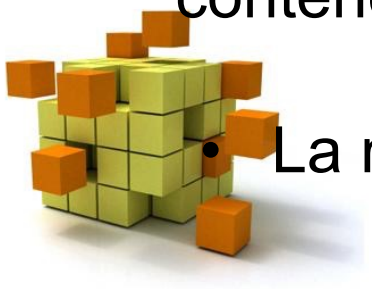
```
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
public class Simple extends JFrame {
        JLabel texte;
        public Simple() {
                super();
                texte = new JLabel("Hello World");
                JPanel p = (JPanel) this.getContentPane();
                p.add(texte);
        }
}
```

3- Arbre d'instanciation: *layout*

Le placement des composants est calculé dans les conteneurs:

- Soit les composants sont placés explicitement (x, y, largeur, hauteur).
- Soit ils sont gérés par un *LayoutManager* qui calcule ces paramètres dynamiquement.

- Les conteneurs définissent la méthode *setLayout(layoutManager)* pour changer le gestionnaire par défaut.
- Le gestionnaire par défaut change d'une classe de conteneur à une autre.



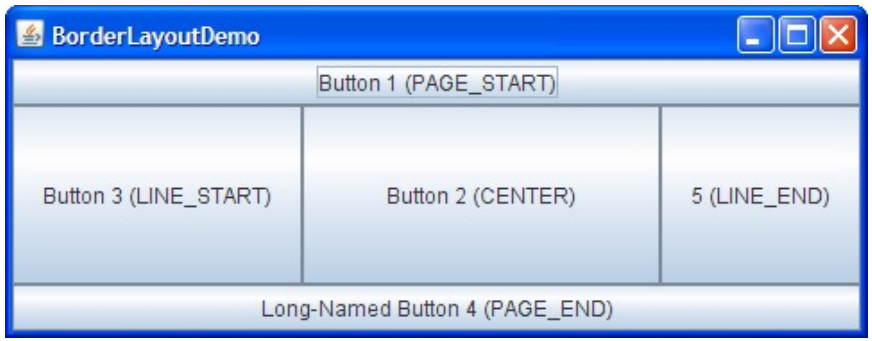
- La méthode *pack()* déclenche le calcul du placement.

3- Arbre d'instanciation: *layout*

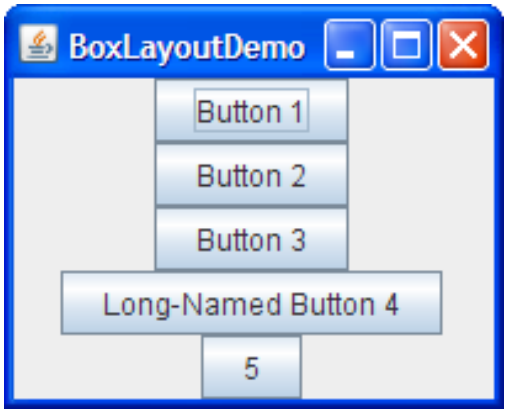
Objets de positionnement (*layout manager*)

classes permettant de disposer des objets dans des objets conteneurs

- **BorderLayout** : 5 positions (North, South, East, West, Center)



- **BoxLayout** : une seule ligne ou une seule colonne



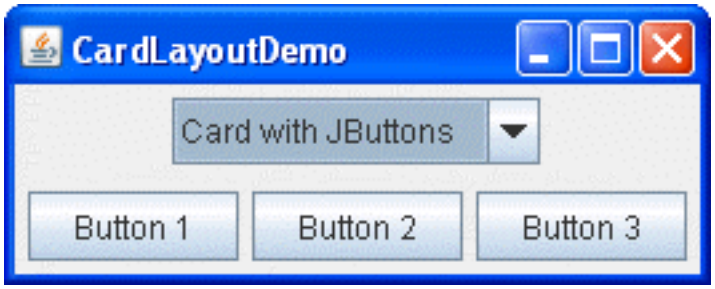
See https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

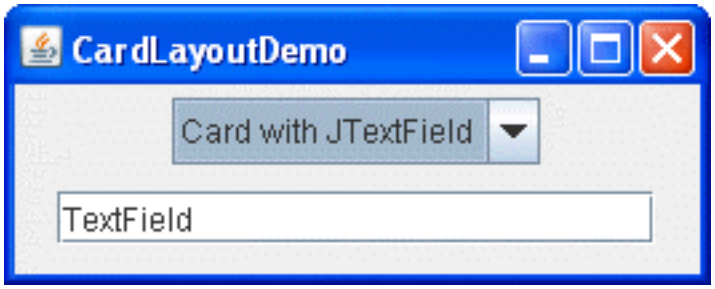
3- Arbre d'instanciation: *layout*

Objets de positionnement (*layout manager*)

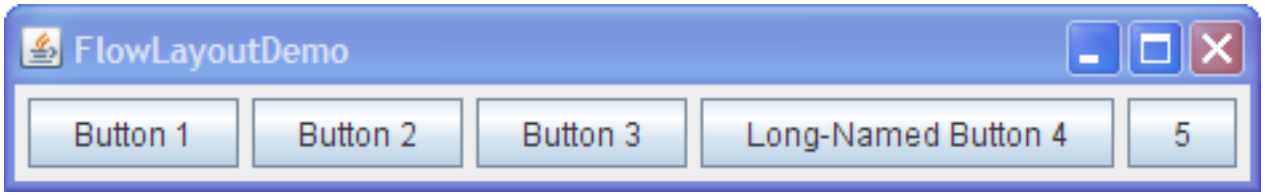
classes permettant de disposer des objets dans des objets conteneurs

