Java pour l'embarqué. Application pour l'Internet des objets et pour smartcards

Langage Java : l'essentiel pour l'embarqué

email : kadionik@enseirb-matmeca.fr
web : http://kadionik.vvv.enseirb-matmeca.fr

Patrice KADIONIK
ENSEIRB-MATMECA

PARTIE 1
INTRODUCTION

Introduction

- Ce cours présente l'essentiel sur le langage Java dans un contexte embarqué. Il ne s'agit pas de devenir un expert Java !
- Les concepts de la programmation orientée objet sont supposés connus à travers les cours sur le langage C++ et indirectement sur le langage VHDL. L'apprentissage se fera donc par analogie…
- L'essentiel du langage Java permettra de faire des travaux pratiques orientés systèmes embarqués sur une cible objet connecté RPi : Java ME.

PARTIE 2
PRESENTATION GENERALE

Qu'est-ce que le langage Java ?

- Un langage de programmation orienté objet comme le langage C++.
- Une architecture de machine virtuelle par simulation d'un processeur universel.
- Un ensemble complet d'API (*Application Programming Interface*) ou paquetages (*packages*).
- Un ensemble d'outils : le JDK (*Java Development Kit*).

Historique

- **1993** : projet *Oak* de Sun : langage de programmation pour l'électronique grand public, commande à distance par Internet.
- **1995** : Java.
- **Sep 95** : JDK 1.0 b1.
- **Jan 96** : JDK 1.0.1.
- **Aoû 96** : Java Study Group ISO/IEC JTC 1/SC22.
- **Fin 96** : RMI, JDBC, JavaBeans (composant logiciel)...
- **Fév 97** : JDK 1.1.
- **Aoû 98** : JDK 1.2 (JFC). Java 2. Swing.
- **2014** : JDK 8 (1.8). Java 8.

Les fausses idées sur Java

- Java n'a rien de commun avec HTML.
- Java n'est pas un langage de script comme sh, Perl ou TCL.
- Java n’est pas JavaScript. C'est un langage généraliste orienté objet.
- Java n’est pas C++. C'est un langage purement objet de plus haut niveau.
- Java permet des développements logiciels importants (Android).

Les points faibles de Java

- Les JVM sont lentes. C’est un langage interprété. Mais des processeurs performants peuvent compenser la lenteur de l’interprétation…
- Java est gourmand en mémoire.
- Java ne permet pas d’accéder directement à des adresses (pointeurs), besoin important dans l’embarqué pour contrôler du matériel. On pourra alors utiliser l’API JNI (*Java Native Interface*).

Caractéristiques de Java

- Orienté objet.
- Interprété.
- Portable.
- Simple.
- Robuste.
- Sécurisé.
- *Multithreads*.
- Distribué (RMI, Corba, JDBC…).

Write Once, Run Anywhere!

Java est un langage orienté objet

- Tout est classe (pas de fonctions) sauf les types primitifs (`int`, `float`, `double`...) et les tableaux.
- Toutes les classes dérivent de la classe `java.lang.Object`
- **Héritage simple pour les classes.**
- **Héritage multiple pour les interfaces.**
- Les objets se manipulent via des références (instances).
- Une API objet standard est fournie.
- **La syntaxe reste proche de celle du langage C.**

Java est portable

- Le compilateur Java génère du *byte code* (binaire pour un microprocesseur virtuel 8 bits).
- La JVM (*Java Virtual Machine*) existe pour *NIX, Windows, Mac, Firefox, IE...
- Le langage a une sémantique très précise (langage fortement typé comme le langage VHDL).
- La taille des types primitifs est indépendante de la plateforme (même taille d'un `int` quelle que soit la plateforme).

Java est robuste

- A l'origine, c'est un langage pour les applications embarquées (Java veut dire café, langage créé pour piloter une machine à café…).
- Gestion de la mémoire par le ramasse-miettes (*Garbage Collector*) donc pas de *malloc() / free()* comme en langage C.
- Pas d'accès direct à la mémoire.
- Mécanisme de gestion d'exception.
- L'accès à une référence `null` génère une exception.
- Compilateur contraignant (erreur si exception non gérée, si utilisation d'une variable non affectée...).
- Tableau = objet (taille connue, le débordement génère une exception).
- Seules les conversions sûres sont automatiques (*cast*).

Java est sécurisé

- Indispensable avec un téléchargement du code possible par le réseau (*applet*).
- Pris en charge dans l'interpréteur.
- Trois couches de sécurité :
  - *Verifier* : vérifie le *byte code.*
  - *Class Loader* : responsable du chargement des classes.
  - *Security Manager* : accès aux ressources locales (disque, ports I/O…).
- Code pouvant être signé par une clé.

Java est multithread

- Intégré au langage et aux différentes API :
  - Mot clé `synchronized` (synchronisation de *threads*).
  - Le ramasse-miettes est un *thread* de basse priorité.
  - *Packages* `java.lang.Thread`, `java.lang.Runnable`.
- Implémentation propre à chaque JVM. La JVM est donc à voir comme un OS multitâche…

Java est distribué

- API réseau (`java.net.Socket, java.net.URL`...).
- *Applet* (exécution local par le navigateur Internet)..
- *Servlet* (exécution à distance par un serveur Web d'une application Java (remplace les scripts CGI moins sécurisés).
- RMI (*Remote Method Invocation*). Exécution de procédures distantes / objets distribués (analogie avec les RPC/XDR).
- CORBA (*Common Object Request Broker Architecture*) : objet distribué.

Performances

- Le *byte code* est interprété.
- Il y a des processeurs exécutant directement du *byte code* (processeurs Sun).

Différences avec C++

- Pas de structures ni d'unions.
- Pas de types énumérés.
- Pas de *#typedef*.
- Pas de préprocesseur.
- Pas de variables ni de fonctions en dehors des classes.
- Pas de fonctions à nombre variable d'arguments.
- Pas d'héritage multiple de classes (héritage simple).
- Pas de surcharge d'opérateurs.
- Pas de passage par copie pour les objets.
- Pas de pointeurs, seulement des références.

On a gardé le meilleur du C++ sans s'encombrer des choses inutiles ou dangereuses !

Les outils

- **Environnements de développement** :
  - Sun JDK (compilateur, interpréteur, *applet viewer*…).
  - Emacs, vi.
  - IDE (*Integrated Design Entry*) : NetBeans, Eclipse, CodeWarrior…
- **Navigateurs Internet** :
  - Firefox.
  - Internet Explorer.
- **JVM :**
  - Oracle, Kaffe, Cacao, Harissa...

