Initiation à la programmation orientée-objet avec le langage Java

Module : POO
Enseignante: BOUZAROURA Ahlam
Année universitaire: 2019-2020

Etapes de programmation

- **Écrire un programme dans un langage (e.g. Java)**
- **Compiler le programme**
  - Traduire le programme dans un langage de bas niveau (machine)
  - [éventuellement optimisation]
  - Produire un programme (code) exécutable
- **Exécution**
  - Charger le programme en mémoire (typiquement en tapant le nom du programme exécutable)
  - Exécution

Termes liés à la Programmation

- **Programme source, code source**
  - Programme écrit dans un langage
- **Code machine, code exécutable**
  - Programme dans un langage de machine, directement exécutable par la machine
- **Compilation (compilateur)**
  - Traduire un code source en code exécutable
- **Interpréteur**
  - Certains langages n'ont pas besoin d'être traduit en code machine
  - La machine effectue la traduction sur la volée (*on the fly*), instruction par instruction, et l'exécute JavaScript

Langage de Programmation

- **Syntaxe d'un langage**
  - Comment formuler une instruction correcte (grammaire)
- **Sémantique**
  - Ce que l'instruction réalise
- **Erreur**
  - de compilation: typiquement reliée à la syntaxe
  - d'exécution: sémantique (souvent plus difficile à détecter et corriger)

Langage Java

- **Langage orienté objet**
  - Notions de classes, héritage, …
- **Beaucoup d'outils disponibles (packages)**
  - JDK (*Java Development Kit*)
- **Historique**
  - Sun Microsystems
  - 1991: conception d'un langage indépendant du hardware
  - 1994: browser de HotJava, applets
  - 1996: Microsoft et Netscape commencent à soutenir
  - 1998: l'édition Java 2: plus stable, énorme librairie

Principe de Java -1-

- **Compiler un programme en *Byte Code***
  - *Byte code*: indépendant de la machine
  - Interprété par la machine