- **CardLayout** : pour switcher entre 2+ layouts





- **FlowLayout** : ajoute à la suite et crée une autre ligne si nécessaire

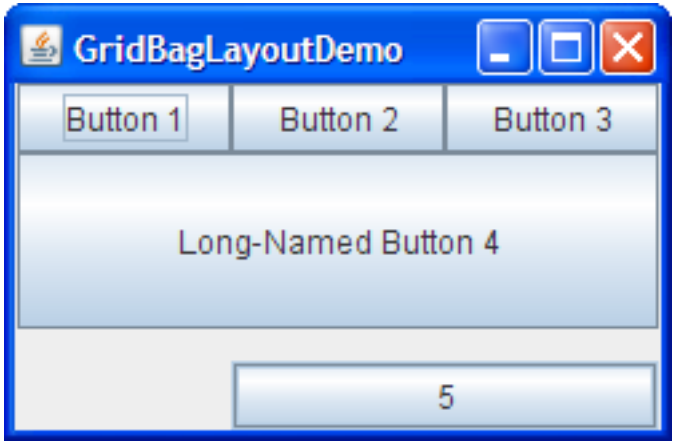


3- Arbre d'instanciation: *layout*

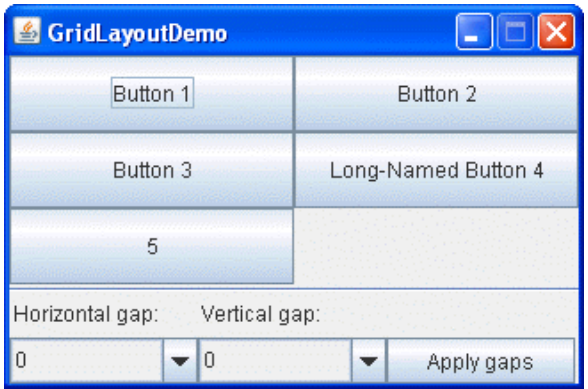
Objets de positionnement (*layout manager*)

classes permettant de disposer des objets dans des objets conteneurs

- **GridBagLayout** : matrice à cases inégales

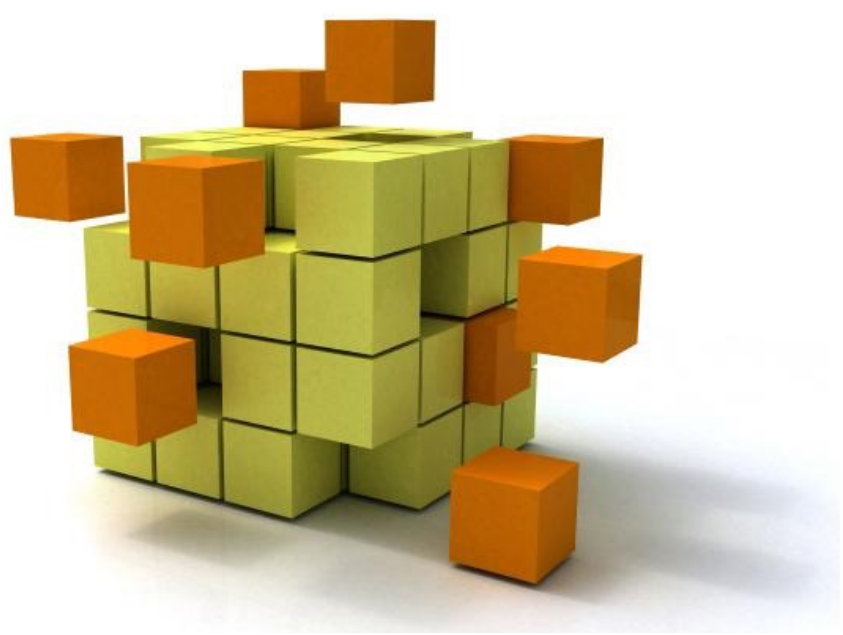


- **GridLayout** : matrice à case égales



3- Arbre d'instanciation: *layout*

```
import …
class Simple extends JFrame {
   JPanel p1, p2;
   public Simple() {
      super();
      p1 = new JPanel();
      p1.setLayout(new FlowLayout()); // Flowlayout
      p1.setBackground(Color.RED);
      JLabel toto = new JLabel("Etiqu1");
      p1.add(toto);

      p2 = new JPanel();
      p2.setLayout(new BorderLayout());
      p2.add(new JButton("bouton"), BorderLayout.CENTER);
      p2.add(new JLabel("Etiqu2"),  BorderLayout.SOUTH);
      p2.setBackground(Color.GREEN);
      p1.add(p2);

      setLayout(new FlowLayout()); // for the JFrame
      add(p1);
      pack(); // don't forget !
   }
}
```