Les outils : le Java Development Kit

- **javac** : compilateur de sources java.
- **java** : interpréteur de *byte code*.
- **appletviewer** : interpréteur d'*applet*.
- **javadoc** : générateur de documentation (HTML).
- **javah** : générateur de *headers* et de *stubs* pour l'appel des méthodes natives.
- **javap** : désassembleur de *byte code*.
- **jdb** : *debugger*.
- **javakey** : générateur de clés pour la signature de code (sécurisation).
- **rmic** : compilateur de *stubs* RMI.
- **rmiregistry** : enregistrement d'un service RMI (*portmapper* des RPC).

Les Core API

- Les *Core API* sont les paquetages (*packages*) de Java :

- **java.lang** : types de base, *threads*, *ClassLoader*, *Exception*, math...
- **java.util** : *stack*, date...
- **java.applet** : *applet*.
- **java.awt** : AWT (*Abstract Window Toolkit*) pour IHM portables.
- **java.io** : accès aux E/S.
- **java.net** : API *socket*, URL...

Les Core API

- **java.beans** : composants logiciels (accès standardisés aux propriétés d'un objet).
- **java.sql** (*Java DataBase Connectivity*) : accès aux bases de données.
- **java.security** : signature, cryptographie, authentification.
- **java.serialisation** : sérialisation d'objets.
- **java.rmi** : *Remote Method Invocation.*
- ...

Les autres API

- **Java Server :** *servlets.*
- **Java Média** : 2D, 3D, animation…
- …

Les différents JDK

- **Java SE :** *Java Platform, Standard Edition lets you develop and deploy Java applications on desktops and servers.*
- **Java EE** : *Java Platform, Enterprise Edition is the standard in community-driven enterprise software.*
- **Java ME** : *Java Platform, Micro Edition provides a robust, flexible environment for applications running on embedded and mobile devices in the Internet of Things: micro-controllers, sensors, gateways, mobile phones, personal digital assistants (PDAs), TV set-top boxes, printers and more.*

Les différents JDK

- **Java Card :** *Java Card technology provides a secure environment for applications that run on smart cards and other devices with very limited memory and processing capabilities. Multiple applications can be deployed on a single card, and new ones can be added to it even after it has been issued to the end user.*

PARTIE 3
ELEMENTS DU LANGAGE

Les types primitifs

- `Boolean` (*true/false*), `byte` (1 octet), `char` (2 octets), `short` (2 octets), `int` (4 octets), `long` (8 octets), `float` (4 octets), `double` (8 octets).
- Les variables peuvent être déclarées n'importe où dans un bloc (à éviter).
- Les affectations non implicites doivent être *castées* (sinon erreur à la compilation).

```
int i = 258;
long l = i;          // ok
byte b = i;          // error: Explicit cast needed to convert int to byte
byte b = 258;        // error: Explicit cast needed to convert int to byte
byte b = (byte)i;    // ok mais b = 2
```

Les structures de contrôle et expressions

- Essentiellement les mêmes qu'avec le langage C :
  - `if`, `switch`, `for`, `while`, `do while`
  - `++`, `+=`, `&&`, `&`, `<<`, `?:`

Les tableaux

- Déclaration :

```
int[] array_of_int;              // équivalent à : int array_of_int[];
Color rgb_cube[][][];
```

- Création et initialisation (allocation dynamique de mémoire) :

```
array_of_int = new int[42];      // équivalent
                                 // malloc(42*sizeof(int)) en C
rgb_cube = new Color[256][256][256];
int[] primes = {1, 2, 3, 5, 7, 7+4};
array_of_int[0] = 3
```

- Utilisation :

```
int l = array_of_int.length;     // l = 42
```

Les exceptions

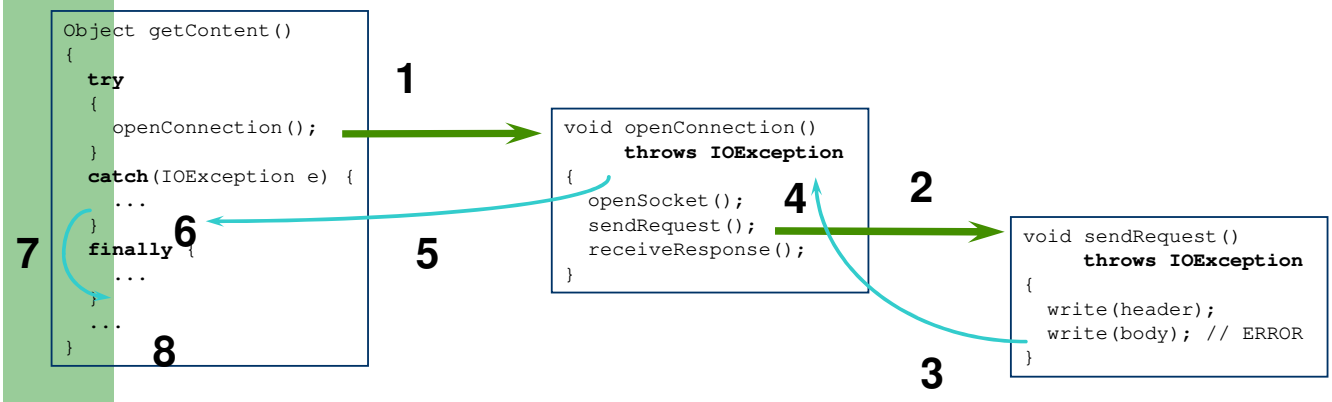
- Elles permettent de séparer un bloc d'instructions de la gestion des erreurs pouvant survenir lors de l'exécution de ce bloc :

```
try {
   // Code pouvant lever des IOExceptions
}
catch (IOException e) {
   // Gestion des IOExceptions
}
catch (Exception e){
   // Gestion de toutes les autres exceptions
}
```


Les exceptions

- Ce sont des instances de classes dérivant de `java.lang.Exception`.
- La levée d'une exception provoque une remontée dans l'appel des méthodes jusqu'à ce qu'un bloc `catch` acceptant cette exception soit trouvé.
- Si aucun bloc `catch` n'est trouvé, l'exception est capturée par l'interpréteur et le programme s'arrête.
- L'appel à une méthode pouvant lever une exception doit :
  - Soit être contenu dans un bloc `try/catch`
  - Soit être situé dans une méthode propageant (`throws`) cette classe d'exception.
- Un bloc (optionnel) `finally` peut-être posé à la suite des `catch`. Son contenu est exécuté après un `catch`.