- ***La compilation :***

javac programme.java

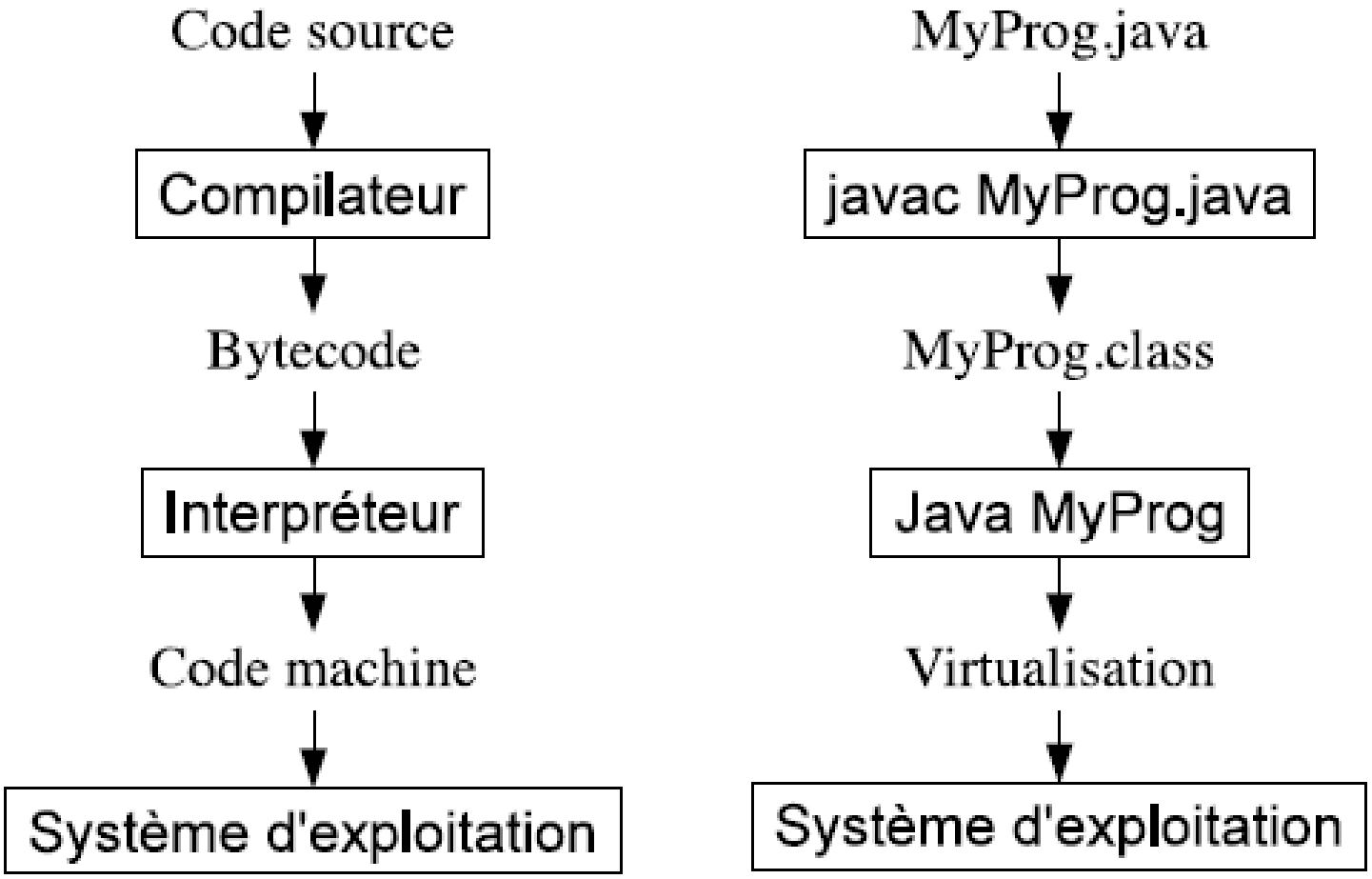
  - Génère programme.class

- ***L'execution***

java programme

  - Lance le programme

Principe de Java -2-



Principe de Java -3-

- Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est .java.
- Ce code source est alors compilé par le compilateur javac en *bytecode* qui n'est pas directement utilisable.
- Le *bytecode* doit être interprété par la *machine virtuelle* de Java qui transforme alors le code compilé en code machine compréhensible par le système d'exploitation.

C'est la raison pour laquelle Java est un langage portable : le bytecode reste le même quelque soit l'environnement d'exécution.

Syntaxe du langage Java -1-

Le langage C a servi de base pour la syntaxe du langage Java :

- le caractère de fin d'une instruction est " ;"

```
a = c + c ;
```

- les commentaires (non traités par le compilateur) se situent entre les symboles "/*" et "*/"

ou commencent par le symbole "//" en se terminant à la fin de la ligne

```
int a ; // ce commentaire tient sur une ligne
int b ;
```

ou

```
/*Ce commentaire nécessite
2 lignes*/
int a ;.
```

Syntaxe du langage Java -2-

les identificateurs de variables ou de méthodes acceptent les caractères {a..z}, {A..Z},$, _ainsi que les caractères {0..9} s'ils ne sont pas le premier caractère de l'identificateur. Il faut évidemment que l'identificateur ne soit pas un mot réservé du langage (comme int ou for).

Ex : **`mon_entier`** et **`ok4all`** **sont des identificateurs valides**

`mon-entier` et **`4all`** **ne sont pas valides pour des identificateurs**.

Types primitifs

ce tableau liste l'ensemble des types primitifs de données de Java.

Type	Classe éq.	Valeurs	Portée	Défaut
boolean	Boolean	true ou false	N/A	false
byte	Byte	entier signé	{-128..128}	0
char	Character	caractère	{\u0000..\uFFFF}	\u0000
short	Short	entier signé	{-32768..32767}	0
int	Integer	entier signé	{-2147483648..2147483647}	0
long	Long	entier signé	$\{-2^{31}..2^{31}-1\}$	0
float	Float	réel signé	$\{-3,4028234^{38}..3,4028234^{38}\}$ $\{-1,40239846^{-45}..1,40239846^{-45}\}$	0.0
double	Double	réel signé	$\{-1,797693134^{308}..1,797693134^{308}\}$ $\{-4,94065645^{-324}..4,94065645^{-324}\}$	0.0

Remarques importantes

En plus de ces types primitifs, le terme ***void*** est utilisé pour spécifier le retour vide ou une absence de paramètres d'une méthode.