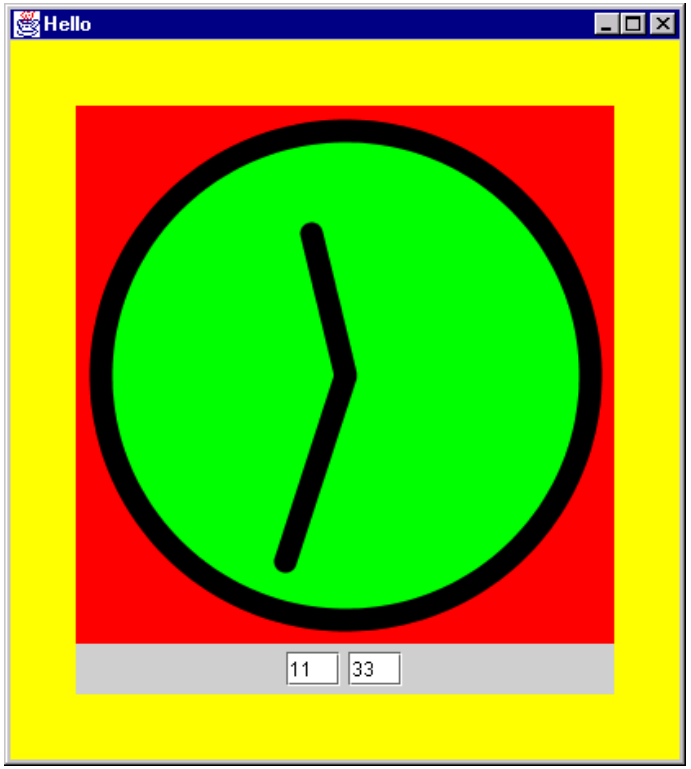


4- Graphiques

4- Graphiques

L'API Java 2D procure des outils pour manipuler

- des photos (*raster graphics*).
- des dessins vectoriels (*vector graphics*).



4- Graphiques

Les dessins vectoriels sont composés de primitives géométriques plus ou moins élémentaires (cercles, courbes de bézier…). L'API Java 2D permet:

- De détecter des collisions sur les formes, le texte et les images.
- De composer des formes élémentaires pour créer des formes plus complexes.
- De gérer la couleur (également la transparence).
- De contrôler la qualité du rendu (*aliasing*).

Java 2D est une technologie puissante. Elle peut être utilisée pour créer des interfaces utilisateur riches, des jeux, des animations ou des applications multimédia.

4- Graphiques: mécanisme de dessin

Le code de dessin personnalisé doit être placé dans la méthode *paintComponent()*. Cette méthode est invoquée lorsqu'il est temps de peindre. Le sous-système de peinture appelle d'abord la méthode *paint()*.

Cette méthode invoque les trois méthodes suivantes :

- *paintComponent()* : peint le composant
- *paintBorder()* : peint les bords du composant
- *paintChildren()* : peint les graphiques enfants du composants

Dans la plupart des cas, nous surchargerons uniquement la méthode *paintComponent()* (le comportement des autres méthodes étant la plupart du temps satisfaisant).

4- Graphiques : l'objet *Graphics*

Le seul paramètre de la méthode *PaintComponent* est un objet *Graphics*. Il procure un certain nombre de méthodes pour dessiner des formes en 2D et obtenir des informations sur l'environnement graphique de l'application.

La classe *Graphics2D* étend la classe *Graphics* pour fournir un contrôle plus sophistiqué de la géométrie, des transformations de coordonnées, de la gestion des couleurs et de la mise en page du texte.

L'objet Graphics est initialisé avant d'être passé à la méthode *paintComponent()*, puis il est retourné aux méthodes *paintBorder()* et *paintChildren()*. Cette réutilisation améliore les performances, mais elle peut entraîner des problèmes si le code de la peinture modifie de façon permanente l'état des graphiques. Par conséquent, nous devons soit restaurer les paramètres d'origine, soit travailler avec une copie de l'objet Graphique. La copie est créée avec la méthode *create()* de *Graphics* ; elle doit être libérée ultérieurement avec la méthode *dispose().*

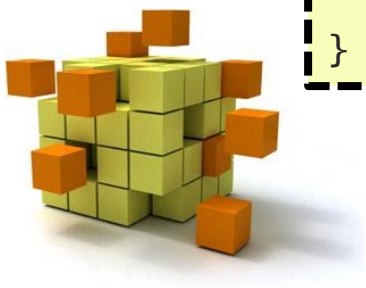
4- Graphiques : l'objet *Graphics*

Concrètement, la copie de l'objet *Graphics* n'a pas besoin d'être créée si nous utilisons les propriétés suivantes : police, couleur et indices de rendu. Pour toutes les autres propriétés (en particulier le clip, les opérations composites et les transformations), nous devons créer une copie de l'objet Graphics et la libérer ensuite.

Exemple simple (Graphics avec texte)

Demo TestGraphics1

```
package TestGraphics1;
import javax.swing.JFrame;

public class TestGraphics1 {
    public static void main(String[] args) {
        BasicEx ex = new BasicEx();
        ex.setVisible(true);
        ex.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



4- Graphiques : l'objet *Graphics*

Exemple (suite)

```
package TestGraphics1;
import javax.swing.JFrame;

class BasicEx extends JFrame {
    public BasicEx() {
        super();
        setTitle("Simple Java 2D example");
        setSize(300, 200);
        add(new Surface());
    }
}
```

```
package TestGraphics1;
import …
class Surface extends JPanel {
    // le constructeur par défaut appel celui de la classe mère
    // Ici rien à faire de plus. Donc pas de constructeur explicit

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawString("Java 2D", 50, 50);
    }
}
```

4- Graphiques : l'objet *Graphics*

Exemple simple (afficher une image)

```
package TestGraphics2;
public class TestGraphics2 {
    public static void main(String[] args) {
        AfficheImage AI = new AfficheImage();
        AI.pack();
        AI.setVisible(true);
        AI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Demo TestGraphics2