Les exceptions



Les unités de compilation

- Le code source d'une classe est appelé unité de compilation.
- Il est recommandé (mais pas imposé) de ne mettre qu'une classe par unité de compilation.
- L'unité de compilation (le fichier) doit avoir le même nom que la classe qu'elle contient.

Les packages

- Un *package* regroupe un ensemble de classes sous un même nom (équivalent à un *package* VHDL).
- Les noms des *packages* suivent le schéma : `name.subname` (identique en VHDL : `USE IEEE.ALL ;`).
- Une classe `Watch` appartenant au *package* `time.clock` doit se trouver dans le fichier : `time/clock/Watch.class`.
- Les *packages* permettent au compilateur et à la JVM de localiser les fichiers contenant les classes à charger.
- L'instruction `package` indique à quel *package* appartient la ou les classes de l'unité de compilation (le fichier).

Les packages

- Les répertoires contenant les *packages* doivent être présents dans la variable d'environnement `CLASSPATH`.
- En dehors du *package*, les noms des classes sont repérés par : `packageName.className`.
- L'instruction `import` *`packageName`* permet d'utiliser des classes sans les préfixer par leur nom de *package*.
- Les API sont organisées en *package* (`java.lang`, `java.io`...).

Les packages

```
CLASSPATH = $JAVA_HOME/lib/classes.zip;$HOME/classes
```

~/classes/graph/2D/Cercle.java

```
package graph.2D;
public class Cercle()
{ ... }
```

~/classes/graph/3D/Sphere.java

```
package graph.3D;
public class Sphere()
{ ... }
```

~/classes/paintShop/MainClass.java

```
package paintShop;

import graph.2D.*;

public class MainClass()
{
  public static void main(String[] args) {
    graph.2D.Cercle c1 = new graph.2D.Cercle(50)
    Cercle c2 = new Cercle(70);
    graph.3D.Sphere s1 = new graph.3D.Sphere(100);
    Sphere s2 = new Sphere(40);      // error: class paintShop.Sphere not found
}
```


Exemple de programme

```
class Cercle(){
   public double x, y;      // Coordonnée du centre
   private double r;        // rayon du cercle

   public Cercle(double r) {
       this.r = r;
   }
   public double area() {
       return 3.14159 * r * r;
   }
}

public class MonPremierProgramme() {
   public static void main(String[] args) {
       Cercle c;             // Référence sur un objet Cercle, pas un objet
       c = new Cercle(5.0); // Instance d'un objet alloué en mémoire
       c.x = c.y = 10;
       System.out.println("Aire de c :" + c.area());
   }
}
```

Création d'un objet

- Pour manipuler un objet, on déclare une référence sur la classe de cet objet :

```
Cercle c;
```

- Pour créer un objet, on instancie une classe en appliquant l'opérateur `new` sur un de ses constructeurs. Une nouvelle instance de cette classe est alors allouée en mémoire :

```
c = new Cercle(5);
```

- Toute classe possède un constructeur par défaut, implicite. Il peut être redéfini. Une classe peut avoir plusieurs constructeurs qui diffèrent par le nombre et la nature de leurs paramètres (équivalent à une surcharge du constructeur).

Création d'un objet

```
class Cercle() {
   double x, y, r;

// Un constructeur : même nom que la classe
   public Cercle(double x, double y, double r) {
       this.x = x; this.y = y; this.r = r;
   }

   public Cercle(Cercle c) {
       x = c.x; y=c.y; r=c.r;
   }

   public Cercle() {
       this.x = 0.0; this.y = 0.0; this.r = 0.0;
   }
}
```

Destruction d'un objet

- La destruction des objets est prise en charge par le ramasse-miettes. Il n'y a pas de `free()` à faire comme en langage C.
- Le ramasse-miettes détruit les objets pour lesquels il n'existe plus de référence.
- Les destructions sont asynchrones (le GC est géré dans un *thread* de basse priorité).
- Aucune garantie n'est apportée quant à la destruction effective d'un objet, c'est à dire la libération de la mémoire.

Destruction d'un objet

```
public class Cercle {
   ...
   void finalize() {
       System.out.println("Je suis libéré par le GC");
   }
}
...
   Cercle c1;

   c1 = new Cercle();

...
   c1=null;   // L'instance ne possède plus de référence. Elle n'est plus
              // accessible. A tout moment le GC peut détruire l'objet.
              // Libérer la mémoire réservée = mettre la référence à NULL
```

Structure des classes

- Une classe est un ensemble d'attributs et de méthodes (fonctions) : on les appelle les *membres*.

- Les membres sont accessibles via une instance de la classe ou via la **classe pour les membres statiques** :

```
c.r = 3;            // On accède à l'attribut 'r' de l'instance 'c'
a = c.area();       // On invoque la méthode 'area' de l'instance 'c'
pi = Math.PI;       // On accède à l'attribut statique 'PI' de la classe
    'Math'
b = Math.sqrt(2.0);// On invoque la méthode statique 'sqrt' de la classe
    'Math'
```

- Les méthodes sont définies directement au sein de la classe.

Structure des classes

- Les variables `static` sont communes à toutes les instances de la classe (équivalent à une variable partagée par différentes instances d'une même classe).

- **Il n'est pas nécessaire d'instancier une classe pour accéder à l'un de ses membres statiques.**

Structure des classes

```
public class Cercle {
   public static int count = 0;
   public static final double PI = 3.14; // final pour éviter Cercle.PI = 4;
   public double x, y, r;

   public Cercle(double r) {
       this.r = r;
       count++;
   } // Constructeur
}

Cercle c1 = new Cercle(10);
Cercle c2 = new Cercle(20);

n = Cercle.count;                  // n = 2
```


L'héritage

- Une classe ne peut hériter (`extends`) que d'une seule classe.
- Les classes dérivent par défaut de `java.lang.Object`.
- L'opérateur `instanceOf` permet de déterminer la classe d'une instance.

