



# Java

## Licence Professionnelle CISII, 2009-2010

---

### Cours 1 : Introduction à Java

A. Belaïd

[abelaid@loria.fr](mailto:abelaid@loria.fr)

Cours disponible sur le site :

<http://www.loria.fr/~abelaid> puis Teaching



# Fonctionnement

---

- 12 séances :
  - 40 h : Cours + TD
- Deux contrôles de connaissances (Cc1 + Cc2)
- Un projet commun (P)
  - Un rapport d'analyse détaillé : rappel du sujet, analyse : algorithmique, choix des structures de données, choix des classes
  - Le tout envoyé par email à [abelaid@loria.fr](mailto:abelaid@loria.fr) avec comme nom :
    - `LPCISII-Proj-nometudiant1- nometudiant2- nometudiant3.zip`
  - Une démonstration sur machine
- Un examen final (Ef)
- Note finale =  $(5 * Ef + 3 * P + Cc1 + Cc2) / 10$



# Le cours

---

## ■ Plan

- Introduction
- Objets et classes
- Héritage et polymorphisme
- Types génériques
- Types énumérés
- Paquets et qualifieurs
- Classes et méthodes abstraites

## ■ Plan (suite)

- Interfaces
- Containers
- Exceptions
- Entrées-sorties
- Threads
- Interfaces graphiques
- Applets



# Le cours

---

- Pour en savoir plus

- <http://java.sun.com>
- <http://java.sun.com/javase/6/docs/api>
- <http://java.sun.com/docs/books/tutorial/>



# Java

---

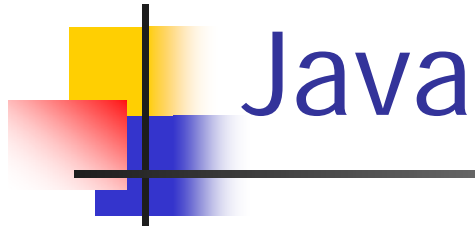
- Qu'est ce que c'est que Java ?
  - Inventé par SUN en 1990
  - Objectif : langage portable
    - grâce à l'exécution par une machine virtuelle JVM  
*« Compile once, run everywhere »*
  - Indépendant des plates-formes
    - *« Write once, debug everywhere »*
  - Simple, orienté objet, familier
    - Syntaxe très proche du langage C
    - Pas de gestion de la mémoire de la part du concepteur
    - Tout est objet sauf les types fondamentaux
    - Héritage simple : épuration par rapport à C++



# Java

---

- Qu'est ce que c'est que Java? (suite)
  - Interprété
    - Transformé en code intermédiaire
  - Robuste, sûr
    - Fortement typé, pas de pointeurs
    - Vérification au chargement des classes et durant leur exécution
  - Dynamique et distribué
    - Manipulation des objets distants et locaux
    - Classes chargées en fonction des besoins (le plus souvent par le réseau)
    - Permet le parallélisme de manière simple : facilités pour distribuer les traitements entre plusieurs machines



## ■ Différentes versions

- Java 1.02 : 250 classes, lent
- Java 1.1 : 500 classes : un peu plus rapide
- Java 2 : 2300 classes (différentes versions) : beaucoup plus rapide
- Java 5 : 3270 classes
- Java 6 : 203 packages et 3792 classes



## ■ Le kit de développement

- Dans chaque version, Sun propose un environnement complet pour le développement (**JDK**) et l'exécution d'applications basées sur Java, comprenant une machine virtuelle Java (JVM) ainsi qu'un ensemble de classes





# Java 2 SDK

---

## ■ Depuis 1 an

- Sun a changé la dénomination de ses différents Kit
- Avant, il n'existait que le JDK. Maintenant on parle du J2SDK qui comprend 3 éditions (à partir de la version 1.2)
  - le Kit de dev standard : J2SE
  - le kit enterprise edition : J2EE
  - le kit de dev micro edition : J2ME
  - le runtime ou la JVM a pour petit nom J2RE
- Ainsi, pour un kit de dev standard version 1.3, on parle du J2SE version 1.3
- Pour un kit de dev standard version 1.4, on parle du J2SE version 1.4, ou de "merlin" (c le petit nom de cette version).
- Pour simplifier, on désigne parfois un J2SE par JDK



# Java 2 SDK

---

## ■ Installation

- <http://java.sun.com> ou par FTP <ftp://java.sun.com/pub/>
- Il est disponible pour les plates-formes Windows32 (Windows 95, 98, Millenium, NT, 2000 et XP), Solaris (système UNIX de SUN) et Linux (RedHat)
- D'autres plates-formes supportent également Java 2 SDK, mais ce sont leurs « constructeurs » respectifs qui se chargent du support
- C'est en particulier le cas de MacOS X (Apple), de HP-UX (HP), d'IBM-Aix et de bien d'autres
- Toutes les grandes plates-formes matérielles et logicielles supportent Java à l'heure actuelle