4- Graphiques : l'objet *Graphics*

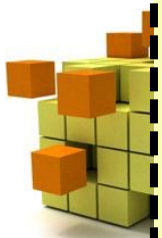
Exemple (suite)

```
package TestGraphics2;

import java.awt.BorderLayout;
import javax.swing.JFrame;

class AfficheImage extends JFrame {
    public AfficheImage() {
        super();
        setTitle("Exemple d'affichage d'une image");
        setLocation(100, 100);
        setLayout(new BorderLayout());
        add(new IHMImages());
    }
}
```

4- Graphiques : l'objet *Graphics*

Exemple (suite)

```
package TestGraphics2;
import …

public class IHMImages extends JPanel {

    String URLimage;
    private Image lune;

    public IHMImages() {
        super();
        setBackground(Color.ORANGE);
        URLimage = "./bin/TestGraphics2/lune.jpg";
        try {
            lune = ImageIO.read(new File(URLimage));
        } catch (IOException exc) {
            exc.printStackTrace();
        }
        Dimension size = new Dimension(lune.getWidth(null), lune.getHeight(null));
        setPreferredSize(size);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(lune, 0, 0, lune.getWidth(null), lune.getHeight(null), this);
    }
}
```



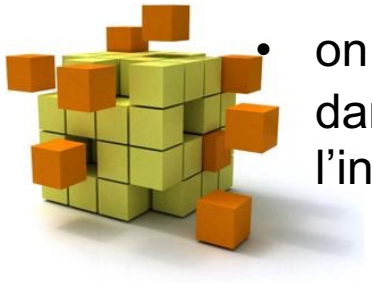
4- Graphiques : rafraichissement

Les méthodes paintComponent() et repaint()

- les objets graphiques de base (**JComponent**) disposent d'une méthode `paintComponent(Graphics g)` appelée automatiquement quand l'élément englobant doit être redessiné (déplacement, masquage, changement de taille de fenêtre).

- si un élément dérivé d'un objet graphique nécessite un dessin particulier la méthode `paintComponent(Graphics g)` doit être redéfinie.

- pour forcer le dessin d'un élément d'interface indépendamment d'une action sur la fenêtre, on appelle `repaint()` qui appelle indirectement `paintComponent(Graphics g)`.



- on appelle `repaint()` quand, par exemple, une valeur est changée dans l'interface et que ce changement nécessite une mise à jour de l'interface.

4- Graphiques : rafraichissement

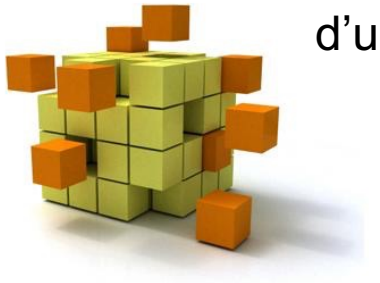
Cascade d'appels lors d'un rafraîchissement

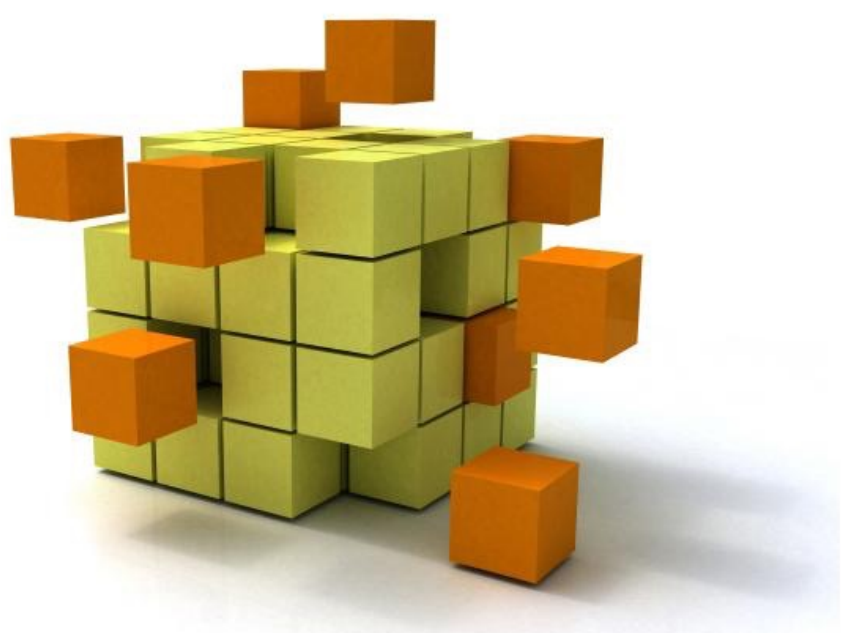
- `repaint()` : méthode par excellence à appeler pour demander un rafraîchissement. Appelle `update(Graphics g, JComponent c)` en lui fournissant un contexte graphique.

- `update(Graphics g, JComponent c)` : appelle `paint(…)`

- `paint(Graphics g)` : appelle successivement
  - `paintComponent()`
  - `paintBorder()`
  - `paintChildren()`

- `paintComponent()` : appelle `ComponentUI.update()` qui efface et redessine le fond (si le composant est opaque, ce qui est le cas d'un **JPanel**).