Structure des classes

```
public class Ellipse {
   public double r1, r2;
   public Ellipse(double r1, double r2) { this.r1 = r1; this.r2 = r2;)
   public double area{...}
}

final class Cercle extends Ellipse {
   public Cercle(double r) {super(r, r);}
   public double getRadius() {return r1;}
}

Ellipse e = new Ellipse(2.0, 4.0);
Cercle c = new Cercle(2.0);
System.out.println("Aire de e:" + e.area() + ", Aire de c:" + c.area());

System.out.println((e instanceOf Cercle));  // false
System.out.println((e instanceOf Ellipse)); // true
System.out.println((c instanceOf Cercle));  // true
System.out.println((c instanceOf Ellipse)); // true (car Cercle dérive de
   Ellipse)
```


Structure des classes

- Une classe peut définir des variables portant le même nom que celles de ses classes ancêtres.
- Une classe peut accéder aux attributs (variables) redéfinis de sa classe mère en utilisant `super` (super classe).
- Une classe peut accéder aux méthodes redéfinies de sa classe mère en utilisant `super`.

Structure des classes

```
class A {
   int x;
   void m() {...}
}
class B extends A{
   int x;
   void m() {...}
}
class C extends B {
   int x, a;
   void m() {...}

   void test() {
       a = super.x;      // a reçoit la valeur de la variable x de la classe B
       a = super.super.x; // Syntax error : héritage simple

       super.m();        // Appel à la méthode m de la classe B
       super.super.m(); // Syntax error : héritage simple
   }
}
```

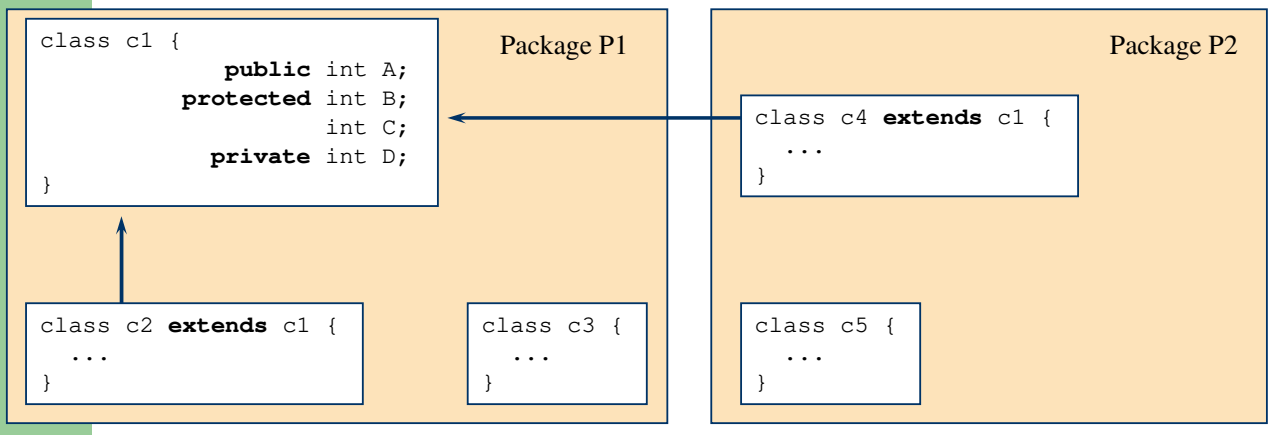

Visibilité

- Les classes, attributs et méthodes peuvent être visibles ou non à l'intérieur d'autres *packages* ou d'autres classes. :
  - Pas de mot-clé : visible par tout le *package* .
  - `private` visible par la classe uniquement.
  - `protected` : visible par tout le *package* et par les sous-classes.
  - `public` : visible par tout le monde.

Visibilité

- `int a` : lié à l'instance, visible par tout le *package*.
- `static int b` : lié à la classe, visible par tout le *package*.
- `private int c` : lié à l'instance, visible par la classe uniquement.
- `private static int d` : lié à la classe, visible par la classe uniquement.
- `protected int e` : lié à l'instance, visible par tout le *package* et par les sous-classes.
- `protected static int f` : lié à la classe, visible par tout le *package* et par les sous-classes.
- `public int a` : lié à l'instance, visible par tout le monde.
- `public static int g` : lié à la classe, visible par tout le monde.

Visibilité : un exemple



	A	B	C	D
Accessible par c2	o	o	o	-
Accessible par c3	o	o	o	-
Accessible par c4	o	o	-	-
Accessible par c5	o	-	-	-

Les classes abstraites

- Une classe abstraite est une classe ayant au moins une méthode abstraite.
- Une méthode abstraite ne possède pas de définition.
- Une classe abstraite ne peut pas être instanciée (`new`).
- Une classe dérivée d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraites est elle-même abstraite.

Les classes abstraites

```
class abstract Shape {
   public abstract double perimeter();
}

class Cercle extends Shape {
   ...
   public double perimeter() { return 2 * Math.PI * r ; }
}

class Rectangle extends Shape {
   ...
   public double perimeter() { return 2 * (height + width); }
}
```

Les interfaces

- **Une interface correspond à une classe où toutes les méthodes sont abstraites. C'est donc un contrat (ensemble de méthodes ici) devant être respecté par quiconque l'implémente.**

- Une classe peut implémenter (`implements`) une ou plusieurs interfaces tout en héritant (`extends`) que d'une seule classe (héritage simple).

- Une interface peut hériter (`extends`) de plusieurs interfaces.

Les classes abstraites

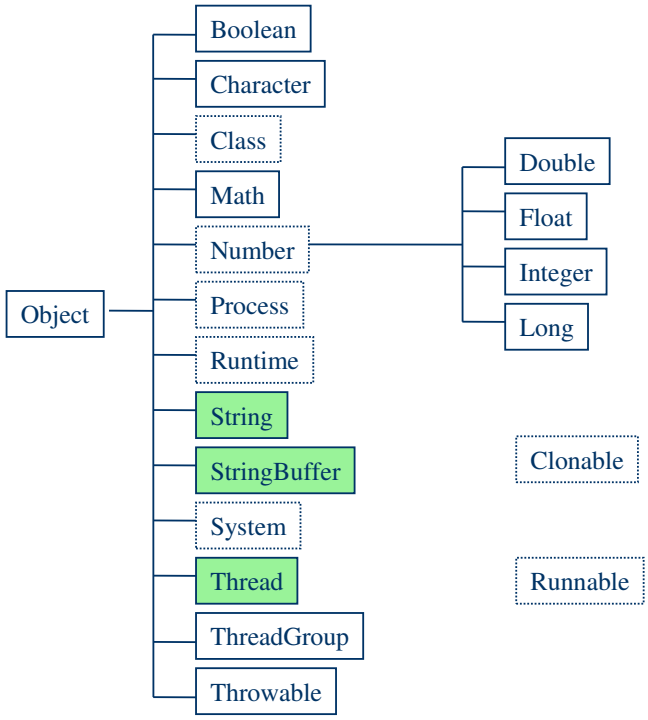
```
abstract class Shape {
   public abstract double perimeter();
}

interface Drawable {
   public void draw();
}

class Cercle extends Shape implements Drawable {
   public double perimeter() { return 2 * Math.PI * r ; }
   public void draw() {...}
}

class Rectangle extends Shape implements Drawable {
   ...
   public double perimeter() { return 2 * (height + width); }
   public void draw() {...}
}
```