A l'inverse du langage C, Java est un langage très rigoureux sur le typage des données. Il est interdit
d'affecter à une variable la valeur d'une variable d'un type différent si cette seconde variable n'est pas explicitement transformée. exemple :

```
int a ;
double b = 5.0 ;
a = b ;
```

est interdit et doit être écrit de la manière suivante :

```
int a ;
double b = 5.0 ;
a = (int)b ;
```

Tableaux

Une variable est déclarée comme un tableau dès lors que des crochets sont présents soit après son type, soit après son identificateur. Les deux syntaxes suivantes sont acceptées pour déclarer un tableau d'entiers (même si la première, non autorisée en C, est plus intuitive) :

```
int[] mon_tableau ;
int mon_tableau2[];
```

Un tableau a toujours une taille fixe qui doit être précisée avant l'affectation de valeurs à ses indices, de la manière suivante :

```
int[] mon_tableau = new int[20];
```

Chaines de caractères -1-

Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau. On utilise une classe particulière, nommée String, fournie dans le package java.lang.

Les variables de type String ont les caractéristiques suivantes :

- leur valeur ne peut pas être modifiée
- on peut utiliser l'opérateur + pour concaténer deux chaînes de caractères :

Chaines de caractères -2-

```
String s1 = ”hello” ;
String s2 = ”world” ;
String s3 = s1 + ” ” + s2 ;
```

//Après ces instructions s3 vaut ”hello world”

l’initialisation d’une chaîne de caractères s’écrit :

```
String s = new String(); //pour une chaine vide
String s2 = new String(”hello world”);
```

// pour une chaîne de valeur ”hello world”

- un ensemble de méthodes de la classe **java.lang.String** permettent d’effectuer des opérations ou des tests sur une chaîne de caractères

Structures de contrôle

- **Instructions conditionnelles**

Syntaxe :

`if (<condition>) <bloc1> [else <bloc2>]];`

Exemple :

```
if (a == b) {
a = 50 ;
b = 0 ;
} else {
a = a - 1 ;
{
```

Boucles

Les boucles peuvent exécuter un bloc de code tant qu'une condition spécifiée est atteinte.

Les boucles sont à portée de main, car elles permet de:

- gagner du temps,
- réduire les erreurs
- rendre le code plus lisible.

While

La boucle while (tantque …. faire) répète le bloc de code aussi longtemps que la condition spécifiée est vérifiée (***true***):

- **Syntaxe**:

```
for (condition) {
   // code a executé
}
```

- **Exemple**:

```
for (int i = 0; i < 5; i++) {
   System.out.println(i);
}
```

For

Quand vous savez exactement combien de fois que vous voulez faire une boucle à travers un bloc de code, utilisez la boucle ***For*** au lieu d'une boucle ***while***:

- **Syntaxe**:

```
while (condition) {
  // code a executé
}
```

- **Exemple**:

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;}
```

Do / While

Cette boucle exécutera le bloc de code une fois, avant de vérifier si la condition est vraie, alors **le code est répétée** tant que la condition est **vraie (true)**.

- **Syntaxe**:

```
do {
   // code a excuté}
while (condition);
```

- **Exemple**:

```
int i = 0;
while (i < 5) {
   System.out.println(i);
   i++; }
```

Méthode en Java-1-

- Une méthode est composée de deux éléments : la déclaration (entête, signature,. . .) et l'implémentation (corps,. . .).
- Une méthode est déclarée par son type de retour, son nom et une liste d'arguments (qui peut être vide).
- la liste des arguments est entre parenthèse, les arguments sont déclarés exactement comme des variables (type puis nom)
- le type de retour peut être un type primitif, une classe ou *void* si la méthode ne renvoie rien.

Méthode en Java-2-

- Après la parenthèse fermante, on place le code de l'implémentation dans une paire d'accolades.
- Si la méthode renvoie quelque chose on doit impérativement trouver le mot **return** suivi d'un élément du type correspondant à celui déclaré dans l'implémentation.
- **Exemple:**

