



# Programmation événementielle

# Interface graphique

- Avant : ligne de commande
  - Appel un programme
    - Si le programme à besoin d'une entrée : utilisateur
    - Donne un résultat

# Interface graphique

- Permettre interactivité plus simple
  - Entrées
  - Sorties
  - Lien
- Interface graphique
  - Interaction -> Action
  - Click -> Action
  - Non linéaire (en terme de déroulement du prog)

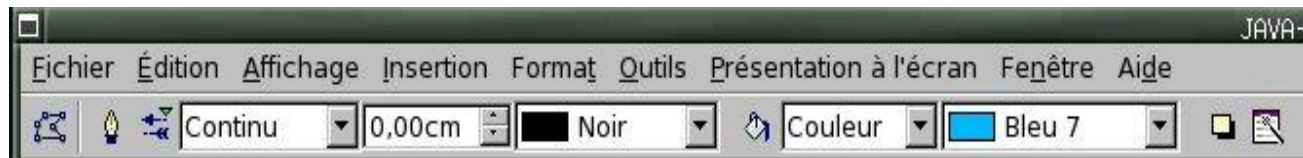
# Les entrées

- Le gestionnaire d'interface graphique (Windows MacOs Xwindow Gtk qt)
  - Récupère information utilisateurs
    - Frappe touche
    - Déplacement souris
    - Autres entrées



# Les sorties

- Affichage
  - Données et état du système
    - Fenêtres, boutons, menu
    - Widgets



# Le lien

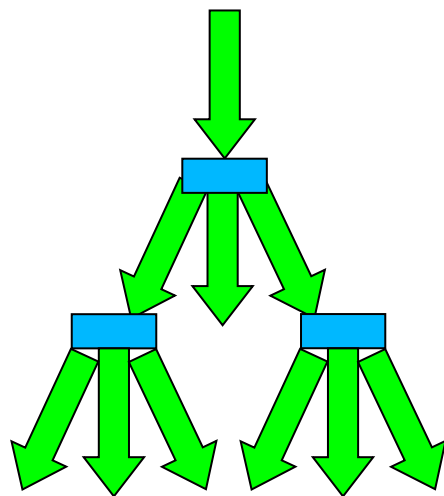
- Entre action utilisateur et cœur du programme
  - Ouvrir un fichier, le mettre en mémoire
  - Faire un calcul
  - Afficher une image
  - ...

# Programmation évènementielle

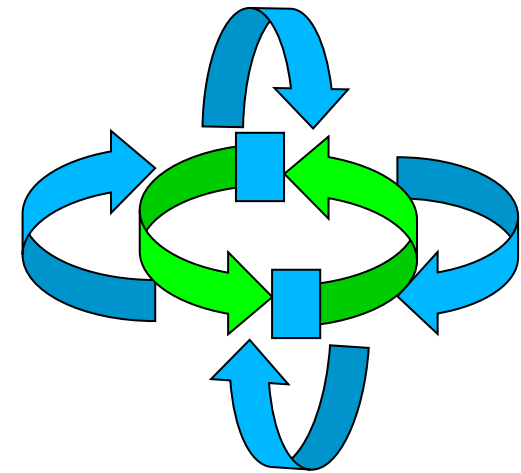
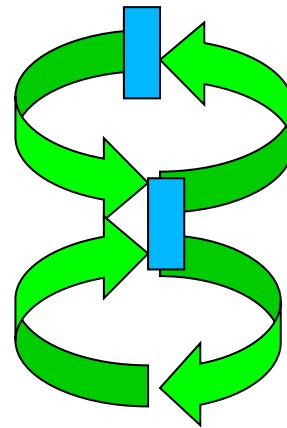
- Concept différent de la programmation fonctionnelle ou impérative
- Interaction non-linéaire
  - Plusieurs évènements peuvent être produit dans un ordre différent

exécution

choix



Programmation  
fonctionnelle



Programmation  
évènementielle

# Programmation événementielle

- Les composants interagissent entre eux et avec l'environnement
- Ils communiquent en réponse à des événements
- Les événements sont captés par le système et sont transmis aux *écouteurs* (listener)
- C'est **le programme** qui mettez en place les écouteurs (→ **implements**)