PARTIE 4
LES API DE JAVA

Introduction

- Nous allons voir les API de base de Java et décrire quelques objets utiles pour les travaux pratiques.
- Il ne s'agit pas là de voir toutes les API.
- Il sera toujours possible de lire la documentation de qualité sur les API Java…

java.lang.*



java.lang.String

- La classe `String` gère des chaînes de caractères (`char`).
- Une `String` n'est pas modifiable.
- Toute modification entraîne la création d'une nouvelle `String`.
- Les valeur littérales (`"abc"`) sont transformées en `String`.
- L'opérateur + permet la concaténation de 2 `String`.

java.lang.String

```
String s = "coucou";                          // s = "coucou"
int lg = s.length();                          // lg = 6
String s = "Java" + "Soft";                   // s = "JavaSoft"


char[] data = {'J', 'a', 'v', 'a'};
String name = new String(data);

String s = String.valueOf(2 * 3.14159); // s = "6.28318"
String s = String.valueOf(new Date());  // s = "Sat Jan 18 12:10:36 2007"
int i = Integer.valueOf("123");         // i = 123

String s = "java";


if (s.equals("java") {
    ...
}
```


java.lang.StringBuffer

- La classe `StringBuffer` gère des chaînes de caractères modifiables (`setCharAt()`, `append()`, `insert()`).
- La méthode `toString()` convertit une `StringBuffer` en `String` :

```
StringBuffer sb = "abc";    // Error: can't convert String to StringBuffer
StringBuffer sb = new StringBuffer("abc");

sb.setCharAt(1, 'B');       // sb= "aBc"
sb.insert(1, "1234");       // sb = "a1234Bc"
sb.append("defg");          // sb = "a1234Bcdefg"

String s = sb.toString();   // s = "a1234Bcdefg"
sb.append("hij");           // sb = "a1234Bcdefghij" s = "a1234Bcdefg"
```

java.lang.Thread

- Cette classe permet de déléguer le traitement d'un objet par un *thread* (équivalent à un processus indépendant).

- Deux possibilités : soit hériter de la classe `Thread` ou soit implémenter l'interface `Runnable` :

```
class C1 extends Thread
{
    public C1() {
          this.start();
    }
    public void run() {
          …
    }
}
class C2 implements Runnable
{
    public C2() {Thread t = new Thread(this); t.start(); }
    public void run() {...}
}
```


java.lang.Thread

- Méthodes courantes :
  - `void start()` : lancement du *thread*, appelle `run()`.
  - `void stop()` : arrêt du *thread* et destruction.
  - `void suspend()` : mise en sommeil du *thread*.
  - `void resume()` : réveil du *thread*.
  - `static void sleep()` : mise en sommeil durant un temps fini en ms.

java.lang.Thread

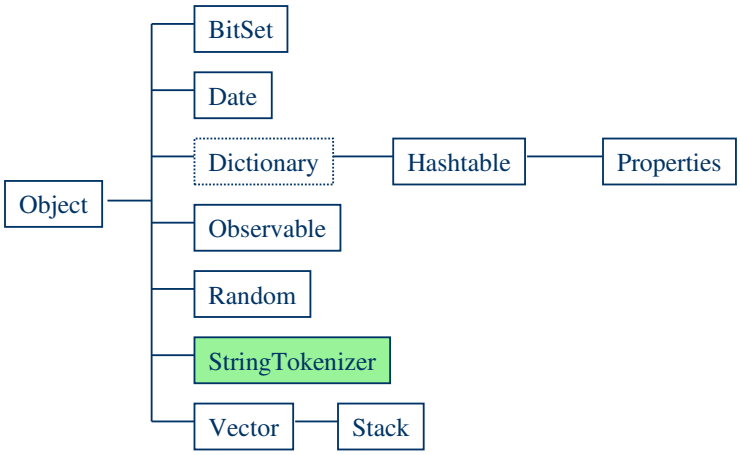
- Le mot réservé `synchronized` permet de synchroniser l'accès à une partie de code ou à une méthode.

- Dans une partie `synchronized` :
  - `wait()` : le *thread* attend l'accès à la ressource (voir P()).
  - `notify()` : réveille un *thread* en attente d'accès à la ressource (voir V()).
  - `notifyAll()` : réveille tous les *threads* en attente.

java.lang.Thread

```
class Semaphore {
    private int counter;

    public Semaphore() {
        counter = 1;
    }

    public Semaphore(int i) {
        if(counter == 0 || counter == 1)
            counter = i;
    }

    public synchronized void V() {
         if (counter == 0) {
             this.notify();
         }
         counter++;
     }

     public synchronized void P() throws InterruptedException {
         while (counter == 0) {
             this.wait();
         }
         counter--;
     }
}
```

java.util.*

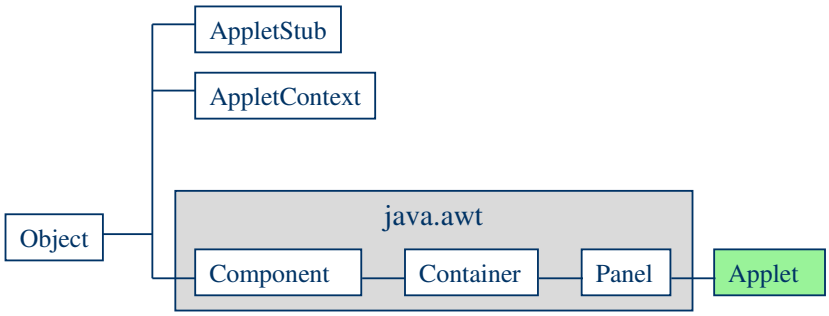


java.util.StringTokenizer

- Cette classe permet de découper une `String` selon des séparateurs de champ :

```
String str = "avion, bateau ; train  ";

StringTokenizer st = new StringTokenizer(str, ";, ");

System.out.println(st.nextToken()); // --> avion
System.out.println(st.nextToken()); // --> bateau
System.out.println(st.nextToken()); // --> train
```

java.applet.*



java.applet.Applet

- Une applet est une classe Java compilée héritant de `java.applet.Applet`.
- Elle est exportée par un serveur Web dans une page HTML par les balises HTML `<APPLET>` et `</APPLET>`.
- Elle est téléchargée puis exécutée par le navigateur (Firefox).
- Elle est soumise au *Security Manager* du navigateur :
  - Pas d'accès en lecture ni en écriture sur le disque du navigateur.
  - Connexion réseau uniquement vers le serveur d'origine.
  - Pas de lancement de processus.