# Java 2 SDK

---

## ■ Installation (suite)

- Installer le JDK dans le répertoire « C:\ »
- Dans « C:\j2sdk\bin », se trouvent les outils permettant de programmer, en particulier :
  - javac : le compilateur Java
  - java : l'interpréteur Java également appelé « machine virtuelle »
  - javadoc : l'outil de génération « automatique » de la documentation (format HTML) à partir de sources Java
  - plus d'autres outils que nous utiliserons peut-être ultérieurement

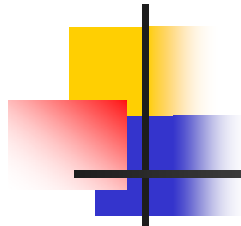


# Java 2 SDK

---

## ■ Utilisation

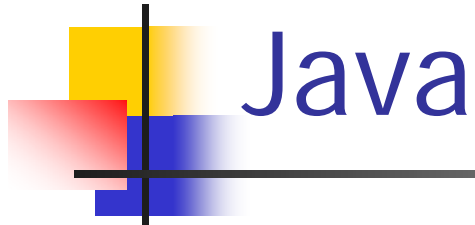
- Nécessite la configuration de variables d'environnement
  - « PATH », « CLASSPATH » et « JDK\_HOME »
  - A l'IUT, ces variables d'environnement sont déjà déclarées lorsque vous vous connectez sur un PC
  - En revanche, si vous installez le « jdk » chez vous, il est fort probable que vous ayez à faire cette déclaration manuellement
  - La déclaration peut se faire dans la fenêtre de commandes ('invite de commandes') en tapant :
    - `set JDK_HOME=C:\j2sdk`
    - `set CLASSPATH=C:\j2sdk\lib\tools.jar; C:\j2sdk\lib\dt.jar;.`
    - `set PATH=%PATH%;C:\j2sdk\bin`
  - Une autre solution est de définir ces variables de manière permanente dans le panneau de configuration/système/avancé/variables d'environnement



# IDE

---

- **Édition des programmes Java**
  - bloc note
  - (X)Emacs
  - Eclipse



## ■ Premier programme

- Fichier : Bonjour.java

```
public class Bonjour {  
    public static void main(String args[]) {  
        System.out.println("Bonjour tout le monde!");  
    }  
}
```

*Public pour que tout le monde puisse y accéder*

*Nom de la classe*

*Nom de la méthode*

*Afficher sur la sortie standard*

*La chaîne à afficher*

- Dans toute application il faut une classe publique qui contient une méthode **main()**
- main() est la première méthode exécutée



# Java, Javac

---

## ■ Premier programme

- Fichier : Bonjour.java

- Compilation

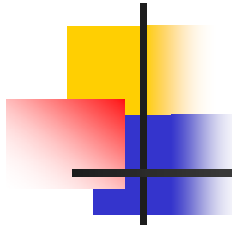
- javac Bonjour.java ➔ Bonjour.class

- Exécution

- java Bonjour

- Le nom de fichier doit être le même que celui de la classe

- ➔ un autre nom provoquera une erreur lors de la compilation



# Java

---

## ■ Compilation

- Un code source ne peut être exécuté directement par un ordinateur
- Il faut traduire ce code source dans un langage que l'ordinateur (le processeur de l'ordinateur) peut comprendre (langage *natif*)
- Un compilateur est un programme qui effectue cette traduction

## ■ En Java

- Le code source n'est pas traduit directement dans le langage de l'ordinateur
- Il est d'abord traduit dans un langage appelé « *bytecode* »,
  - langage d'une machine virtuelle (JVM ; *Java Virtual Machine*) définie par *Sun*
- Ce langage est indépendant de l'ordinateur qui va exécuter le programme