5- La gestion des évènements en Java

5- Les events

La gestion des évènements

- les objets d'interface présents à l'écran peuvent être rendus réactifs à des actions utilisateur (clic souris, clavier, etc.)
- le système traduit automatiquement l'action utilisateur en un objet de la classe **AWTEvent**, transmis au programme.
- on doit définir des objets écouteurs pour effectuer le traitement lié à un événement
- on doit définir des méthodes de traitement d'évènement dans les objets écouteurs
- on doit relier les objets écouteurs et les objets d'interface.

Demo TestSwing1

5- Les events

Objets écouteurs

- instances de classes spécialisées pour certains types d'évènement (fenêtre, bouton, souris, etc.)
- ces classes doivent implémenter une des **interfaces** écouteurs fournies dans le package **java.awt.event**

Les interfaces écouteurs d'évènements

- `ActionListener`
- `AdjustementListener`
- `ComponentListener`
- `ContainerListener`
- `FocusListener`
- `ItemListener`
- `KeyListener`
- `MouseListener`
- `MouseMotionListener`
- `TextListener`
- `WindowListener`

5- Les events

Exemple d'interface écouteur : évènements « fenêtre »

Action utilisateur : fenêtre activée, désactivée, réduite, fermée, . . .

Méthodes :

```
windowActivated(WindowEvent e)
windowClosed(WindowEvent e)
windowClosing(WindowEvent e)
windowDeactivated(WindowEvent e)
windowDeiconified(WindowEvent e)
windowIconified(WindowEvent e)
windowOpened(WindowEvent e)
```

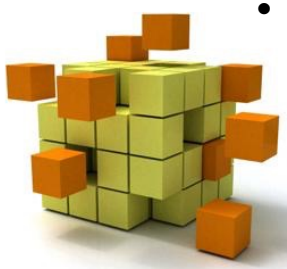
Objets sources : Window (classe ancêtre de **JFrame**)

5- Les events

Lien entre objet écouteur et objet interface

- Les objets d'interface ont des méthodes d'ajout d'objets écouteurs
- Possibilité de plusieurs écouteurs pour un objet interface
- Les méthodes d'ajout d'écouteurs sont spécialisées par type d'évènement
- Méthodes d'ajout d'écouteur :
  - `addActionListener`
  - `addAdjustementListener`
  - `addComponentListener`
  - `addFocusListener`
  - `addItemListener`
  - `addKeyListener`
  - `addMouseListener`
  - `addMouseMotionListener`
  - `addTextListener`
  - `addWindowListener`

Demo TestSwing2



Liste complète des listeners:
https://docs.oracle.com/javase/8/docs/api/java/util/EventListener.html

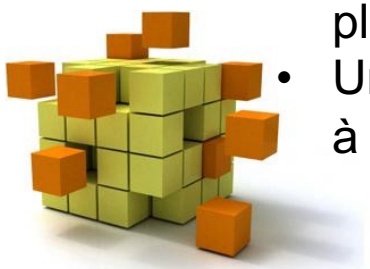
5- Les events

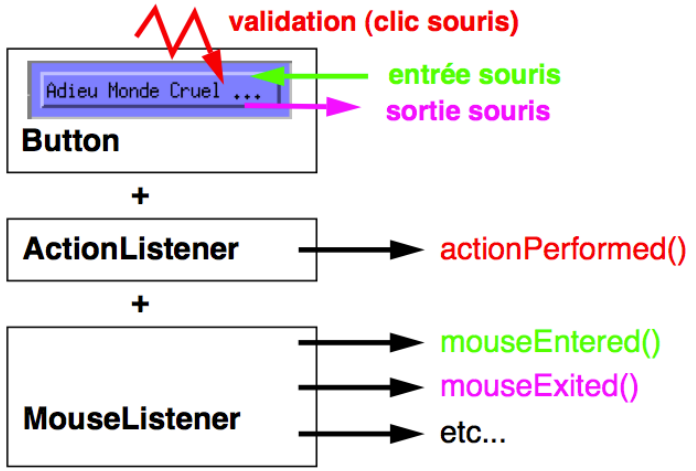
Exemple : MouseEvent

- Evènement : MouseEvent
- Listener : MouseListener
  - `MouseClick(MouseEvent)`
  - `MouseEntered(MouseEvent)`
  - `MouseExited(MouseEvent)`
  - `MousePressed(MouseEvent)`
  - `MouseReleased(MouseEvent)`
- Listener : MouseMotionListener
  - `MouseDragged(MouseEvent)`
  - `MouseMoved(MouseEvent)`

Remarques :

- Un composant graphique peut avoir plusieurs *listeners*
- Un même *listener* peut être associé à plusieurs composants graphiques.

