Java
Approfondissement

Arnaud Delafont

08/07/19

Plan

- Interfaces graphiques
- Threads
- Connexion aux bases de données
- UML, Design Patterns
- Spécificités de la plate-forme Java

Interface Graphique

Définition

- Swing est une partie de « Java Foundation Class »
  - Bibliothèques produisant des interfaces graphiques.
  - Drag & Drop, i18n, Java 2D, Accessibility, AWT
- Avantages de Swing par rapport à AWT
  - Résout les problèmes de portabilité
  - AWT est utilisé dans les couches basses de Swing

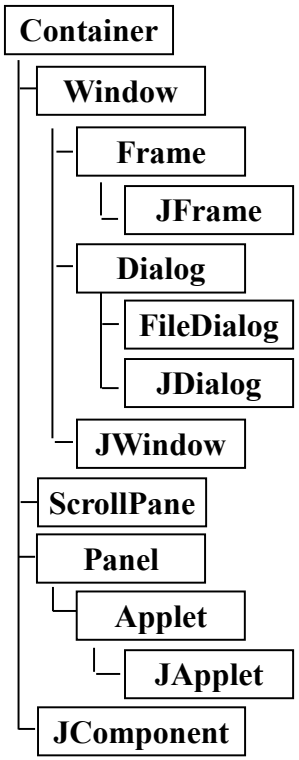
Outils WYSIWYG

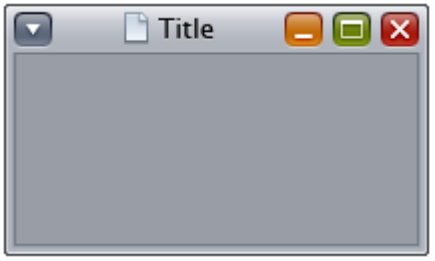
- What You See Is What You Get

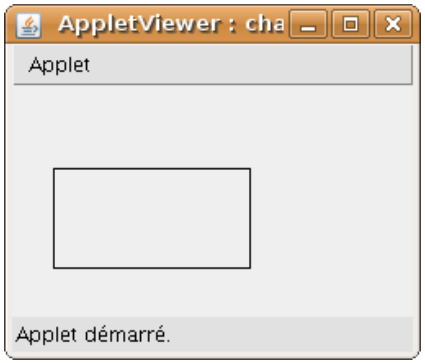
- Certains IDE fournissent leur propre plugin
  - Visual Editor Plugin (VEP) : le plugin officiel, gratuit, gère swing et SWT.
  - Swing designer, SWT designer et Window Builder qui englobe les deux précédents : parmi les meilleurs. Payants, ils gèrent Swing et/ou SWT (en fonction des versions).
  - Jigloo Gratuit pour usage non commercial, gère SWT et Swing

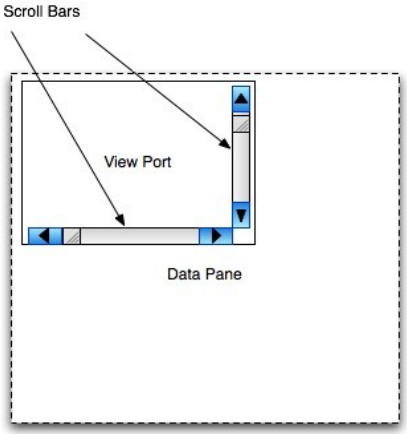
Conteneurs

Un Conteneur peut regrouper plusieurs composants









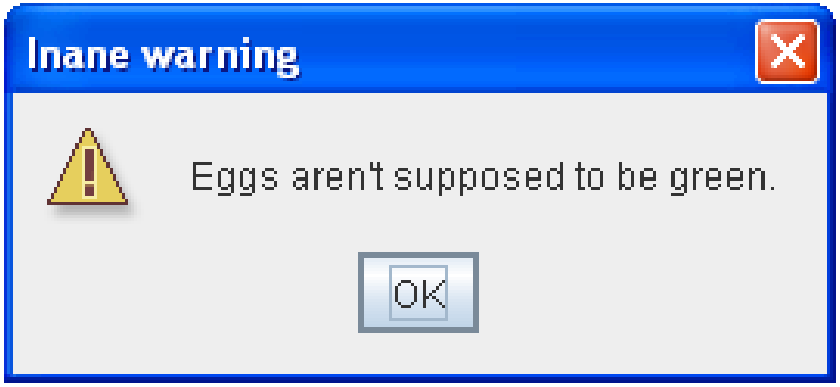
JFrame

Créer une fenêtre principale visible :

```
…
public static void main(String [] a) {
   //décoration pour toutes les fenêtres
   JFrame.setDefaultLookAndFeelDecorated(true);
   JFrame jf = new JFrame("ma fenêtre");

   jf.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);

   jf.getContentPane().add(new JButton);
   //autres creations…

   jf.pack();
   jf.setVisible(true);
}
…
```

Boîtes de message

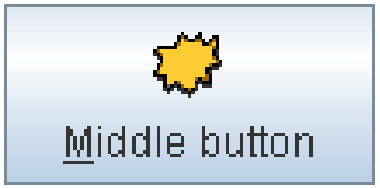
- Parmis les conteneurs
- Déclenchées depuis un conteneur (la fenêtre « mère »)
- Modales




```
JOptionPane.showMessageDialog(jf,
    "Les oeufs ne doivent pas être verts.",
    "Avertissement",
    JOptionPane.WARNING_MESSAGE);
```