# Java

Compilation → ByteCode

Programme Java

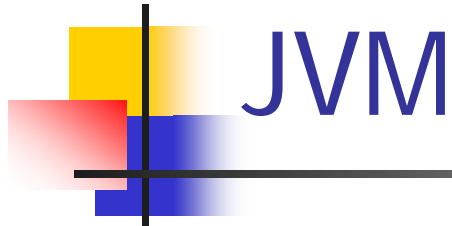
Programme source  
Bonjour.java

Compilateur : javac

Compilateur

Programme en *bytecode*,  
indépendant de  
l'ordinateur

Bytecode  
Bonjour.class

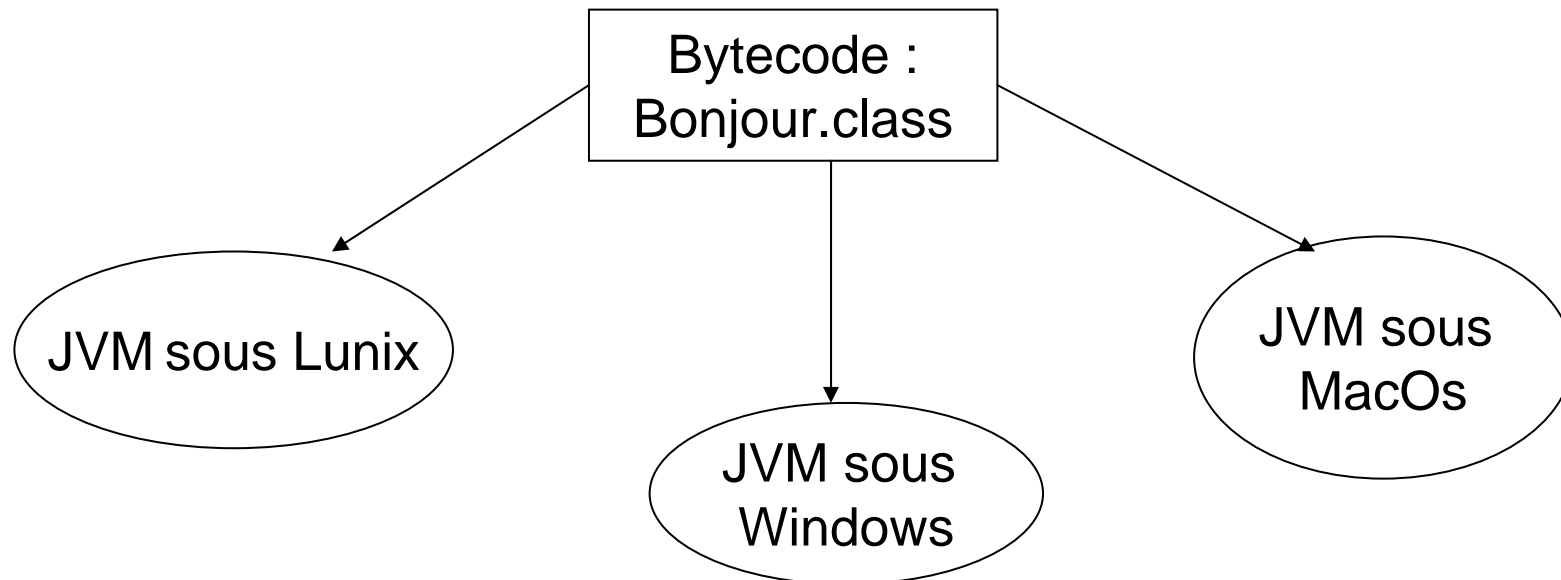


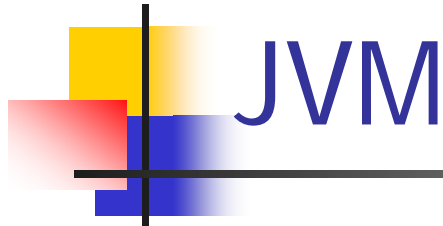
## ■ Java Virtual Machine

- Les systèmes qui veulent pouvoir exécuter un programme Java doivent fournir une JVM
- A l'heure actuelle, tous les systèmes ont une JVM (Linux, Windows, MacOS, ...)
- Il existe aussi depuis peu quelques JVM "en dur", sous forme de processeurs dont le langage natif est le *bytecode* ; elles sont rarement utilisées (en raison de la portabilité)

# JVM

- Le *bytecode* peut être exécuté par n'importe quelle JVM
  - Si un système possède une JVM, il peut exécuter tous les fichiers **.class** compilés sur n'importe quel autre système





## ■ Avantages de la JVM pour Internet

- Grâce à sa portabilité, le *bytecode* d'une classe peut être chargé depuis une machine distante du réseau, et exécuté par une JVM locale
- La JVM fait de nombreuses vérifications sur le *bytecode* avant son exécution pour s'assurer qu'il ne va effectuer aucune action dangereuse
- La JVM apporte donc
  - de la souplesse pour le chargement du code à exécuter
  - mais aussi de la sécurité pour l'exécution de ce code



