Année 2008-2009

esil

JAVA
Première approche

Nicolas Baudru
mél : nicolas.baudru@esil.univmed.fr
page web : nicolas.baudru.perso.esil.univmed.fr

Qu'est ce que Java ?

C'est le nom d'une technologie mise au point par Sun Microsystem constituée :

- du langage Java qui est un langage de programmation
  - orienté objet et fortement typé
  - utilisant des mécanismes d'héritages simples, d'interfaces et de polymorphismes
- du JRE (Java Runtime Environment) regroupant
  - la JVM (Java Virtual Machine) qui interprète le code Java et le convertit en code natif
  - les APIs (Application Program Interface) : ensembles de bibliothèques standards
- du JDK (Java Development Kit) regroupant
  - le compilateur Java
  - le débogueur Java

logo Java

Licence GNU GPL

Le Javascript (développé par Netscape, Inc.) n'a rien à voir avec le Java.

Pourquoi apprendre Java ?

De par son évolutivité, son efficacité, la portabilité de sa plate-forme et sa sécurité, la technologie Java est devenue la solution idéale pour l'informatique de réseau.

Java équipe plus de 4,5 milliards de périphériques, notamment :

- 800 millions de PC
- 1,5 milliards de téléphones portables et autres périphériques de poche (source : Ovum)
- 2,2 milliards de cartes à puce
- Des décodeurs, des imprimantes, des webcams, des jeux, des systèmes de navigation automobile, des terminaux de loterie, des appareils médicaux, des bornes de paiement de parking, etc.

www.java.com

Pourquoi les développeurs de logiciels ont choisi Java ?

Sa flexibilité, son efficacité et sa portabilité font de Java un outil indispensable, qui permet aux développeurs :

- d'écrire des logiciels sur une plate-forme et de les exécuter sur pratiquement toutes les autres plates-formes,
- de créer des programmes à exécuter dans un navigateur Web et dans des services Web : ces programmes sont appelés des applets
- de développer des applications serveur pour des forums, des magasins et des sondages en ligne, pour le traitement de formulaires HTML, et plus encore,
- de combiner des applications ou des services basés sur Java pour créer des applications ou services très personnalisés,
- d'écrire des applications puissantes et efficaces pour les téléphones portables, les processeurs à distance et tous les autres types de périphériques dotés d'un signal numérique.

www.java.com

Avantages du Java sur le C++

- Peu de concepts à assimiler :
  - Il n'y a que des classes (pas de struct, enum, union)
  - Il n'y a que des références (pas de pointeurs)
- Pas besoin de gérer la mémoire grâce au garbage collector
- Java est indépendant de la plate-forme utilisée
- Java gère nativement les threads

Historique de Java

Java 1.3 (kestrel)

Java 1.4 (merlin)

Java 1.5 (tiger)

Java 1.6 (mustang)

Java 1.02	lent	250 classes	applets
Java 1.1	très répendu	500 classes	interfaces graphiques, JDBC, RMI
Java 2 (J2SE 1.2 à 1.4)	beaucoup plus rapide, puissant	2300 classes	applications web, applications mobiles API swing, Image I/O, parseur XML, JCE
Java 5.0 (J2SE 1.5)	plus puissant, plus facile	3500 classes	énumérations, VarArgs

Depuis 2006, il existe la version Java SE 6. Une version Java SE 7 est prévue pour 2009.

Comment fonctionne Java ?

1 Créez un document source en utilisant un protocole établi : le Java

2 Passez votre document dans un compilateur de code source Java. Tant que votre fichier contiendra des erreurs, le compilateur refusera de compiler.

3 Une fois que le compilateur est sûr que tout fonctionnera correctement, il crée un nouveau document codé en bytecode Java. C'est ce document qui va être exécuté par la JVM.

4 La JVM est un logiciel installé sur de nombreuses plate-formes. Elle traduit votre bytecode en quelque chose de compréhensible par la plate-forme sous-jacente. Votre bytecode est donc indépendant de la plate-forme utilisée !