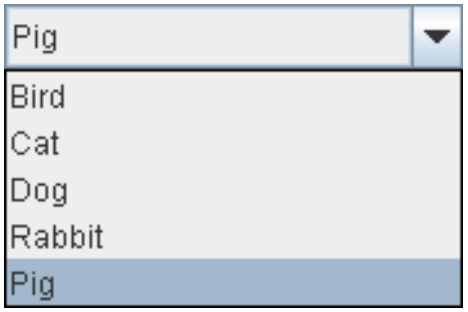
Composants simples

- JButton



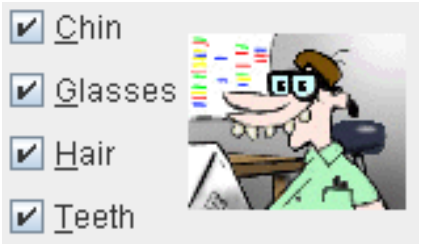
- JLabel et JTextfield

- JComboBox

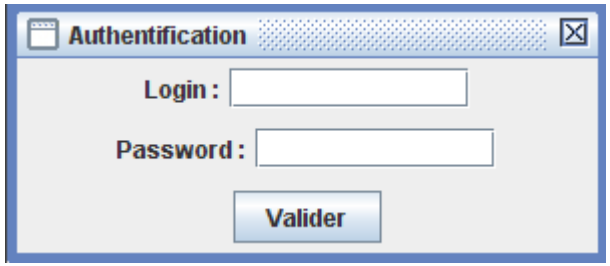


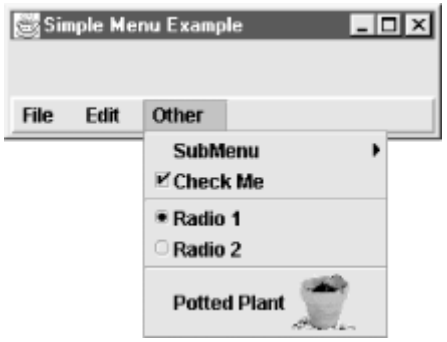
- JList

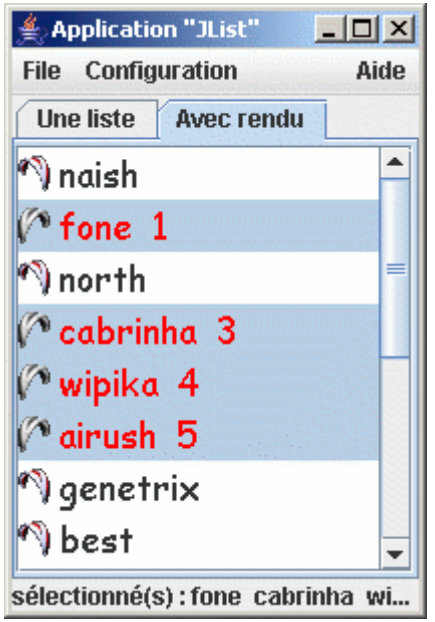
- JCheckBox

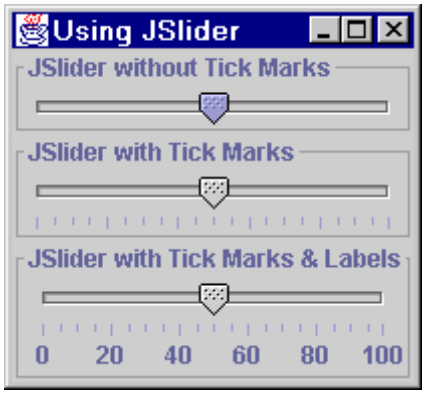


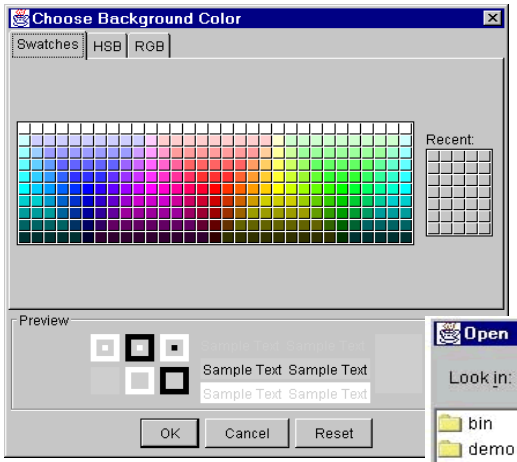
Widgets

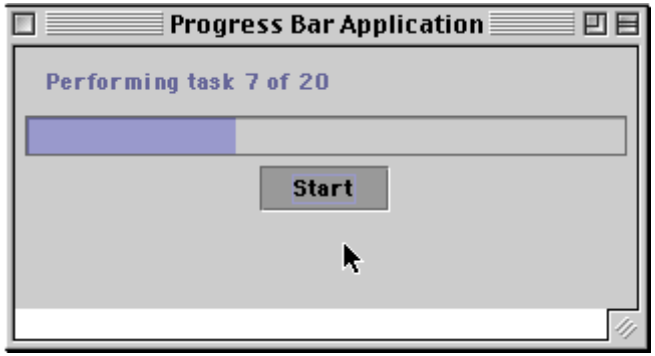

Authentification
Login :
Password :
Valider

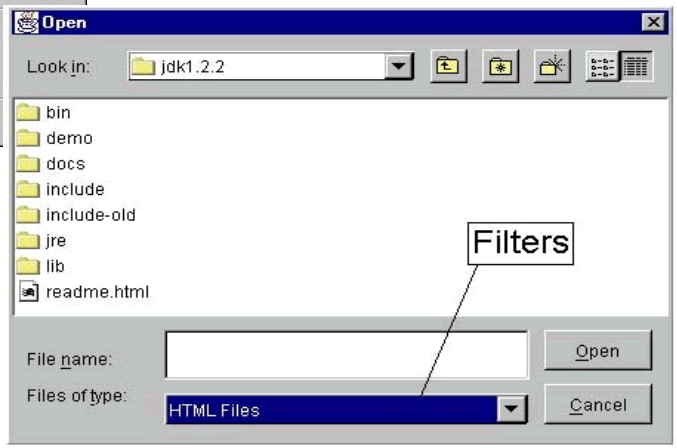

Simple Menu Example
File
Edit
Other
SubMenu
Check Me
Radio 1
Radio 2
Potted Plant


Application "JList"
File
Configuration
Aide
Une liste
Avec rendu
naish
fone 1
north
cabrinha 3
wipika 4
airush 5
genetrix
best
sélectionné(s) : fone cabrinha wi...