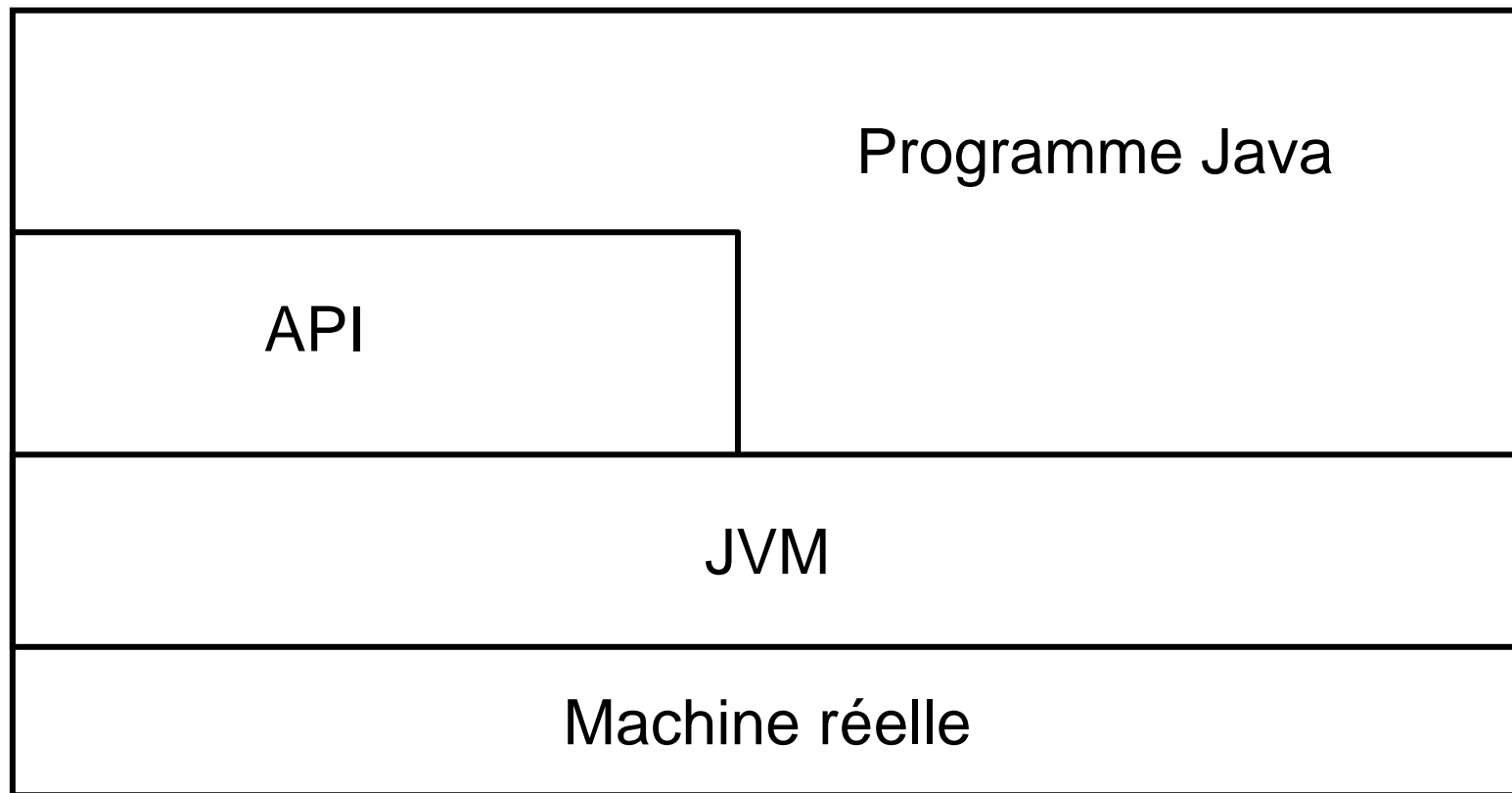
## ■ Avantages de la JVM pour Internet (suite)

- Les vérifications effectuées sur le bytecode et l'étape d'interprétation de ce bytecode (dans le langage natif du processeur)
  - ralentissent l'exécution des classes Java
- Les techniques « *Just In Time (JIT)* » ou « *Hotspot* » réduisent ce problème :
  - permettent de ne traduire qu'une seule fois en code natif et à la volée les instructions qui sont exécutées



# Plateforme Java

---





# Java en résumé

---

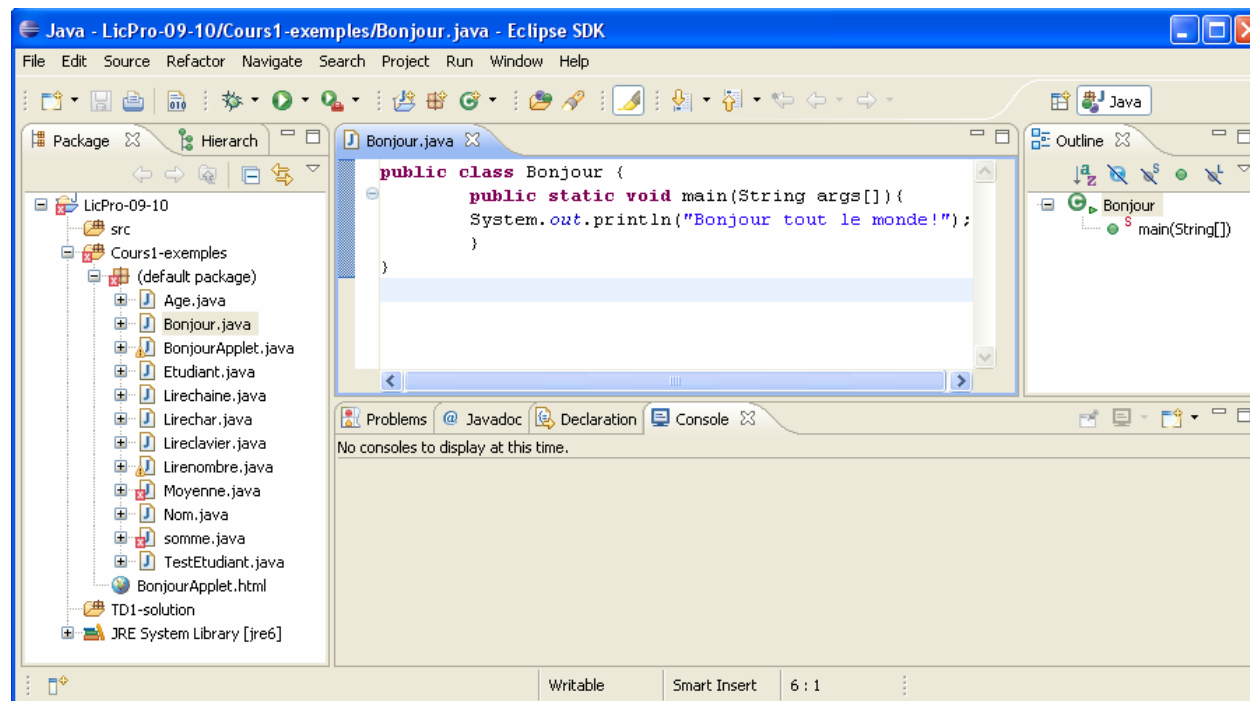
## ■ Votre environnement de développement