java.applet.Applet

- Structure d'une *applet* :

```
public class MyApplet extends java.applet.Applet
{
    public void init() {...}
    public void start() {...}
    public void paint(java.awt.graphics g) {...}
    public void stop() {...}
    public void destroy() {...}
}
```

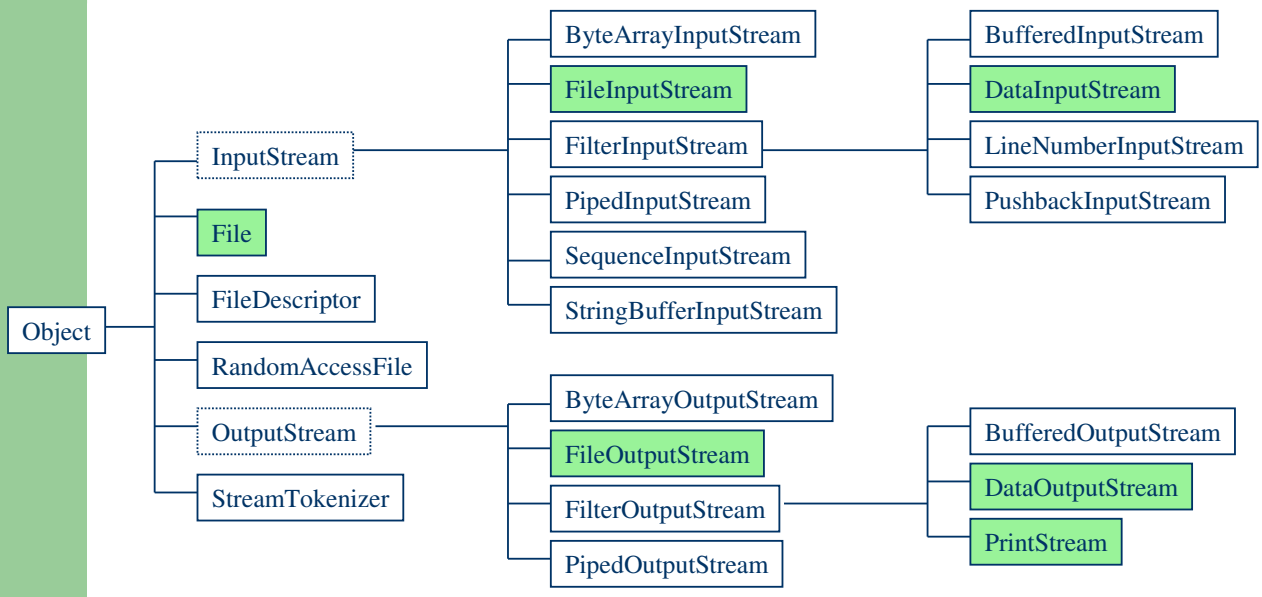
- Analogie avec une *MIDlet*…

java.applet.Applet

- Déclaration de l'*applet* :

```
<HTML>
  <BODY>
    <APPLET code="MyApplet.class"
            codebase="http://www.enseirb.fr/~dupont/applets/"
            width=300 height=200>
   <PARAM name="message" value="Hello World">
   </APPLET>
  </BODY>
</HTML>
```

java.io.*



java.io.File

- Cette classe fournit un accès aux fichiers et aux répertoires :

```
File f = new File("/etc/passwd");  // ouverture de /etc/passwd sur UNIX
System.out.println(f.exists());    // --> true
System.out.println(f.canRead());   // --> true
System.out.println(f.canWrite());  // --> false
System.out.println(f.getLength()); // --> 11345

File d = new File("/etc/");
System.out.println(d.isDirectory()); // --> true

String[] files = d.list();
for(int i=0; i < files.length; i++)
   System.out.println(files[i]);
```

java.io.File(Input|Output)Stream

- Ces classes permettent d'accéder en lecture et en écriture à un fichier :

```
FileInputStream fis = new FileInputStream("source.txt");
byte[] data = new byte[fis.available()];
fis.read(data);
fis.close();

FileOutputStream fos = new FileOutputStream("cible.txt");
fos.write(data);
fos.close(); // copie d'un fichier source dans un fichier destination
```


java.io.Data(Input|Output)Stream

- Ces classes permettent de lire et d'écrire des types primitifs et des lignes sur des flux d'octets non structurés (*stream*) :

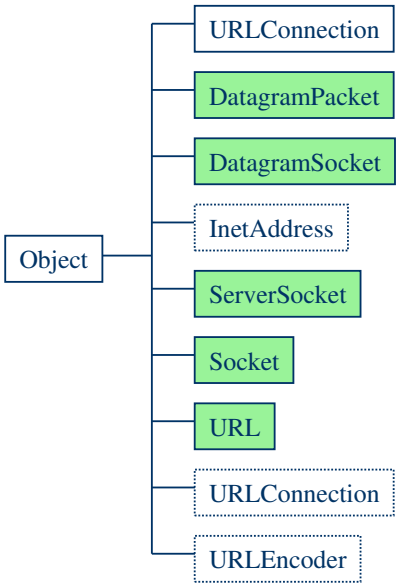
```
FileInputStream fis = new FileInputStream("source.txt");

DataInputStream dis = new DataInputStream(fis);

int i    = dis.readInt();
double d = dis.readDouble();
String s = dis.readLine();

FileOutputStream fos = new FileOutputStream("cible.txt");

DataOutputStream dos = new DataOutputStream(fos);

dos.writeInt(123);
dos.writeDouble(123.456);
dos.writeChars("Une chaine");
```