Using JSlider
JSlider without Tick Marks
JSlider with Tick Marks
JSlider with Tick Marks & Labels
0
20
40
60
80
100


Choose Background Color
Swatches
HSB
RGB
Recent:
Preview
Sample Text Sample Text
OK
Cancel
Reset


Progress Bar Application
Performing task 7 of 20
Start


Open
Look in:
jdk1.2.2
bin
demo
docs
include
include-old
jre
lib
readme.html
Filters
File name:
Files of type:
HTML Files
Open
Cancel

Positionnement des composants

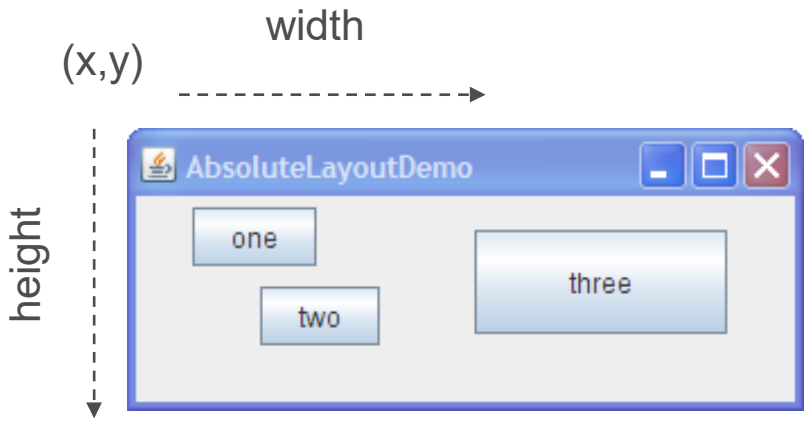
- Il existe plusieurs gestionnaires de positionnement (Layout Managers) :

  - Positionnement absolu
  - *FlowLayout*
  - *BorderLayout*
  - *GridLayout*
  - *GroupLayout*

Positionnement Absolu

Le programmeur détermine manuellement l'emplacement de chaque composant :

- Coordonnées :
  - x
  - y
  - width
  - height




```
Dimension dimension =
     jComponent.getPreferredSize ();

jComponent.setBounds (x, y, width, height);
```

Positionnement Absolu

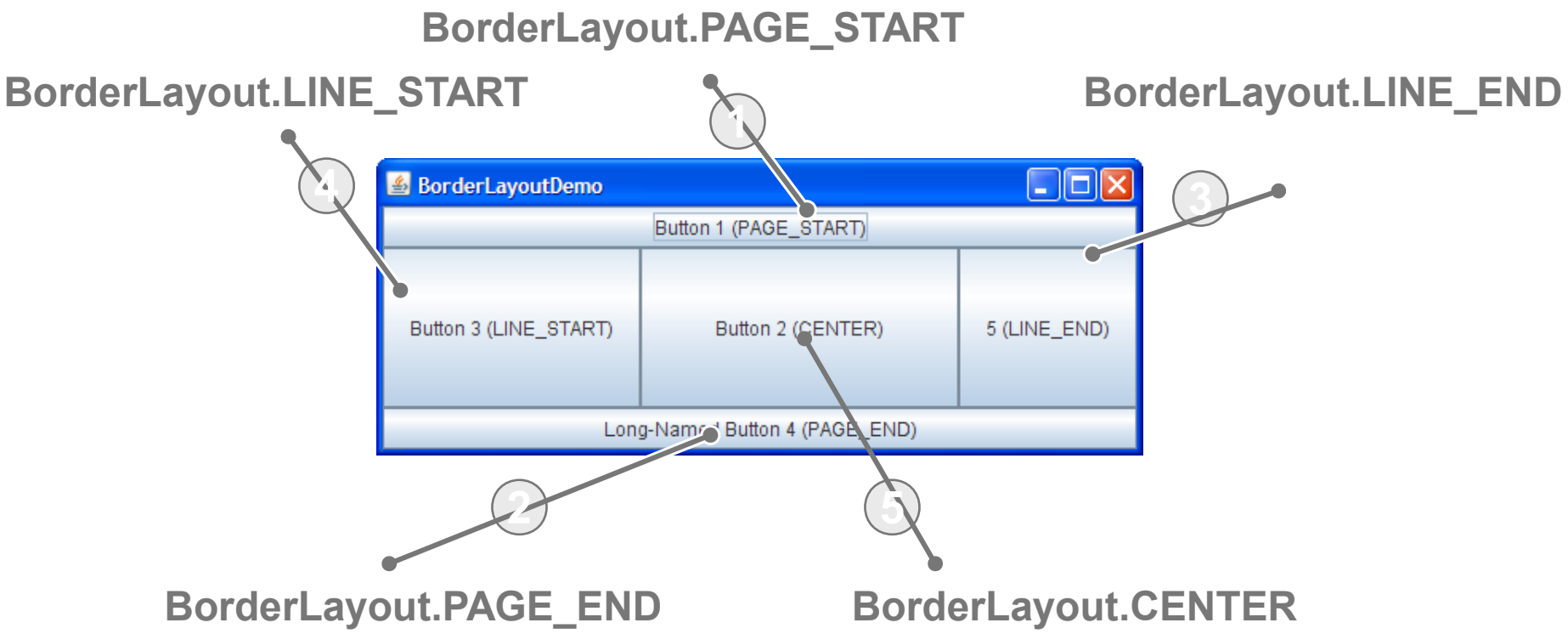
```
[…]
public void initialise( )  {
   // choisir un layout absolu
   setLayout(null);
   // observer la taille préférée
   Dimension size = jLabel.getPreferredSize();
   // et l'utiliser
   jLabel.setBounds(25, 5, size.width, size.height);
   size = getJTextField().getPreferredSize();
   getJTextField().setBounds(25, 20, size.width, size.height);
   size = getJButton().getPreferredSize();
   getJButton().setBounds(25, 40, size.width, size.height);
   // choisir la taille du conteneur (après pack())
   setSize(300 , 125);
}
[…]
```