- SE : Linux ou Windows
- Éditeur de texte (emacs, word, eclipse...)
- Compilateur : javac
- Interpréteur de bytecode : java
- Aide en ligne
- Générateur automatique de documentation : javadoc
- Débogueur : jbd
- Interface de développement : Eclipse

# Java

## ■ Premier programme (sous eclipse)

1. Créer un projet : File >> New >> Java Project
2. Créer un répertoire de ressources (cours1-exemples) : File >> New >> Source Folder
3. Créer une classe : File >> New >> class







# Java

---

- **Deuxième programme : Age.java**

- L'utilisateur fournit son âge en argument de ligne de commande

```
public class Age {  
    public static void main(String args[]) {  
        int age;  
        age = Integer.parseInt(args[0]);  
        System.out.println("Vous avez " + age + " ans.");  
    }  
}
```

- Exécution : java Age 5
- Sortie : Vous avez 5 ans.



# Java

---

## ■ Explications

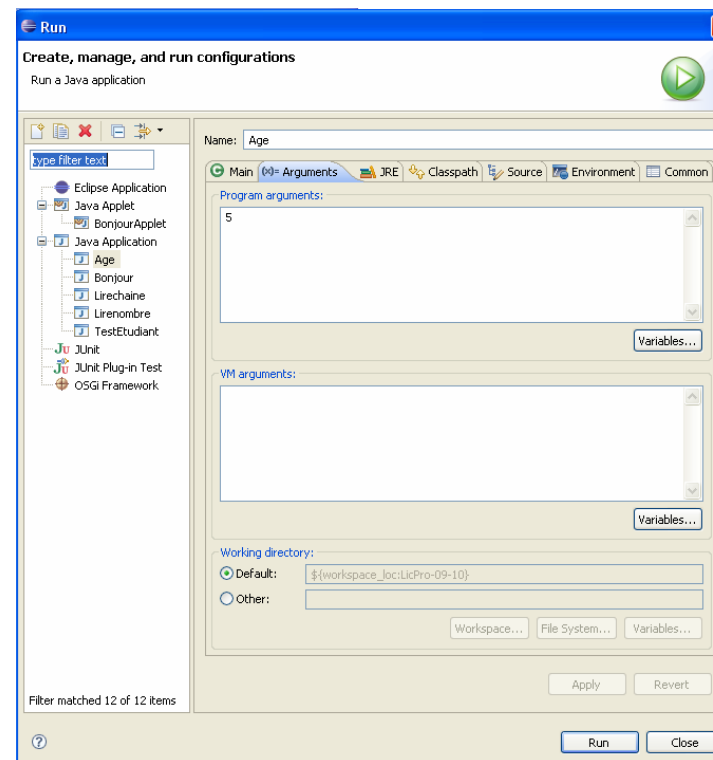
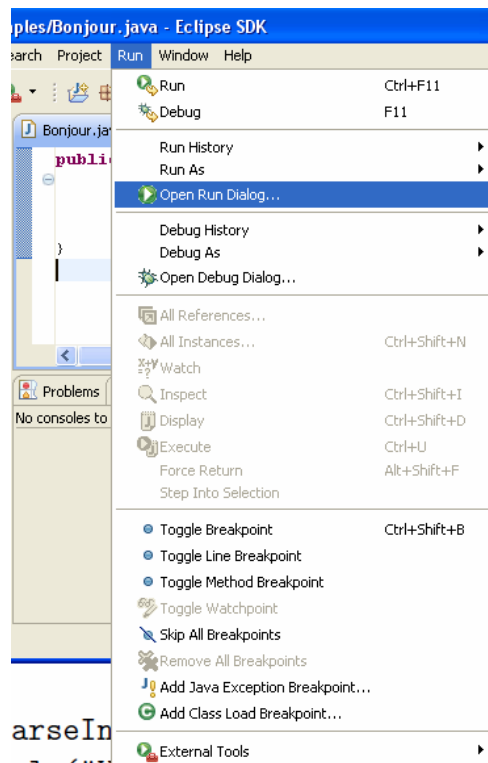
- **String** est une classe
- **String args[]** est un tableau de chaînes de caractères qui stocke les arguments fournis en ligne de commande
- **int** est un type primitif (fondamental)
- **Integer** est une classe d'objets de type int
- **parseInt()** est une méthode de la classe Integer qui convertit un String en int