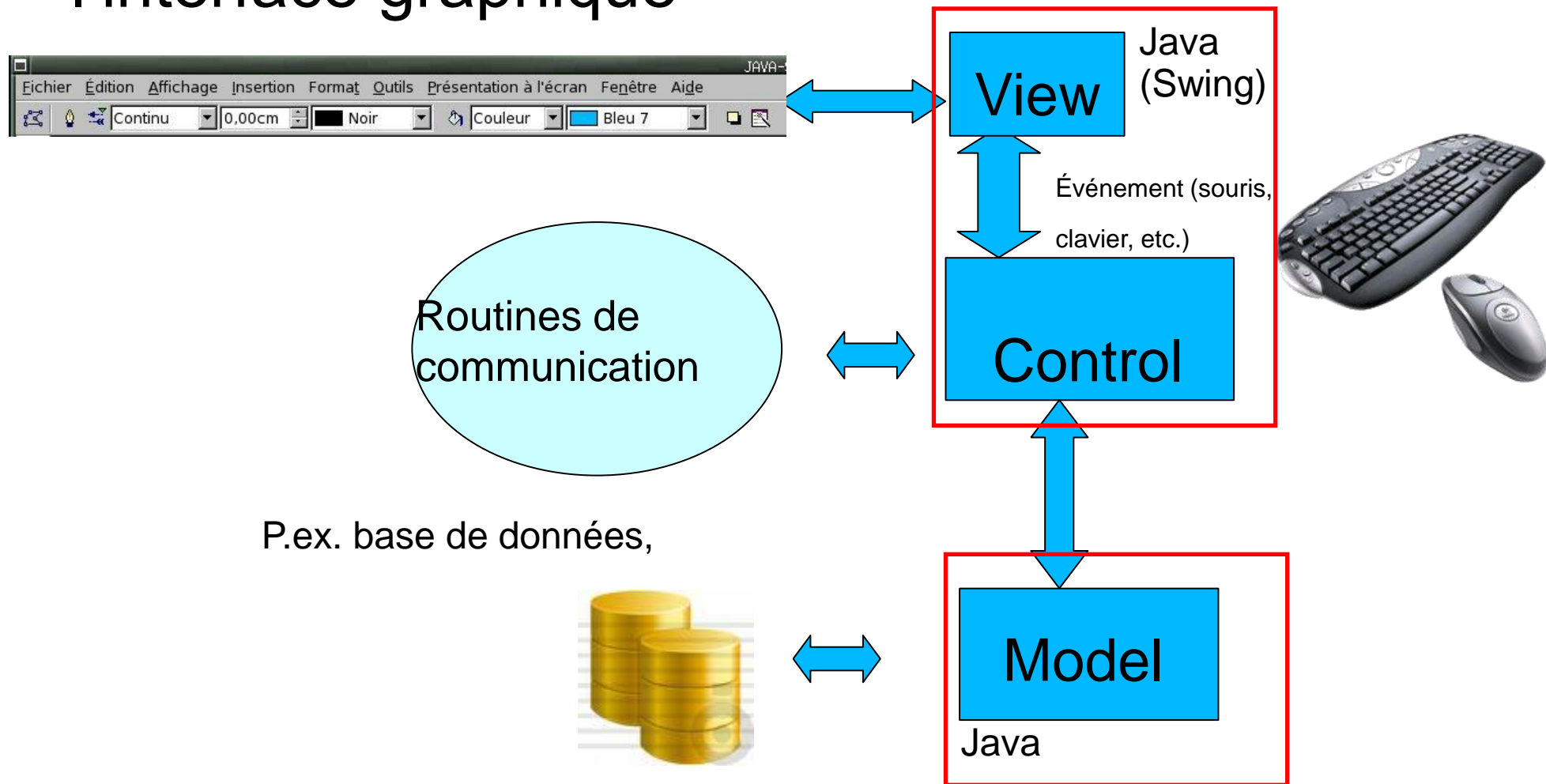
# Programmation événementielle

Pour réaliser des interfaces, il est **impératif** de programmer de manière événementielle et donc de prévoir les événements.

Dans ce but, vous devez connaître les capacités de votre interface et les événements qu'elle peut générer.

# Conception évènementielle

- Séparer la gestion du modèle de la gestion de l'interface graphique



# Model View Controller

- Motif de conception qui sépare
  - le modèle de données
  - l'interface utilisateur
  - la logique de contrôle
  
- Mis au point en 1979 par Trygve Reenskang, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC.

# Vue



- Interface avec laquelle l'utilisateur interagit
- Les résultats renvoyés par le modèle sont dénués de toute présentation mais sont présentés par les vues.
- N'effectue aucun traitement, affiche les résultats des traitements effectués par le modèle
- Permet à l'interaction
- Plusieurs vues peuvent afficher les informations d'un même modèle

# Modèle

- Représente le comportement de l'application
  - traitements des données
  - interactions avec la base de données
  - ...
- Décrit les données manipulées par l'application
- Définit les méthodes d'accès.