FlowLayout

- Différents LayoutManagers utilisent des politiques différentes pour le placement des composants
- **Flow layout** arrange les composants de gauche à droite jusqu'à ce que la ligne soit remplie
- Chaque ligne est centrée
- C'est le layout par défaut des panneaux et du *contentPane*

BorderLayout

Il divise le conteneur en 5 régions - dans chaque région, un seul composant (qui occupe toute la place) :

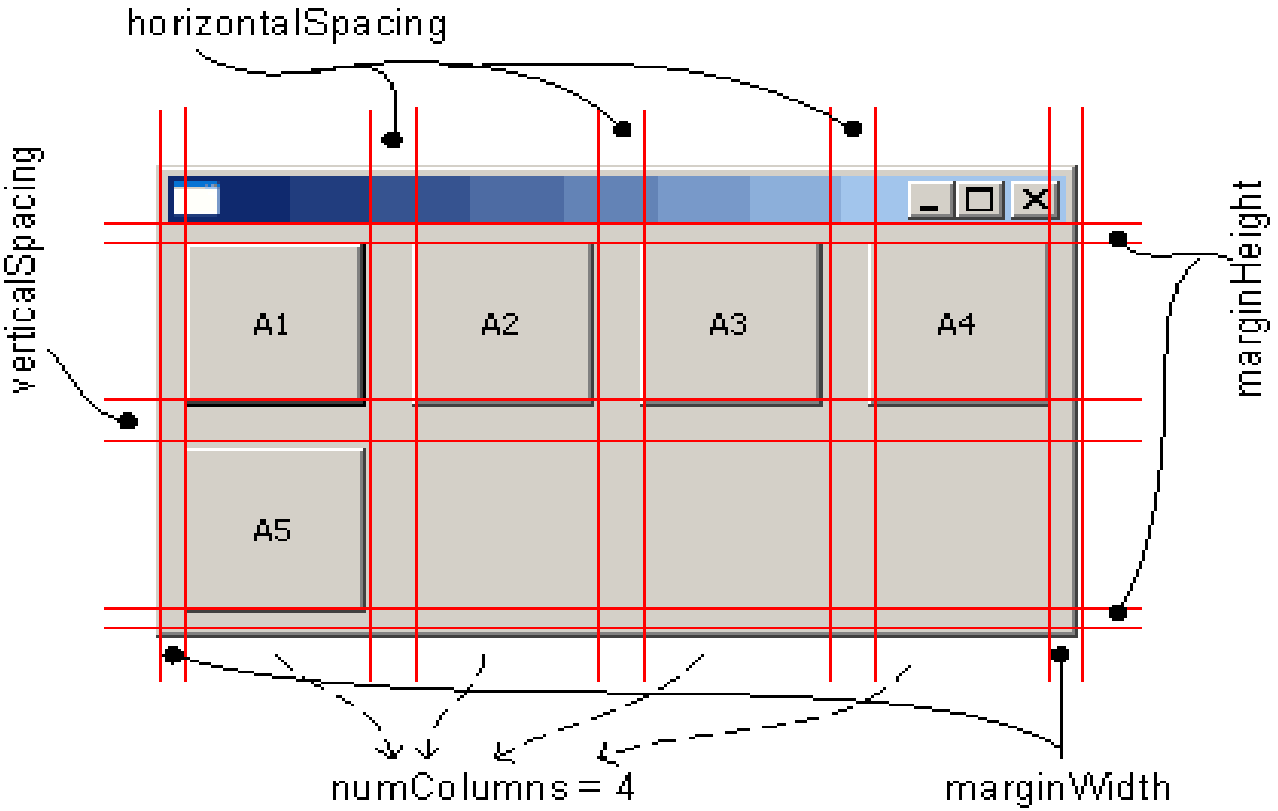


BorderLayout

```
[…]
public void initialise( )  {
      …
       JPanel p = new JPanel();
       p.setLayout(new BorderLayout());
       p.add(new Jlabel("Gauche <-",
              BorderLayout.LINE_START);
       p.add(new Jlabel("Bas",
              BorderLayout.PAGE_END);
       …
       ....add(p);
   }
[…]
```

GridLayout

Gère le conteneur comme un tableau.
L'insertion se fait soit de gauche à droite ou le contraire



GridLayout

```
[…]
   GridLayout gridLayout = new GridLayout();
   gridLayout.setRows(1);
   gridLayout.setColumns(2);
   jPanel = new JPanel();
   jPanel.setLayout(gridLayout);
   jPanel.add(getJButton());
   jPanel.add(getJButtonReset());
    //le panneau est rempli
[…]
```

Autres Layouts

D'autres gestionnaires de positionnement existent mais sont moins utilisés :