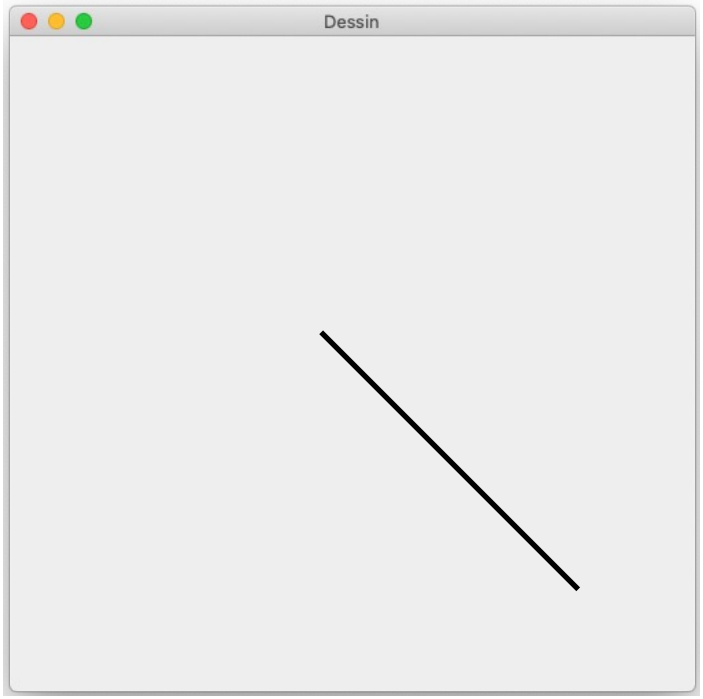




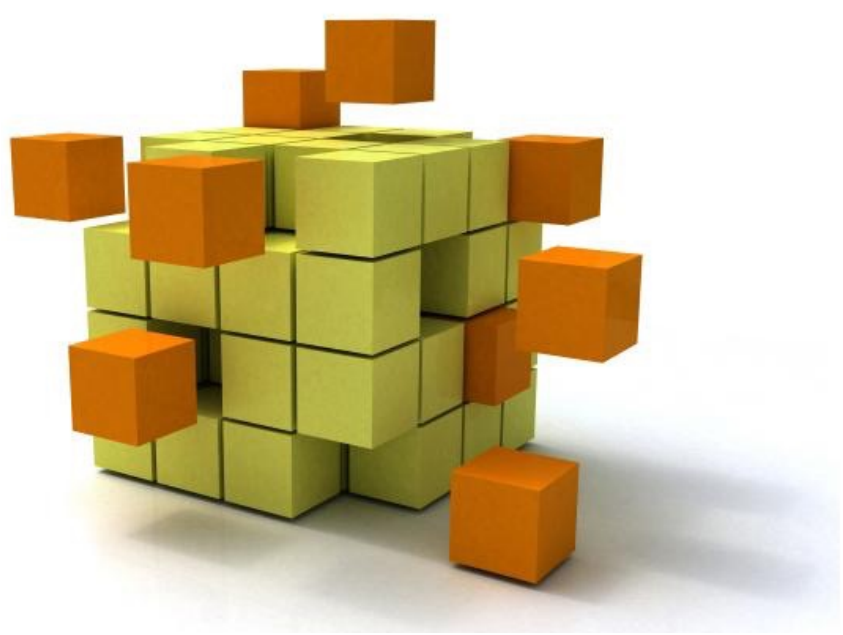
5- Les events

Exemple d'application Swing

- tracé d'un segment (point fixe - position du curseur)
- clic souris -> nouveau point fixe = position curseur
- sortie de fenêtre -> plus de tracé
- entrée dans la fenêtre -> tracé



Demo TestSwing3



6- Classe "BipBip"

Emprunté du cours d'E. Lecolinet

BibBip – Solution 1

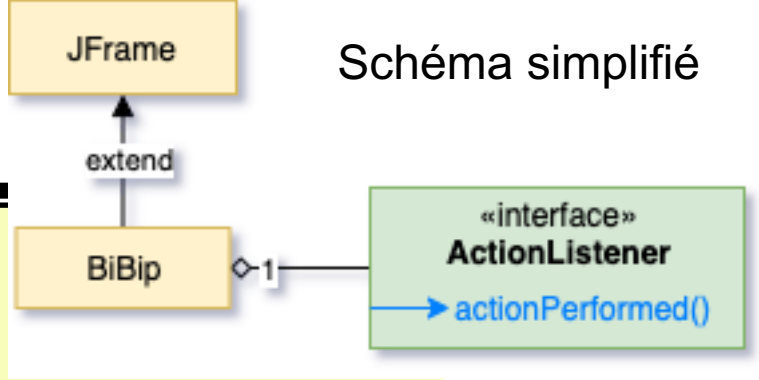


Schéma simplifié

```
public class BipBip extends JFrame {
    JButton button;
    public static void main(String argv[]) {
        BipBip b = new BipBip();
    }

    public BipBip() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();

        button = new JButton("Do It!");
        p.add(button);
        Ecoute elc = new Ecoute(); // Associer un ActionListener
        button.addActionListener(elc);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pack(); setVisible(true);
    }
}
```

Oui, oui, on a le droit de faire cela!

Demo BipBip1

```
class Ecoute implements ActionListener {
    // méthode appelée quand on active le bouton
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Done!");
    }
}
```

BibBip – Solution 1

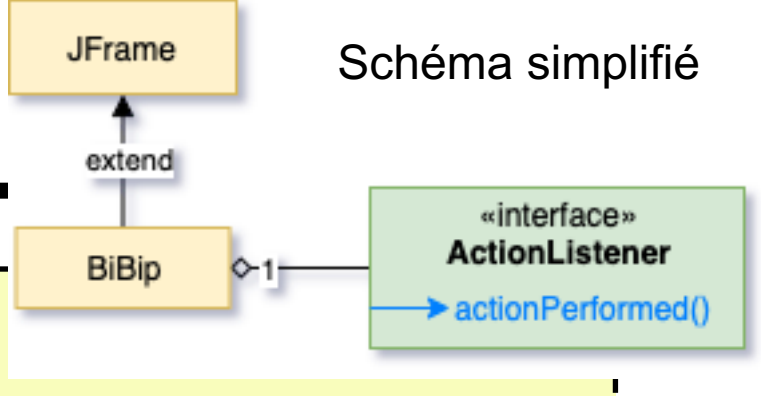


Schéma simplifié

```
public class BipBip2 extends JFrame {
    JButton button;
    JLabel label;

    public static void main(String argv[]) {
        BipBip2 b = new BipBip2();
    }

    public BipBip2() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();

        button = new JButton("Do It!");
        p.add(button);
        label = new JLabel("Texte");
        p.add(label);

        Ecoute elc = new Ecoute(label);// Associe un ActionListener
        button.addActionListener(elc);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack(); setVisible(true);
    }
}
```