java.io.PrintStream

- Cette classe permet de manipuler un `OutputStream` au travers des méthode `print()` et `println()` :

```
PrintStream ps = new PrintStream(new FileOutputStream("cible.txt"));

ps.println("Une ligne");  // avec saut de ligne
ps.println(123);
ps.print("Une autre ");   // sans saut de ligne
ps.print("ligne");
ps.flush();
ps.close();
```

java.net.*



java.net.Socket

- Cette classe implémente une *socket* TCP côté client :

```
String serveur = "www.urec.fr";
int port = 80;

Socket s = new Socket(serveur, port);

PrintStream ps = new PrintStream(s.getOutputStream());

ps.println("GET / HTTP/1.0\n\n");

DataInputStream dis = new DataInputStream(s.getInputStream());

String line;

while((line = dis.readLine()) != null)
   System.out.println(line);
```


java.net.ServerSocket

- Cette classe implémente une *socket* TCP côté serveur :

```
int port_d_ecoute = 1234;

ServerSocket serveur = new ServerSocket(port_d_ecoute);

while(true){
   Socket socket_de_travail = serveur.accept();

   new ClasseQuiFaitLeTraitement(socket_travail);
}
```

java.net.DatagramSocket

- Cette classe implémente une *socket* UDP.

- Exemple de client UDP :

```
Byte data[] = "un message".getBytes();

InetAddress addr = InetAddress.getByName("www.enseirb.fr");

DatagramPacket packet = new DatagramPacket(data, data.length, addr, 1234);

DatagramSocket ds = new DatagramSocket();

ds.send(packet);

ds.close();
```


java.net.DatagramSocket

- Exemple de serveur UDP :

```
DatagramSocket ds = new DatagramSocket(1234);

while(true){
   DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);

   s.receive(packet);

   System.out.println("Message: " + packet.getData());
}
```

java.net.URL

- Création d'un flux (*stream*) à partir d'une URL (*Uniform Resource Locator*) :

```
URL url = new URL("http://www.enseirb.fr/index.html");

DataInputStream dis = new DataInputStream(url.openStream());
```


PARTIE 5
L'API AWT

Généralités

- L'API AWT (*Abstract Window Toolkit*) permet de créer simplement des interfaces graphiques portables (plateforme indépendant).

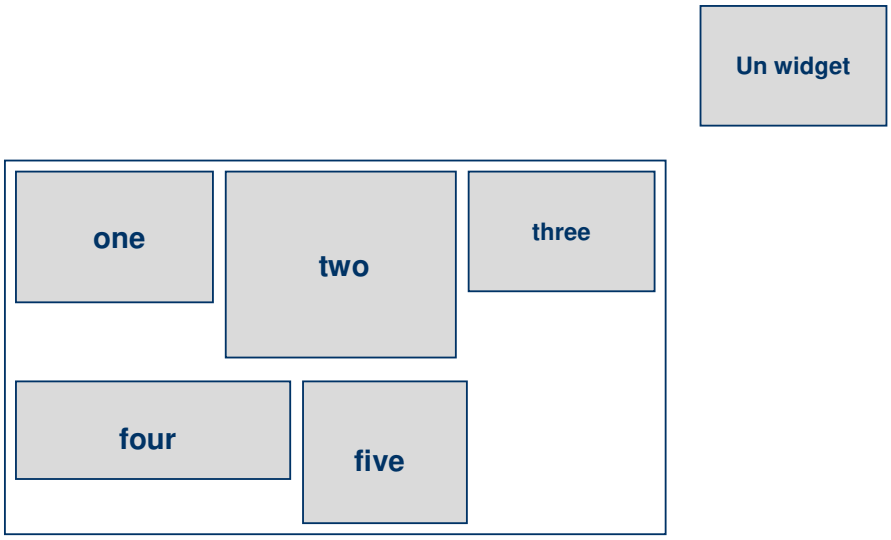
- AWT comporte :
  - des *widgets* (*WInDow GadGETs*) : bouton, *textfield*...
  - des *LayoutManager* (gestionnaire de mise en page des *widgets*) .
  - Un mécanisme de gestion d'événements...

- Les classes JFC (*Java Foundation Class*) ou *swings* permettent aussi de créer des interfaces graphiques…

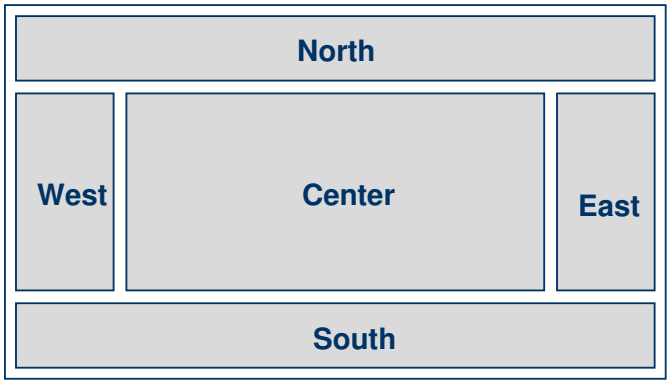
Généralités

- Les *widgets* :
  - java.awt.Button
  - java.awt.Canvas
  - java.awt.Checkbox
  - java.awt.Choice
  - java.awt.Label
  - java.awt.List
  - java.awt.MenuBar
  - java.awt.MenuItem
  - java.awt.TextArea
  - java.awt.TextField