- *CardLayout*
- *GridBagLayout*
- *BoxLayout*
- *OverlayLayout*
- *ScrollPaneLayout*
- *SpringLayout*
- *ViewPortLayout*

Gestion des Evènements

- Pour intercepter un événement, un listener doit être associé au composant
- Le listener est appelé lors de l'évènement
- Il peut y avoir plusieurs listener pour un même événement sur un même composant

Listener 1

Actions sur un JButton

ActionListener est une interface

Chaque composant proposant des événements "action" dispose de :

- addActionListener(ActionListener l)
- removeActionListener(ActionListener l)
- ActionListener[] getActionListeners()

Classe ActionEvent

- Appartient au package java.awt.event
- Permet de gérer les évènements
- Il existe d'autres types de Listener dans java.awt.event :
  - FocusListener
  - WindowListener
  - ContainerListener
  - MouseListener
  - KeyListener
  - etc.

Interface ActionListener

Association du Listener et actions en même temps (classe anonyme) :

```
JButton jButton = new Jbutton("Cliquer !");

// Ajout d'un action listener
jButton.addActionListener(
   new java.awt.event.ActionListener() {
      public void actionPerformed(
         java.awt.event.ActionEvent e) {
   // Faire une action quand il est cliqué
      }
   }
);
```

Interface ActionListener

Plus propre, dans une autre classe :

```
public class SubscriptionAction implements
ActionListener {
   public void actionPerformed(ActionEvent e) {
      // action command permet d'utiliser un
      //Listener avec plusieurs composants
      if(e.getActionCommand().equals("quitter"))
      // Action Exit
      }
}
```

Dans l'interface graphique :

```
jButton.setActionCommand("quitter");
jButton.addActionListener(new
    SubscriptionAction());
```

Application

- Test de plusieurs actions :

  Implémentation d'ActionListener, MouseListener ...

Threads

Threads

- Un processus léger (unité réelle d'exécution d'un programme)
- Exécute une ou plusieurs tâches
- Partage de mémoire et de temps CPU

Avantages :

- Permettent de faire du multi-tâche
- Faire des boucles infinies
- Création de processus fils.

Utilisation basique des threads

Création et la manipulation de threads

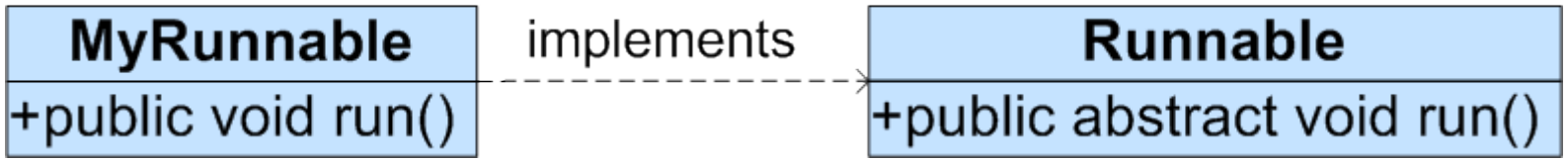
- L'interface **Runnable**
- La classe **Thread**

(package : java.lang)

L'interface Runnable

L'écriture d'un Thread nécessite l'implémentation de l'interface Runnable et la surcharge de la méthode run() :

```
public class MyRunnable implements Runnable{…}
```




```
public void run() {// Instructions }
```

Pour instancier notre Thread, on écrira :

```
Runnable myRunObject = new MyRunnable ();
Thread thread = new Thread (myRunObject);
```

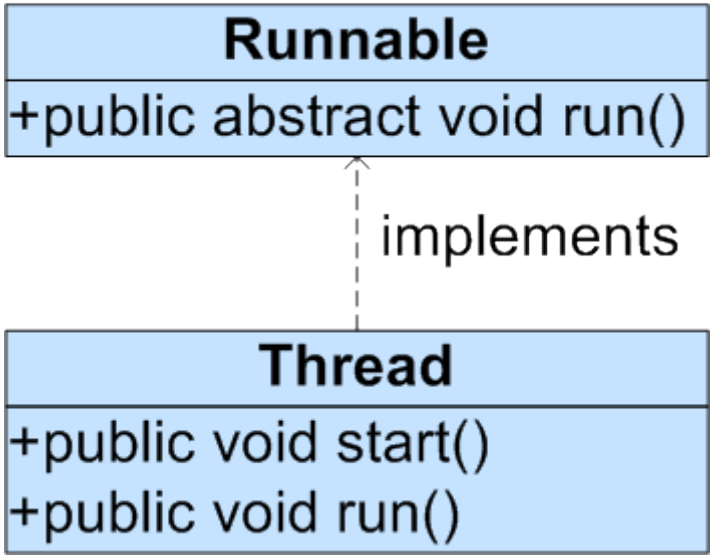
La classe Thread

Définition :