# Contrôleur

- Prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle
  - N'effectue aucun traitement
  - Ne modifie aucune donnée
  - Analyse la requête du client et appelle le modèle adéquat
  - Renvoie la vue correspondant à la demande

# MVC

- Lorsqu'un client envoie une requête à l'application:
  - celle-ci est analysée par le contrôleur
  - ce contrôleur demande au modèle approprié d'effectuer les traitements
  - puis renvoie la vue adaptée

# Concevoir une interface

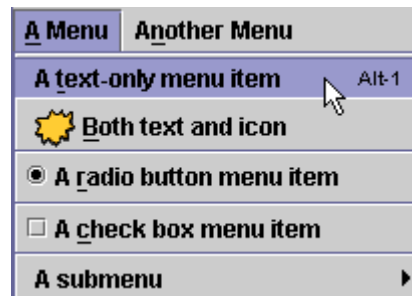
- Connaître le modèle et son fonctionnement
- Définir le contrôle sur le modèle
  - ouvrir, fermer, afficher, etc.
- Faire un croquis de l'interface pour définir les widgets à utiliser et leurs caractéristiques
  - menu, bouton, label, etc.
  - nom, position, etc.

# En Java

- AWT (nearly deprecated)
- SWING



Buttons



Menu



Spinner



# Swing

- Contenu dans le package javax.swing.

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

- Indépendant du système d'exploitation
  - Portabilité des programmes

# Programmer avec SWING

- Comme un autre programme Java
  - En utilisant les classes de SWING
- Programmer une interface graphique
  - Implémenter un ensemble de classes

# Programmer une vue

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

public class HelloWorldSwing extends JFrame {

    HelloWorldSwing(String title) {
        super(title);
        Container pane = getContentPane();
        pane.add(new JLabel("Hello !"));
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        HelloWorldSwing mainWindow =
            new HelloWorldSwing("title");
    }
}
```

# Programmer une vue

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

public class HelloWorldSwing extends JFrame {

    HelloWorldSwing(String title) {
        super(title);
        Container pane = getContentPane();
        pane.add(new JLabel("Hello !"));
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        HelloWorldSwing mainWindow =
            new HelloWorldSwing("title");
    }
}
```

# Programmer une vue

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

Importation

```
public class HelloWorldSwing extends JFrame {
```

```
    HelloWorldSwing(String title) {  
        super(title);  
        Container pane = getContentPane();  
        pane.add(new JLabel("Hello !"));  
        pack();  
        setVisible(true);  
    }
```

Constructeur

```
    public static void main(String[] args) {  
        HelloWorldSwing mainWindow =  
            new HelloWorldSwing("title");  
    }
```

Instanciation

```
}
```

```
}
```

# Création d'une fenêtre en SWING

- En cinq étapes
  - 1. Création d'une fenêtre (par héritage)
  - 2. Récupération du container
  - 3. Modification du Layout
  - 4. Insertion des widgets dans le conteneur
  - 5. Arrangement des composantes

# Création d'une fenêtre

- JFrame : composant de haut niveau
  - contiendra toute l'application
  - première brique de l'interface
  - Une JFrame possède un conteneur par défaut.
- Implémente une nouvelle classe qui hérite de JFrame.
- Toujours créer un constructeur pour insérer les éléments dans cette fenêtre

```
public class HelloWorldSwing extends JFrame {
```

# Programmer une vue

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

public class HelloWorldSwing extends JFrame {

    HelloWorldSwing(String title) {
        super(title);
        Container pane = getContentPane();
        pane.add(new JLabel("Hello !"));
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        HelloWorldSwing mainWindow =
            new HelloWorldSwing("title");
    }
}
```

# Création d'une fenêtre en SWING

- En cinq étapes
  - 1. Création d'une fenêtre (par héritage)
  - 2. Récupération du container
  - 3. Modification du Layout
  - 4. Insertion des widgets dans le conteneur
  - 5. Arrangement des composantes

# Récupération du conteneur

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

public class HelloWorldSwing extends JFrame {

    HelloWorldSwing(String title) {
        super(title);
        Container pane = getContentPane(
            pane.add(new JLabel( "Hello ! ")),
            pack();
            setVisible(true);
        }

    public static void main(String[] args) {
        HelloWorldSwing mainWindow =
            new HelloWorldSwing("title");
    }
}
```