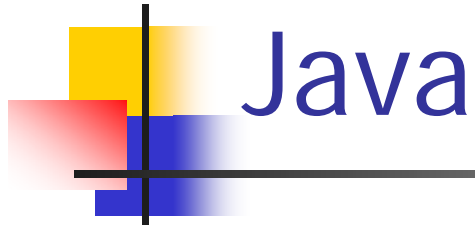
```
public class Age {  
    public static void main(String  
        args[]) {  
        int age;  
        age =  
            Integer.parseInt(args[0]);  
        System.out.println("Vous  
            avez " + age + " ans.");  
    }  
}
```

# Java

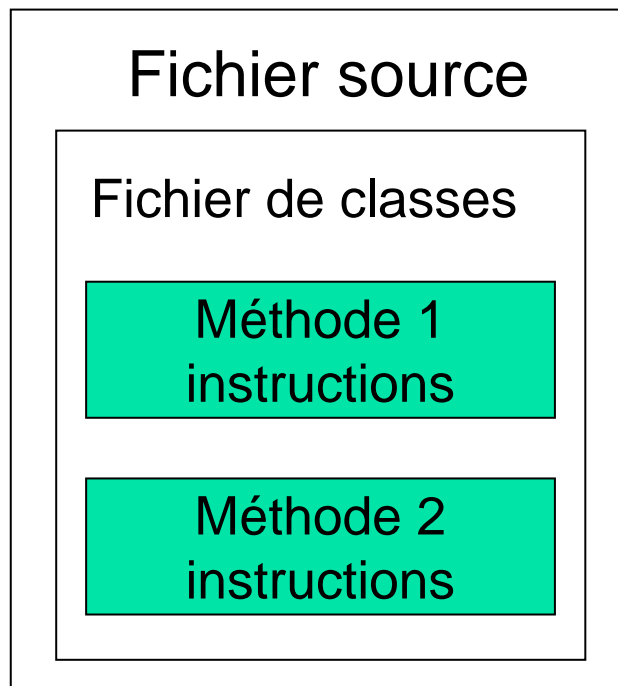
## ■ Deuxième programme

- Lecture des arguments sous Eclipse





## ■ Structure d'une application



- Placer une classe dans un fichier source
- Placer les méthodes dans une classe
- Placer les instructions dans les méthodes



# Java

## 2 classes et 1 fichier : Point.java

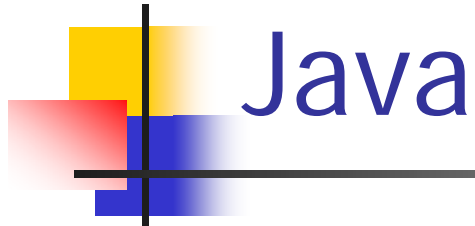
---

- Structure d'une application

```
/** Modélise un point de  
    coordonnées x, y */
```

```
public class Point {  
    private int x, y;  
    public Point(int x1, int y1) {  
        // constructeur  
        x = x1;  
        y = y1;  
    }  
  
    public double distance(Point p)  
    {  
        // une méthode  
        return Math.sqrt((x-p.x)*(x-p.x)  
            + (y-p.y)*(y-p.y));  
    }  
}
```

```
public static void main(String[]  
args) {  
    Point p1 = new Point(1, 2);  
    Point p2 = new Point(5, 1);  
    System.out.println("Distance : " +  
        p1.distance(p2));  
}
```



## ■ Compilation et exécution

### - Compilation

- javac Point.java fournit 1 fichier classe :
  - Point.class

### - Exécution

- java Point
  - On exécute Point qui lance en premier la méthode main():



# Java

## 2 classes et 2 fichiers : Point.java

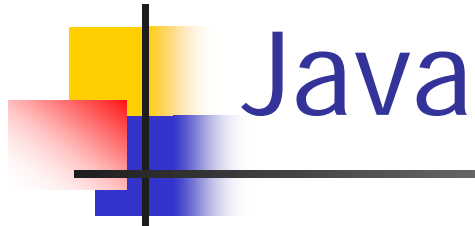
### ■ Fichier Point.java

```
/** Modélise un point de
    coordonnées x, y */
public class Point {
    private int x, y;
    public Point(int x1, int y1) {
        // constructeur
        x = x1;
        y = y1;
    }

    public double distance(Point p)
    {
        // une méthode
        return Math.sqrt((x-p.x)*(x-p.x)
            + (y-p.y)*(y-p.y));
    }
}
```