```
double calculMoyenne() {
return ((double)notes) / nombreDeNotes ;
 {
```

Programme en Java

Un programme écrit en Java consiste en un ensemble de classes représentant les éléments manipulés dans le programme et les traitements associés.

L'exécution du programme commence par l'exécution d'une classe qui doit implémenter une méthode particulière "`public static void main(String[] args)`". Les classes implémentant cette méthode sont appellées **classes exécutables**.

Entrée / Sortie

- Pour afficher une chaine de caractères, ou des variable on utilise:

```
System.out.println("..............");
```

- Pour lire une variable depuis le clavier vous allez utiliser **Scanner** comme suit:

1. Importer la classe **Scanner** en ajoutant au début de programme:

```
import java.util.Scanner;
```

2. Ensuite il faut créer un objet de la classe Scanner ajouter:

```
Scanner sc = new Scanner(System.in);
```

- Pour lire une variable **réelle** on ajoute: `float x=sc.nextFloat()`
- Pour lire une variable **entière** on ajoute: `int a=sc.nextInt();`
- Pour lire une variable **String** on ajoute: `String S sc.nextLine()`

Mon Premier Programme en Java

```
public class Hello
{
  public static void main(String[] args)
  {
      Scanner sc= nex Scanner(System,in);
      System.out.println("Hello, World!");
  }
}
```

- **Stocker ce programme dans le fichier Hello.java**

Programme en Java

```
import java.util.Scanner;
public class exemple1 {
public static void main(String[] args){
        Scanner sc= newScanner(System.in);
        int x= sc.nextInt();
        int y= sc.nextInt();
        int z= x+y;
        System.out.println(z);
        float a = sc.nextFloat();
        System.out.println(a);
        String s1= sc.nextLine();
        System.out.println(s1);}
```

Packages -1-

- En 2009, Sun Microsystems est racheté par Oracle Corporation qui fournit dorénavant les outils de développement **Java SE** (Standard Edition) contenus dans le **Java Development Kit (JDK).**
- Un grand nombre de classes, fournies par , implémentent des données et traitements génériques utilisables par un grand nombre d'applications. Ces classes forment l'API (*Application Programmer Interface)* du langage Java. Une documentation en ligne pour l'API java est disponible à l'URL :
- ***http://docs.oracle.com/javase/7/docs/api/***

Packages -2-

- Toutes ces classes sont organisées en packages (ou bibliothèques) dédiés à un thème précis.
- Parmi les packages les plus utilisés, on peut citer les suivants :
- **java.awt :**Classes graphiques et de gestion d'interfaces
- **java.io:** Gestion des entrées/sorties
- **java.lang:** Classes de base (importé par défaut)
- **java.util :**Classes utilitaires
- **javax.swing:** Autres classes graphiques

Packages -3-

- Pour accéder à une classe d'un package donné, il faut préalablement importer cette classe ou son package. Par exemple, la classe Date appartenant au package **java.util** qui implémente un ensemble de méthodes de traitement sur une date peut être importée de deux manières :

- **une seule classe du package est importée :**

```
import java.util.Date ;
```

- **toutes les classes du package sont importées (même les classes non utilisées) :**

```
import java.util.* ;
```

Packages -4-

Il est possible de créer vos propres packages en précisant, avant la déclaration d'une classe, le package auquel elle appartient. Pour assigner la classe précédente à un package, nommé `poo.cours`,

il faut modifier le fichier de cette classe comme suit :

```
package poo.cours ;
import java.util.Date ;
public class DateMain {
...
}
```

Exemple

- Le programme suivant utilise cette classe pour afficher la date actuelle :

```
import java.util.Date ;
public class DateMain {
public static void main(String[] args) {
Date today = new Date();
System.out.println(”Nous sommes le ” +
today.toString());
{
{
```

```
Nous sommes le: 09/02/2020
```