# Récupération du conteneur

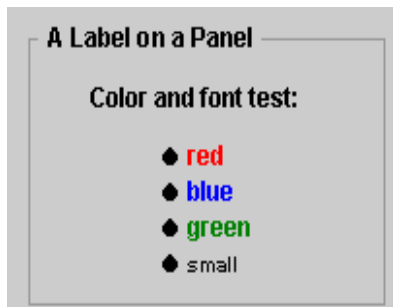
- Le conteneur permet d'afficher plusieurs objets graphiques
  - On peut insérer un ou plusieurs objets dans un même conteneur.
- Les objets sont placés dans le conteneur suivant les règles imposées par le layout
- Pour réaliser des interfaces plus complexes
  - on insère des conteneurs dans d'autres conteneurs pour spécifier plusieurs politiques de placement
- Les conteneurs les plus souvent utilisés sont les JPanel

# Récupération du conteneur

- Le panneau est un conteneur intermédiaire invisible
- Il a pour finalité de positionner d'après son "layout" les composants, boutons et les labels (JScrollPane, JTabbedPane).
- Son layout par défaut est un FlowLayout.

```
JPanel panel = new JPanel();
```

# Différents panels



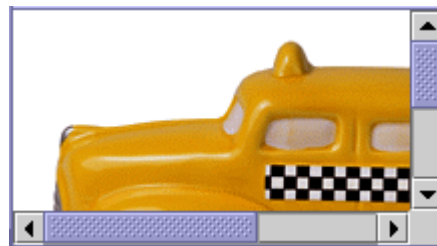
Panel



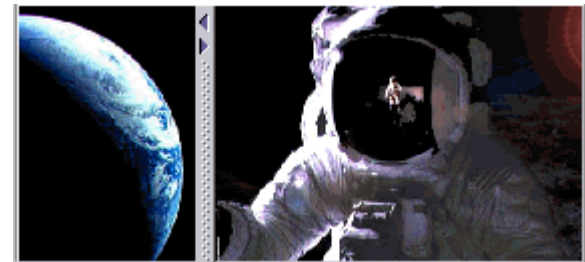
Tabbed pane



Tool bar



Scroll pane



Split pane

# Création d'une fenêtre en SWING

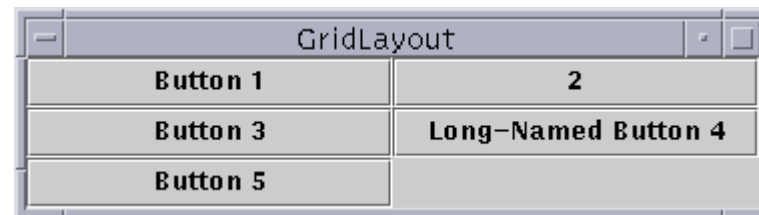
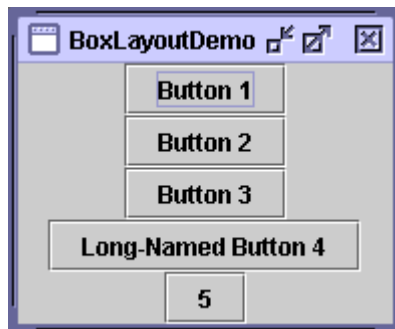
- En cinq étapes
  - 1. Création d'une fenêtre (par héritage)
  - 2. Récupération du container
  - 3. Modification du Layout
  - 4. Insertion des widgets dans le conteneur
  - 5. Arrangement des composantes

# Modification du conteneur

```
Container pane = getContentPane();  
pane.setLayout(new GridBagLayout());
```

# Layout

- Permettent de positionner les éléments avec une politique de placement
- Toujours liés aux conteneurs (JPanels, Tabbed pane).



- Il sont par défaut dans tous les composants (JFrame, JPanel). Il existe 7 types : FlowLayout, BorderLayout, GridLayout, BoxLayout, GridBagLayout, SpringLayout et le CardLayout.
- Chaque layout est une classe Java :
- Pour créer un layout : déclaration puis instantiation

```
FlowLayout layout = new FlowLayout();
```

- Les composants utilisent souvent par défaut un BorderLayout (JFrame, JDialog, JApplet).
- Les Layouts sont utilisés surtout avec les panneaux (JPanel) et les fenêtres
  - On peut affecter un layout dans un panneau au moment de l'instanciation :
  - Affectation d'un layout dans le conteneur d'une fenêtre "frame":

```
Jpanel panel = new Jpanel(new BorderLayout());  
Container pane = frame.getContentPane();  
pane.setLayout(new FlowLayout());
```

# Programmer une vue

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

public class HelloWorldSwing extends JFrame {

    HelloWorldSwing(String title) {
        super(title);
        Container pane = getContentPane();
        pane.add(new JLabel("Hello !"));
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        HelloWorldSwing mainWindow =
            new HelloWorldSwing("title");
    }
}
```