BibBip – Solution 1

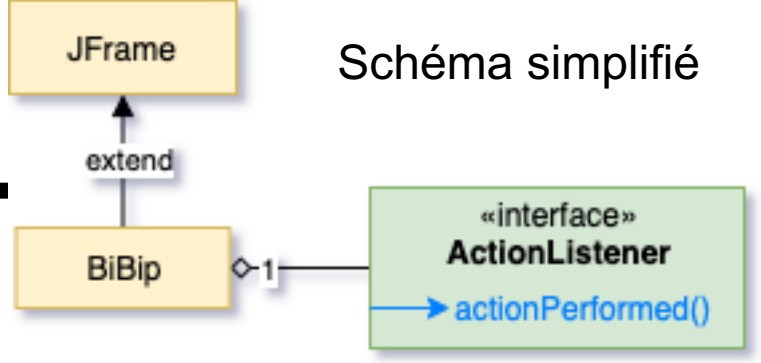


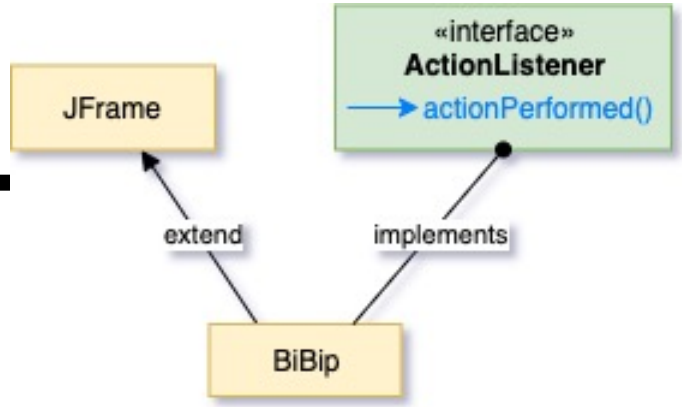
Schéma simplifié

Demo BipBip2

```
class Ecoute implements ActionListener {
    JLabel label;
    public Ecoute(JLabel label) {
        this.label = label;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText("Done!");
    }
}
```

Solution OK mais un peu lourde!

BibBip – Solution 2

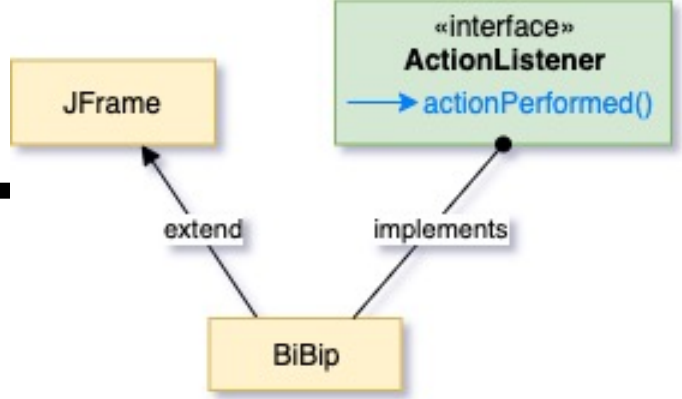



```
public class BipBip3 extends JFrame implements ActionListener {
    JLabel label;
    …
    public BipBip3() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();
        button = new JButton("Do It!");
        p.add(button);
        label = new JLabel("Texte");
        p.add(label);
        button.addActionListener(this);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack(); setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText("Done!");
    }
}
```

Demo BipBip3

Accès direct à label !

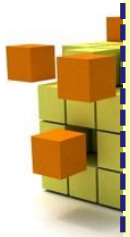
BibBip – Solution 2



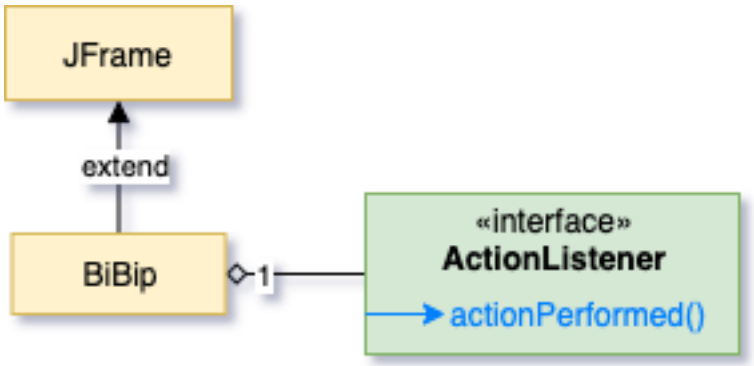

```
public class BipBip4 extends JFrame implements ActionListener {
    JButton doit, close;
    JLabel label;
    …
    public BipBip4() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();
        p.add(doit = new JButton("Do It!"));
        p.add(close = new JButton("Close"));
        p.add(label = new JLabel("Texte"));
        doit.addActionListener(this);
        close.addActionListener(this);
        …
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == doit)
            label.setText("Done!");
        else if (e.getSource() == close)
            System.exit(0);
    }
}
```