### Fichier TestPoint.java

```
class TestPoint {
    public static void main(String[]
        args) {
        Point p1 = new Point(1, 2);
        Point p2 = new Point(5, 1);
        System.out.println("Distance : "
            + p1.distance(p2));
    }
}
```



## ■ Compilation et exécution

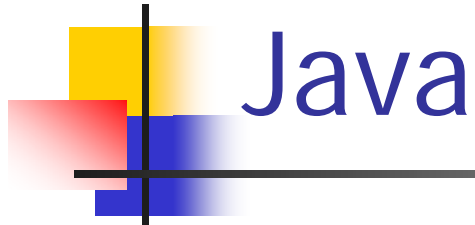
### - Compilation

- javac Point.java fournit 2 fichiers classes :
  - Point.class
  - TestPoint.class

### - Exécution

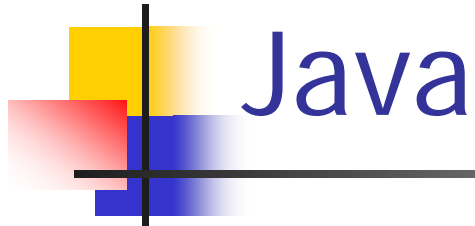
- java TestPoint
  - On exécute TestPoint qui lance en premier la méthode main()





## ■ Architecture générale d'un programme Java

- Programme source Java = ensemble de fichiers «**.java**»
- Chaque fichier «**.java**» contient une ou *plusieurs* définitions de classes
- Au plus une définition de classe **public** par fichier «**.java**» avec **nom du fichier = nom de la classe publique**
  - c'est plus simple, sinon, le compilateur fournit un autre fichier.class où fichier est le nom de la classe, et c'est cette classe qu'il faudra exécuter



## ■ Chargement dynamique des classes

- Durant l'exécution d'un code Java, les classes (leur *bytecode*) sont chargées dans la JVM au fur et à mesure des besoins
- Une classe peut être chargée
  - depuis la machine locale (le cas le plus fréquent)
  - depuis une autre machine, par le réseau, par tout autre moyen (base de données,...)



# Java

## Éléments de base

---

### ■ Commentaires

- // Ceci est un commentaire sur une seule ligne

- /\* Ceci est un commentaire  
multi-lignes

\*/

- /\*\* Ceci est un commentaire que javadoc va  
utiliser pour créer la documentation sous forme  
HTML

\*/



# Java

## Éléments de base

---

### ■ Types de données primitifs

- boolean : true ou false
- char : 16 bits
- byte : 8 bits -128 au +127
- short : 16 bits
- int : 32 bits
- long : 64 bits
- float : 32 bits
- double : 64 bits



# Java

## Éléments de base

---

### ■ Opérateurs

- Incrémentation, décrémentation  
++ --
- arithmétiques  
\* / %  
+ -
- de comparaison  
< <= > >=  
== !=
- logiques  
&& || ! (non logique)
- affectation dans la variable de gauche  
= += -= \*= /= %=



# Java

## Éléments de base

---

### ■ Opérateurs : exemples

- `i++;`
  - utiliser `i` et incrémenter par 1 après
- `++i;`
  - incrémenter `i` et utiliser après
- `i += 5;`
  - est équivalent à `i = i + 5`
- `i = i < 3 ? i + 1 : i - 1;`
  - Exécuter `i + 1` si `i < 3` et `i - 1` sinon

## ■ Opérateurs et leur priorité

### Priorité des opérateurs

+++++	()	[]	.					
+++++	--	++	!	~	-(un opérande)	+(un opérande)	(casting)	new
+++++	*	/	%					
+++++	+	-						
+++++	<<	>>	>>>					
+++++	<	<=	>=	>	instanceof			
+++++	==	!=						
+++++	&							
+++++	^							
+++++								
+++++	&&							
+++++								
+++++	?	:						
+++	=	+=	-=	*=	/=	%=	&=	
++	<<=	>>=	>>>=	^=	=			
+	,							



# Java

## Éléments de base

---

### ■ Conversion de type

- Appelée aussi *opération de cast*, consiste en une modification du type de donnée forcée
- Cela signifie que l'on utilise un opérateur dit *de cast* pour spécifier la conversion
- L'opérateur de cast est tout simplement le type de donnée, dans lequel on désire convertir une variable, entre des parenthèses précédant la variable
  - Exemple : conversion de somme en double pour avoir une division en nombres flottants

```
double moyen;  
int somme, nombre;  
moyen = (double) somme / nombre;
```
- Note : une conversion explicite peut engendrer une perte de précision
  - Exemple : conversion d'un double en un int