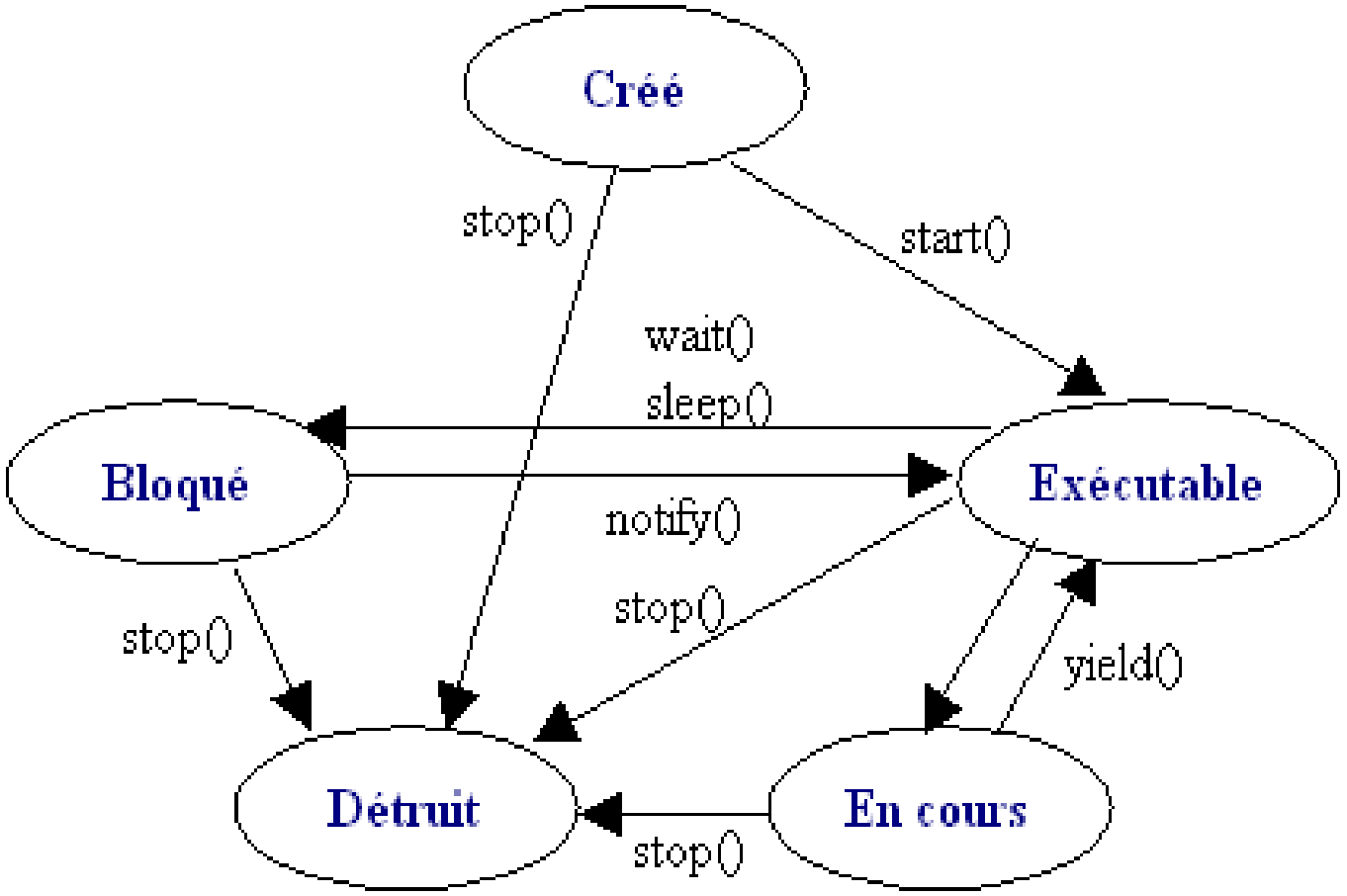
- Représente un thread d'exécution
- La classe **Thread** implémente l'interface **Runnable**

Un thread :

- Possède un nom et une priorité
- Peut être un *deamon* ou non.
- Possède plusieurs états



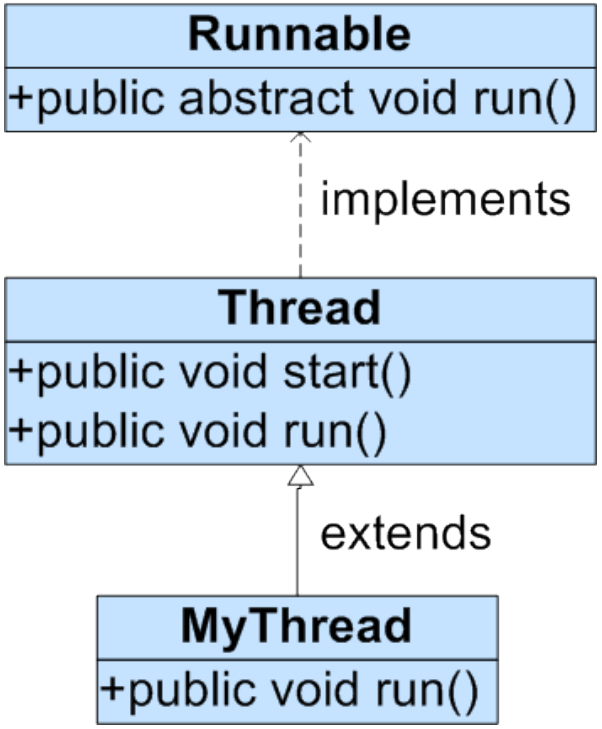
Etats d'un thread



La classe Thread

Ecrire ses propres threads :