1 Créez un fichier source et l'enregistrez sous : Chien.java

```
public class Chien {
  void aboyer(){
    int x = 0;
  }
}
```

2 Compilez votre fichier source : javac Chien.java
Vous devriez obtenir un fichier bytecode nommé : Chien.class

```
Method Chien()
0 aload_0
Methode Aboyer()
0 new #2 <int>
3 dup
```

3 Lancez votre programme dans la JVM : java Chien.class

```
int taille = 27;

String nom = "Fido";

Chien monChien = new Chien(nom, taille);

int x = taille - 5;

if (x < 15) monChien.aboyer(8);

int[] listeNombres = {1, 3, 4, 6};

System.out.print("Bonjour");

System.out.print("nom␣du␣chien␣:␣" + nom);
```

Un fichier source Java contient une classe.
Une classe contient des attributs et des méthodes.
Une méthode contient des intructions.

Exemple :

```
public class Chien {  //début de la classe "Chien"
  String nom = "Fido"; // attribut

  void aboyer(){ //début de la méthode "aboyer"

    int x = 0; //instruction 1
    x = x +1; //instruction 2

  } //fin de la méthode "aboyer"

} //fin de la classe "Chien"
```

Un fichier source (avec extension .java) contient une classe. La classe représente en général une portion de votre programme seulement. Une classe est introduite par le mot clé class et est délimitée par deux accolades.

```
public class Chien {  //début de la classe "Chien"

  //ici se trouvent les définitions des différentes
  //méthodes de la classe Chien

} //fin de la classe "Chien"
```

- public signifie que “tout le monde peut accéder” à cette classe
- class signifie que nous sommes en train de définir une classe
- Chien est le nom de la classe

Méthodes

Une classe contient une ou plusieurs méthodes. Une méthode décrit un "comportement" de votre classe. Elle est aussi délimitée par deux accolades.

```
public class Chien {

  void aboyer(int nb_fois){ //début de la méthode

    //ici se trouve une suite d'instructions
    // de conditions et de boucles

  } //fin de la méthode "aboyer"
}
```

- void type de retour. Void signifie "pas de valeur de retour"
- public signifie que "tout le monde peut accéder" à cette méthode
- (int nb_fois) signifie que la méthode utilise un argument de type entier

Une méthode contient une liste d'instructions
(déclarations, affectations, appels de méthodes, ...)

```
public class Chien {

  void aboyer(int nb_fois){
    int x;                                  //déclaration
    x = nb_fois;                            //affectation

      System.out.print("x vaut 0");         //appel de méthodes


        System.out.println("ouaf!");        //appel de méthodes
        x = x - 1;                          //affectation


  }
}
```

Une méthode contient une liste d'instructions, des conditions

(if/else)

```
public class Chien {

  void aboyer(int nb_fois){
    int x;
    x = nb_fois;
    if ( x == 0 ) {  // instructions si x vaut 0
      System.out.print("x vaut 0");
    }  else {   //instructions si x ne vaut pas 0

        System.out.println("ouaf!");
        x = x - 1;

    } // fin else
  }
}
```

Une méthode contient une liste d'instructions, des conditions et des boucles (for, while et do-while)

```
public class Chien {

  void aboyer(int nb_fois){
    int x;
    x = nb_fois;
    if ( x == 0 ) {
      System.out.print("x vaut 0");
    }  else {
      while( x > 0 ){
        //tant que x est positif,
        //répeter les instructions suivantes
        System.out.println("ouaf!");
        x = x - 1;
      } //fin des instructions à répéter
    }
  }
}
```

Lorsque vous essayez d'exécuter votre bytecode (ex : java Chien.class), la JVM doit savoir où commencer l'exécution de votre bytecode. Pour cela il faut définir dans votre classe une méthode spéciale, appelée main, qui ressemble à

```
class Chien{
  ...
  public static void main(String[] args){
    // votre code va commencer par s'exécuter ici.
  }
  ...
}
```

La JVM exécute tout ce qui se trouve entre les accolades de la méthode main().