# Java

## Éléments de base

---

### ■ Saisir des données au clavier

- Java propose la fonction `System.in.read()`
- Cette fonction est définie dans la bibliothèque `System`, à l'intérieur du sous-ensemble `in`
- Elle utilise le programme de lecture au clavier `read()`
- Cette fonction permet de lire un et un seul caractère saisi au clavier, ce qui est complexe à faire pour des chaînes ou des entiers...
- Nous proposons pour l'instant la classe `Lire.java` que vous trouverez dans Cours1-Exemples
  - . Elle permet de saisir autant de caractères que l'on veut
  - . Pour terminer la saisie, il suffit de taper entrée
  - . On peut saisir aussi bien des valeurs entières que réelles, que des chaînes de caractères...
  - . Un exemple est donnée : `TestLire.java`

testLire.java

```
public class TestLire {  
    public static void main (String  
        [] arg) {  
        byte val_byte;  
        short val_short;  
        int val_int;  
        long val_long;  
        float val_float;  
        double val_double;  
        char val_char;  
        String val_String;  
  
        //Saisir une valeur de type byte  
        System.out.print("Entrez un  
        byte : ");  
        val_byte = Lire.b();  
  
        //Saisir une valeur de type short  
        System.out.print("Entrez un  
        short : ");  
        val_short = Lire.s();
```

```
        //Saisir une valeur de type int  
        System.out.print("Entrez un int : ");  
        val_int = Lire.i();  
  
        //Saisir une valeur de type long  
        System.out.print("Entrez un long : ");  
        val_long = Lire.l();  
  
        //Saisir une valeur de type float  
        System.out.print("Entrez un float : ");  
        val_float = Lire.f();  
  
        //Saisir une valeur de type double  
        System.out.print("Entrez un double :  
        ");  
        val_double = Lire.d();  
  
        //Saisir une valeur de type String  
        System.out.print("Entrez un String:  
        ");  
        val_String = Lire.S();  
  
        //Saisir une valeur de type char  
        System.out.print("Entrez un char: ");  
        val_char= Lire.c();
```

```
// Afficher les différentes valeurs lues au clavier
```

```
System.out.println("Vous avez entré le byte : " + val_byte);  
System.out.println("Vous avez entré le short : " + val_short);  
System.out.println("Vous avez entré le int : " + val_int);  
System.out.println("Vous avez entré le long : " + val_long);  
System.out.println("Vous avez entré le float: " + val_float);  
System.out.println("Vous avez entré le double: " + val_double);  
System.out.println("Vous avez entré le String: " + val_String);  
System.out.println("Vous avez entré le char: " + val_char);  
    }  
}
```



# Java

## Éléments de base

---

### ■ Sélection

```
if (expression) {  
    bloc de code  
}  
else {  
    bloc de code  
}
```



# Java

## Éléments de base

---

- Sélection : Exemple : ifElseDemo.java

```
class IfElseDemo {  
    public static void main(String[] args) {  
        int testscore = 76;  
        char grade;  
        if (testscore >= 90) {  
            grade = 'A';  
        } else if (testscore >= 80) {  
            grade = 'B';  
        } else if (testscore >= 70) {  
            grade = 'C';  
        } else if (testscore >= 60) {  
            grade = 'D';  
        } else {grade = 'F';}  
        System.out.println("Grade = " + grade);  
    }  
}
```



# Java

## Éléments de base

---

- Sélection : switch

```
switch (expression) {  
    case constante1 : {  
        bloc de code  
        break;  
    }  
    case constante2 : {  
        bloc de code  
        break;  
    }  
    ...  
    default : {  
        bloc de code  
        break;  
    }  
}
```

## ■ Sélection : switch : SwitchDemo.java

```
class SwitchDemo {
    public static void main(String[] args) {
        int month = 8;
        switch (month) {
            case 1: System.out.println("January"); break;
            case 2: System.out.println("February"); break;
            case 3: System.out.println("March"); break;
            case 4: System.out.println("April"); break;
            case 5: System.out.println("May"); break;
            case 6: System.out.println("June"); break;
            case 7: System.out.println("July"); break;
            case 8: System.out.println("August"); break;
            case 9: System.out.println("September"); break;
            case 10: System.out.println("October"); break;
            case 11: System.out.println("November"); break;
            case 12: System.out.println("December"); break;
            default: System.out.println("Invalid month.");break;
        }
    }
}
```



# Java

## Éléments de base

---

- **Itération : 3 formes**

```
while (condition) {  
    bloc de code  
}
```

```
do {  
    bloc de code  
}while(condition);
```

```
for (expression1; condition2; expression3) {  
    bloc de code  
}
```





# Java

## Éléments de base

---

- **while** : exemple

```
class WhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        while (count < 11) {  
            System.out.println("Count is: " +  
count);  
            count++;  
        }  
    }  
}
```



# Java

## Éléments de base

---

- do while : exemple

```
class DoWhileDemo {  
    public static void main(String[] args){  
        int count = 1;  
        do {  
            System.out.println("Count is: " +  
count);  
            count++;  
        } while (count <= 11);  
    }  
}
```



# Java

## Éléments de base

---

- **for** : exemple : Moyenne.java

```
int somme, i;  
    somme = 0;  
    for ( i = 0; i < args.length; i++ )  
        somme += Integer.parseInt(args[i]);  
    if ( args.length > 0 )  
        System.out.println("moyenne : " +  
        (float)somme/args.length); // attention,  
        division sur les entiers  
    }
```