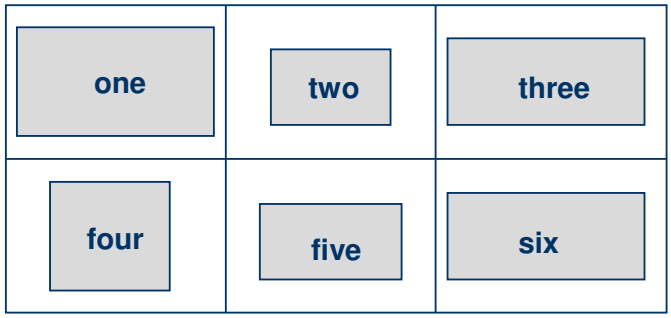
java.awt.FlowLayout



java.awt.BorderLayout

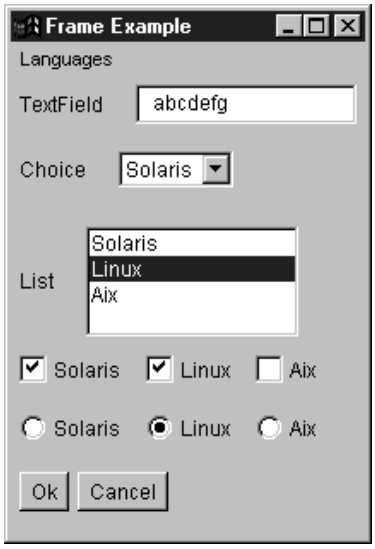


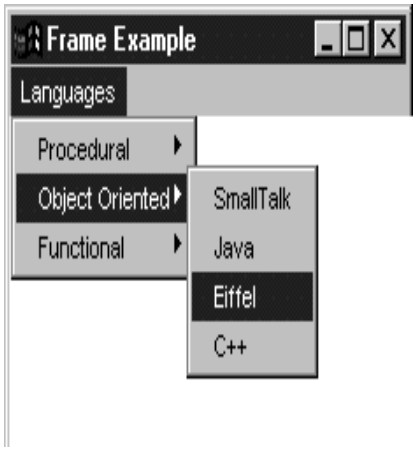
java.awt.GridLayout



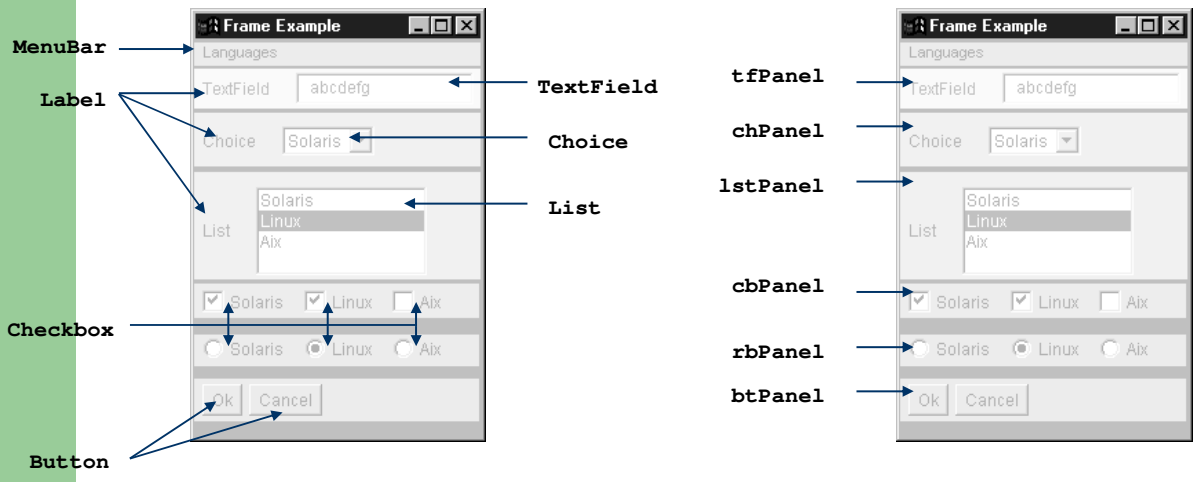
- Les *layouts* d'AWT sont rustiques. Ils en existent d'autres qui permettent d'avoir un *layout* WYSIWYG.

Exemple





Exemple



Exemple

```
// Choice Panel
Panel chPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

label = new Label("Choice");
chPanel.add(label);

choice = new Choice();
choice.addItem("Solaris");
choice.addItem("Linux");
choice.addItem("Aix");

chPanel.add(choice);
```

Exemple

```
// Button Panel
Panel btPanel = new Panel(new FlowLayout(FlowLayout.LEFT));

okButton = new Button("Ok");
okButton.addActionListener(new OkButtonListener());
btPanel.add(okButton);

cancelButton = new Button("Cancel");
cancelButton.addActionListener(new CancelButtonListener());
btPanel.add(cancelButton);
```

- Un événement est généré quand on cliquera sur un bouton (événement asynchrone). Il faut alors installer (`addActionListener()`) la routine de traitement qui est appelée *listener* (`OkButtonListener()` ou `CancelButtonListener()`).

Gestion des événements

- La gestion des événements est basée sur des mécanismes semblables à ceux utilisés par l'environnement graphique X11/Xorg.

- Il faut installer le *listener* puis écrire le code source du *listener*. La méthode exécutée dans le *listener* s'appelle ici `actionPerformed()` :

```
okButton.addActionListener(new OkButtonListener()); // installation
...
public class OkButtonListener implements ActionListener { // code listener

   public void actionPerformed(ActionEvent evt) {
      System.out.println("textField = " + textField.getText());
      System.out.println("choice = "    + choice.getSelectedItem());
      ...
   }
}
```

Gestion des événements

- La gestion des *listeners* sous Java ME est comparable. On installe le *listener* avec la méthode `setInputListener()`. La méthode exécutée dans le *listener* s'appelle ici `valueChanged()` :

```
public class TstBpListen extends MIDlet implements PinListener {
   private final int BP1 = 1;
   final boolean PRESSED = false;
   GPIOPin bp1;

   public void startApp()
       try {
              bp1 = (GPIOPin) DeviceManager.open(new GPIOPinConfig(
                     0, BP1, ...,  false));
              bp1.setInputListener(this);
       } catch (Exception ex) {
              ex.printStackTrace();
       }
   }
```


Gestion des événements

```
   public void valueChanged(PinEvent event) {
       try {
              if(bp1.getValue() == PRESSED)
                     System.out.println("BP1 PRESSED");
       } catch (IOException ex) {
              ex.printStackTrace();
       }
   }
}
```

PARTIE 6
CONCLUSION

Conclusion

- Java est plus qu'un langage de programmation orienté objet.
- Java est orienté réseau.
- Java a bénéficié de toutes les avancées de l'informatique technique distribuée (*applet*, multimédia, *thread*, accès à une base de données, objets distribués, objets sérialisés…).
- Java sera mis en œuvre du point de vue pratique sur objet connecté (Java ME)…

BIBLIOGRAPHIE

Bibliographie

- The Java Language Specification. J. Gosling, B. Joy, G. Steele. http://docs.oracle.com/javase/specs
- The Java Programming Language. K. Arnold, J. Gosling.
- Java 2 plate-forme. L. Lemay, R. Cadenhead. Editions CampusPress.
- Mieux programmer en Java. P. Haggar. Editions Eyrolles.
- Java embarqué. Y. Bossu. C. Nicolas, A. Proust, J.B. Blanchet. Editions Eyrolles.
- Java client/serveur. C. Nicolas. Editions Eyrolles.