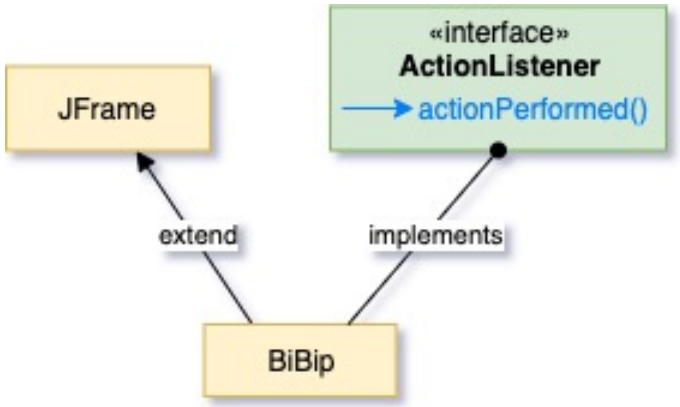
Demo BipBip4

OK mais non adapté s'il y a beaucoup de commandes



BibBip – Synthèse





Version 1

- plus souple :
  autant de listeners que l'on veut
- mais peu concis :
  on multiplie les objets et les lignes de code

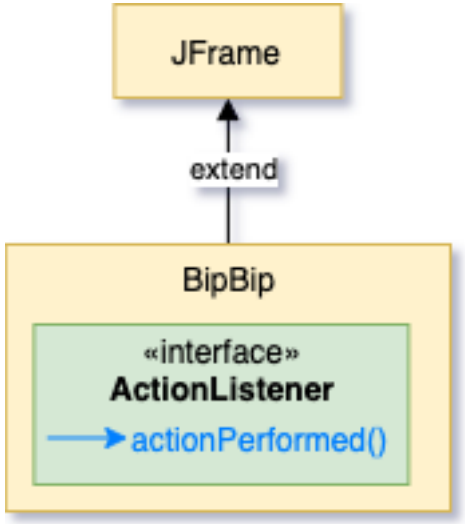
Version 2

- plus simple mais limitée :
  on ne peut avoir qu'une seule méthode actionPerformed()
- peu adapté si beaucoup de commandes

→ Classes imbriquées

BibBip – Solution 3

→ Classes imbriquées (classe interne ou *inner class*)



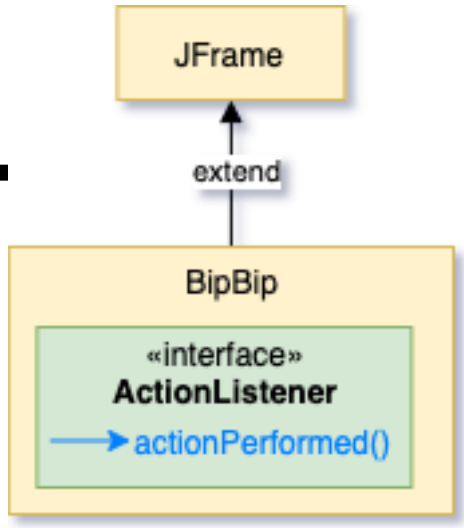
Classes définies à l'intérieur d'une autre classe

- ont accès aux variables d'instance des classes qui les contiennent
  (c'est une forme de capture de variables)

Combinent les avantages des 2 solutions précédentes

- souplesse sans la lourdeur !

BibBip – Solution3

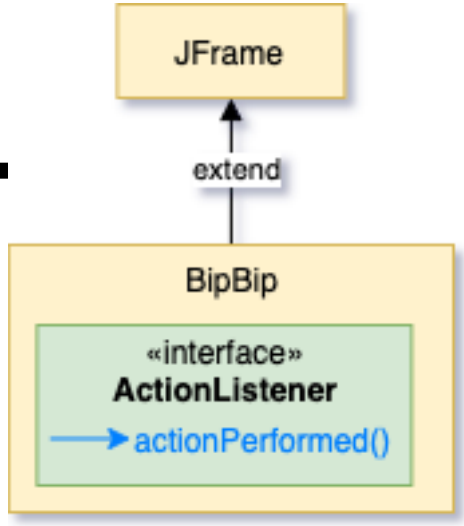



```
public class BipBip5 extends JFrame {
    …
    public BipBip5() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();
        p.add(doit = new JButton("Do It!"));
        p.add(close = new JButton("Close"));
        p.add(label = new JLabel("Texte"));
        doit.addActionListener(new DoItListener());
        close.addActionListener(new CloseListener());
        …
    }

    class DoItListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            label.setText("Done!");
        }
    }

    class CloseListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    }
}
```

ActionPerformed() à accès à label car DoItListener est une classe imbriquée de BipBip5

Demo BipBip5

BibBip – Solution3




```
public class BipBip6 extends JFrame {
    …
    public BipBip6() {
        setLayout(new FlowLayout());
        JPanel p = (JPanel) getContentPane();
        p.add(doit = new JButton("Do It!"));
        p.add(close = new JButton("Close"));
        p.add(label = new JLabel("Texte"));
        doit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("Done!");
            }
        });
        close.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        …
    }
}
```

Sous-classe anonyme de *ActionListener*

Demo BipBip6