Toute application Java contient donc au moins une classe et au moins une méthode main().

Faut-il mettre une méthode main() dans chaque classe ?

Notre premier petit programme

```
public class Chien {

  void aboyer(int nb_fois){
    int x;
    x = nb_fois;
    if ( x == 0 ) {
      System.out.print("x vaut 0");
    }  else {
      while( x > 0 ){
        System.out.println("ouaf!");
        x = x - 1;
      }
    }
  }
  public static void main(String[] args){
    aboyer(3);
  }
}
```

OUAF ! OUAF ! OUAF !

Un autre exemple

```
public class Boucle {
  public static void main(String[] args){
    int x = 1;
    System.out.println("Début␣de␣la␣boucle");
    while( x < 4 ) {
      System.out.println ("Dans␣la␣boucle");
      System.out.println ("La␣valeur␣de␣x␣est␣:␣" + x);
      x = x + 1;
    }
    System.out.println("fin␣de␣la␣boucle");
  }
}
```

Quel est le résultat de ce programme ?

Encore un autre exemple

```
public class TestIf {
  public static void main(String[] args){
    int x = 1;
    if( x == 1 ) {
      System.out.println ("La␣valeur␣de␣x␣est␣1");
    }
    System.out.println("fin");
  }
}
```

Quel est le résultat de ce programme ?

Encore encore un autre exemple

```
public class TestIf {
  public static void main(String[] args){
    int x = 2;
    if( x == 1 ) {
      System.out.println ("La valeur de x est 1");
    }
    else{
       System.out.println ("x ne vaut pas 1");
    }
    System.out.println("fin");
  }
}
```

Quel est le résultat de ce programme ?

Quelques questions

Par quoi se termine toujours une instruction ?

Qu'est-ce qu'un bloc de données ?

Quelle est la différence entre "==" et "=" ?

Quelle est la différence entre System.out.print et System.out.println ?

Avons-nous le droit d'écrire le morceau de code suivant "à la C" ?

```
...
int x = 1;
while ( x ) {...}
...
```

Le programme du bistro d'en face...

esil

```
public class SongForABeer
{
  public static void main(String[] args){
    int nbBiere = 33;
    String mot = "bouteilles";

    while(nbBiere > 0){
      if( nbBiere == 1) { mot =  "bouteille"; }
      System.out.println(nbBiere + mot + "de biere");
      System.out.println("je la prends");
      System.out.println("je la bois");
      System.out.println("je la jette");
      System.out.print("Combien reste-t-il")
      System.out.println("de bouteilles de biere?");
    }
    System.out.println("Y-a plus de bouteilles !");
  }
}
```

Un programme générateur de phrases

esil

```
public class GenerateurDePhrase
{
  public static void main(String[] args){
    String[] listeUn = {"etudiant(e)","enseignant(e)"};
    String[] listeDeux = {"premiere","deuxieme"};
    String[] listeTrois = {"biomed","info","internet"};

    int lg1 = listeUn.length;
    int lg2 = listeDeux.length;
    int lg3 = listeTrois.length;

    int r1 = (int) (Math.random() * lg1);
    int r2 = (int) (Math.random() * lg2);
    int r3 = (int) (Math.random() * lg3);

    String phrase = listeUn[r1] + "␣en␣"
    + listeDeux[r2] + "␣année␣de␣" + listeTrois[r3];
    System.out.println("Je␣suis␣un(e)␣" + phrase);
  }
}
```

Qu'avons-nous vu jusqu'ici ?

esil

- écrire une classe simple avec une fonction main
- écrire des instructions, des conditions et des boucles en Java
- déclarer une variable entière, une variable chaîne de caractères
- afficher du texte et des variables
- générer un nombre aléatoire
- créer un tableau de chaînes de caractères
- concaténer des chaînes de caractères