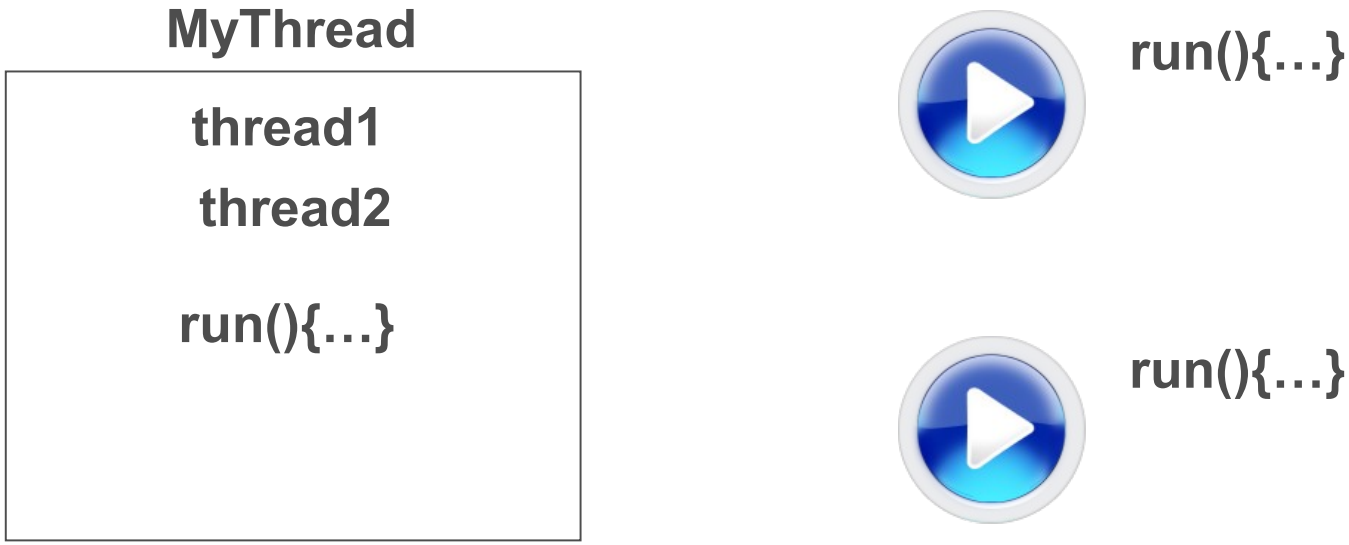
```
public class MyThread extends Thread {…}
```



Utilisation des threads

```
MyThread thread1 = new MyThread ();
MyThread thread2 = new MyThread ();
thread1.start();
thread2.start();
```

- Simulation



Utilisation des threads

Exemple de pause d'un thread en exécution :

RunningThread.java avec **sleep**()

```
public class RunningThread extends Thread{

 public void run(){
   try{
      for (int i=0;i<3;i++){
            Thread.sleep(1000);
            System.out.println(getName()+" is
                  running");
      }
   }catch(InterruptedException e){}
 }
}
```

UML

Unified Modeling Language

UML = Language pour la modélisation des classes, des objets, des intéractions etc...

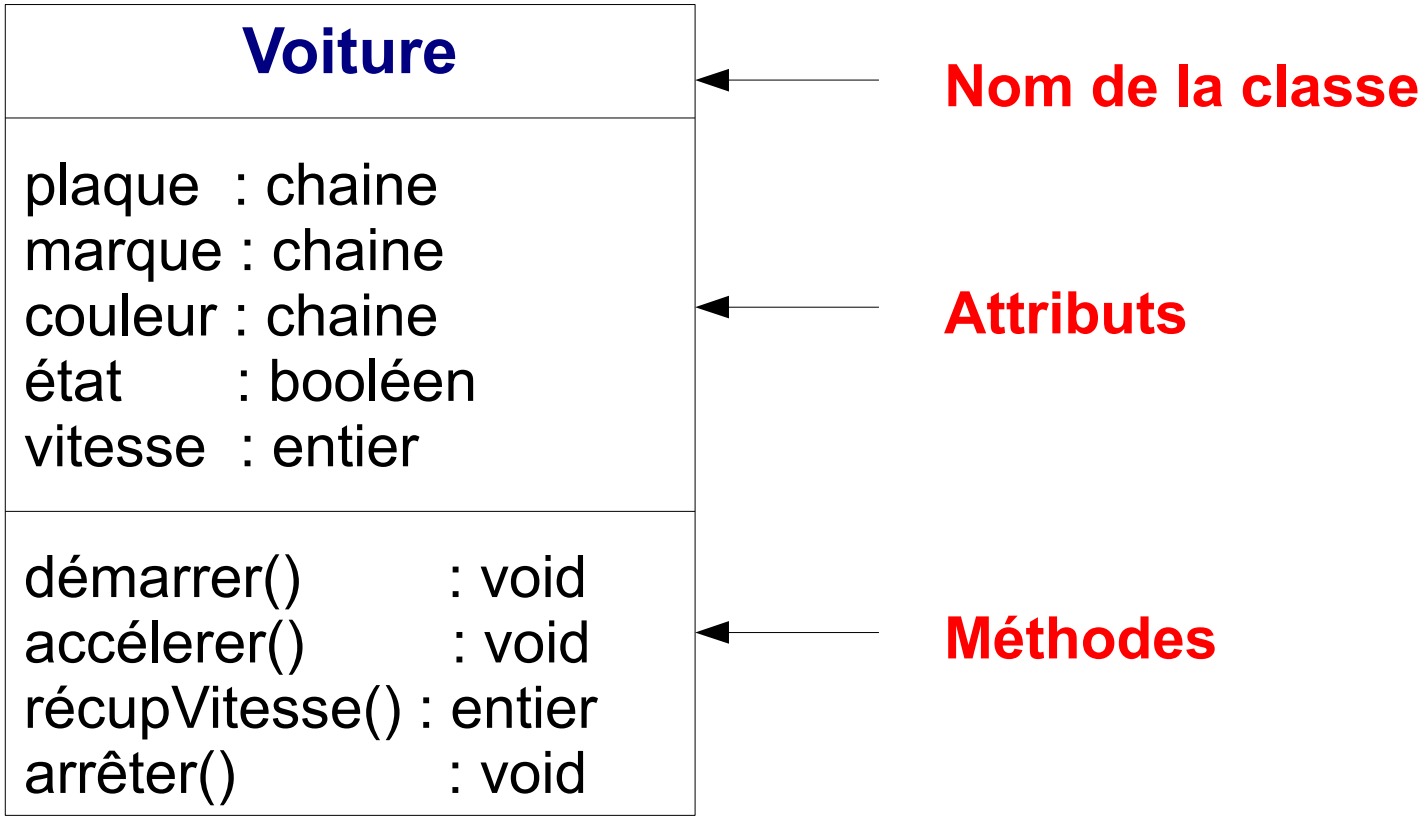
UML 2.0 comporte ainsi treize types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information :

- Diagramme fonctionnel

- Diagrammes structurels (statiques)

- Diagrammes comportementaux (dynamiques)

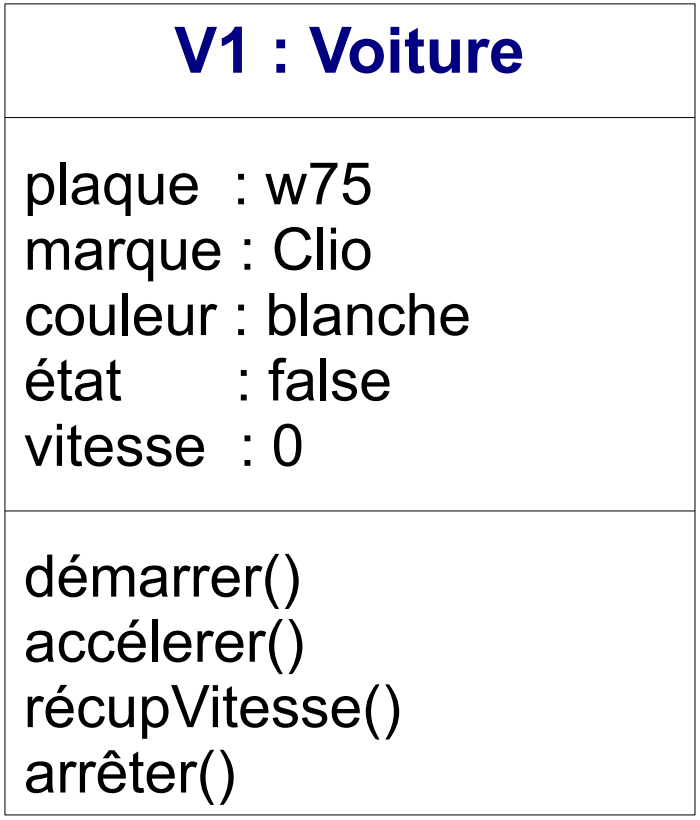
Représentation UML d'une Classe



Les attributs ou les méthodes peuvent être précédés par un opérateur (+, #, -) pour indiquer le niveau de visibilité

Représentation UML d'un Objet

Un objet est représenté graphiquement par un rectangle à coins divisé en 3 parties pour inscrire l'identité, les champs et les opérations.



L'accès aux membres d'un objet s'effectue grâce au point (.) :

v1.plaque
v1.démarrer()

Design Patterns

- Solution standard pour un problème d'architecture
  - Solution à disposition
  - Eprouvée, de qualité
  - Connue, permettant la communication
- A appliquer au code, particulièrement de POO
- 23 Design Patterns fondamentaux (« GoF »)
- Ex. courants en Java : Factory, Listener, Iterator

Spécificité Plateforme java

Javadoc

- Outil permettant de générer une documentation technique
- Utiliser des commentaires /** … */ et quelques annotations. Ex : @author

```
/**
* Calcul de la somme de 2 entiers.
* @author Mohamed DERKAOUI
* @param an integer a
* @param an integer b
* @return the sum of the two param (a+b)   */

int somme(int a, int b) {
    return a+b;
}
```

JAR : Java ARchive

Permet de regrouper :

- Fichiers *.class*
- Images, resources diverses
- Fichiers de configuration

- JAR exécutable avec un fichier manifest
- jar [options] [manifest] destination input-file [input-file]

```
jar cvf game.jar JavaProgram.class Game.class
java -jar game.jar
```

Décompilation

- Récupérer le code source à partir du *bytecode*
- javap, inclus dans le JDK

JDBC - présentation

- Préparation : télécharger un pilote (Java DataBase Connectivity)

- Ouvrir une connexion
  - Chargement dynamique de la classe du pilote
  - Paramètres dépendants du type de base

- Exécuter des requêtes SQL sur cette connexion

- Utiliser les résultats (s'il y en a)

- Fermer la connexion

- Prévoir et traiter les exceptions

JDBC - classes

- java.lang.DriverManager : Capable d'instancier une autre classe (du classpath) dynamiquement
- java.sql.Connection : une connexion à la base
  - *close()* : fermer la connection
- java.sql.Statement : une requête
  - *executeUpdate()* : exécuter la requête
  - *executeQuery()* : exécuter la requête avec résultat
- java.sql.ResultSet : ensemble de résultats (curseur)
  - *next()* : passer au suivant

JDBC - Exemple

```
try {
   Class.forName( "com.mysql.jdbc.Driver" );
   con = DriverManager.
      getConnection(
         "jdbc:mysql://localhost/test",
         "root", "");
   Statement st = con.createStatement();
   ResultSet rs = st.executeQuery(
      "SELECT * FROM bidules");
   while(rs.next()) {
      ...rs.getString(1)...
   }
   con.close();
}catch(...) { ... }
```

Mini-projet

Réalisation d'une application CARNET D'ADRESSES :

- Création de la classe Contact représentant un individu

- Création de la classe Carnet implémentant une collection de contacts

- Implémentation de méthodes permettant l'ajout, la modification, la suppression et la recherche d'un contact dans le carnet de contacts

- Création de l'interface graphique avec Swing

- Connexion à une BDD MySQL pour le stockage des données

- Utilisation des Design Pattern DAO et Singleton.