Java & Génie Logiciel
Cours 1- Introduction

Dr.Sidi CHEIKH

Université de Nouakchott
Master Informatique

Sommaire

« Basé sur le support de cours de E.Duris »

I. Variables et Types

I.Variables et Types

Styles de programmation

- **Style applicatif**
  - Fondé sur l'évaluation d'expressions qui ne dépendent que de la valeur des arguments, et non de l'état de la mémoire
    - On parle aussi de programmation fonctionnelle
    - Proche des notations mathématiques, utilise beaucoup la récursivité
    - Accepte des arguments, produit un résultat
    - Ex: Lisp, Caml, ML, HaskelBandes de fréquences
- **Style impératif**
  - Fondé sur l'exécution d'instructions qui modifient l'état de la mémoire
    - Utilise beaucoup les itérations et autres structures de contrôle
    - Les structures de données sont fondamentales
    - Ex: Fortran, C, Pascal

I.Variables et Types

Le Style Objet

- C'est un style de programmation où l'on considère que des composants autonomes (les objets) disposent de ressources et de moyens d'interactions entre eux.
- Ces objets représentent des données qui sont modélisées par des classes qui définissent des types
- En plus de la manière dont sont structurés leurs objets, les classes définissent les actions qu'ils peuvent prendre en charge et la manière dont ces actions affectent leur état
  - Ce sont des « messages » ou des « méthodes ».
- Java n'est pas le seul langage objet
  - Simula, Smalltalk, C++, OCaml...

I.Variables et Types

Les avantages de la programmation objet

- Le style objet favorise:
  - La programmation modulaire ;
  - L’abstraction ;
  - La spécialisation.
- L'objectif est de produire du code :
  - facile à développer, à maintenir, à faire évoluer,
  - réutilisable, tout ou en partie, sans avoir besoin de le dupliquer
  - générique, et dont les spécialisations sont transparentes

I.Variables et Types

Les avantages de la programmation objet

- **La modularité :** est un principe de conception qui permet :
  - un couplage faible entre les composants
  - de définir un contrat et les dépendances entre des composants
  - de masquer le détail des implémentations en utilisant une encapsulation forte.
- **L'abstraction :** demande à séparer la définition (d'un type, d'une classe, d'une méthode) de l'implémentation
  - Permet d'identifier un modèle commun à plusieurs composants
  - Ce modèle commun pourra être partagé via le mécanisme d'héritage
- **La spécialisation :** traite des cas particuliers, mais elle doit autant que possible rester transparente:
  - C'est possible grâce à la dérivation

I.Variables et Types

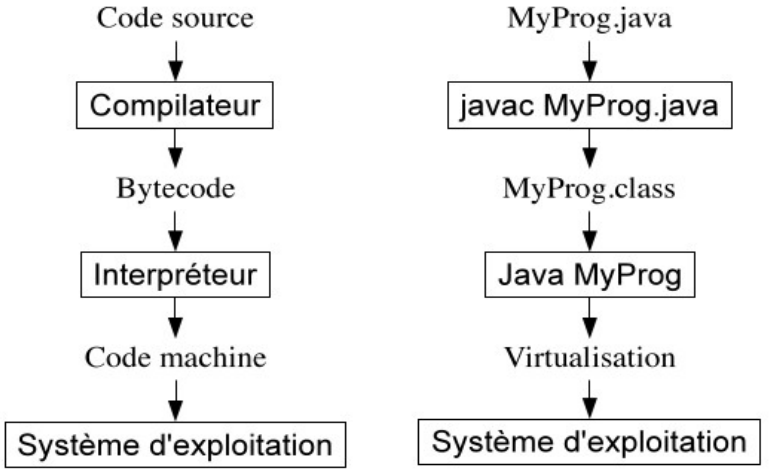
Le langage Java

- est né en 1995 chez Sun Microsystèmes (actuellement Oracle)
- **est orienté objet**
- **est fortement typé**
  - Toute variable doit être déclarée avec un type
  - Le compilateur vérifie que les utilisations des variables sont compatibles avec leur type (notamment via un sous-typage correct)
  - Les types sont d'une part fournis par le langage, mais également par la définition des classes
- **est compilé**
  - En « bytecode », i.e., code intermédiaire indépendant de la machine
- **est interprété**
- Le « bytecode » est interprété par une machine virtuelle Java (JVM).

I.Variables et Types

Le langage Java

- 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 un langage appelé **« bytecode »** et enregistre le résultat dans un fichier dont l'extension est **« .class »** .
- **« 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.



I.Variables et Types

Premier exemple

- Dans un fichier de nom « HelloWorld.java »
  - Règle: toute classe publique doit être dans un fichier qui a les même nom que la classe
  - Règle: tout code doit être à l'intérieur d'une classe

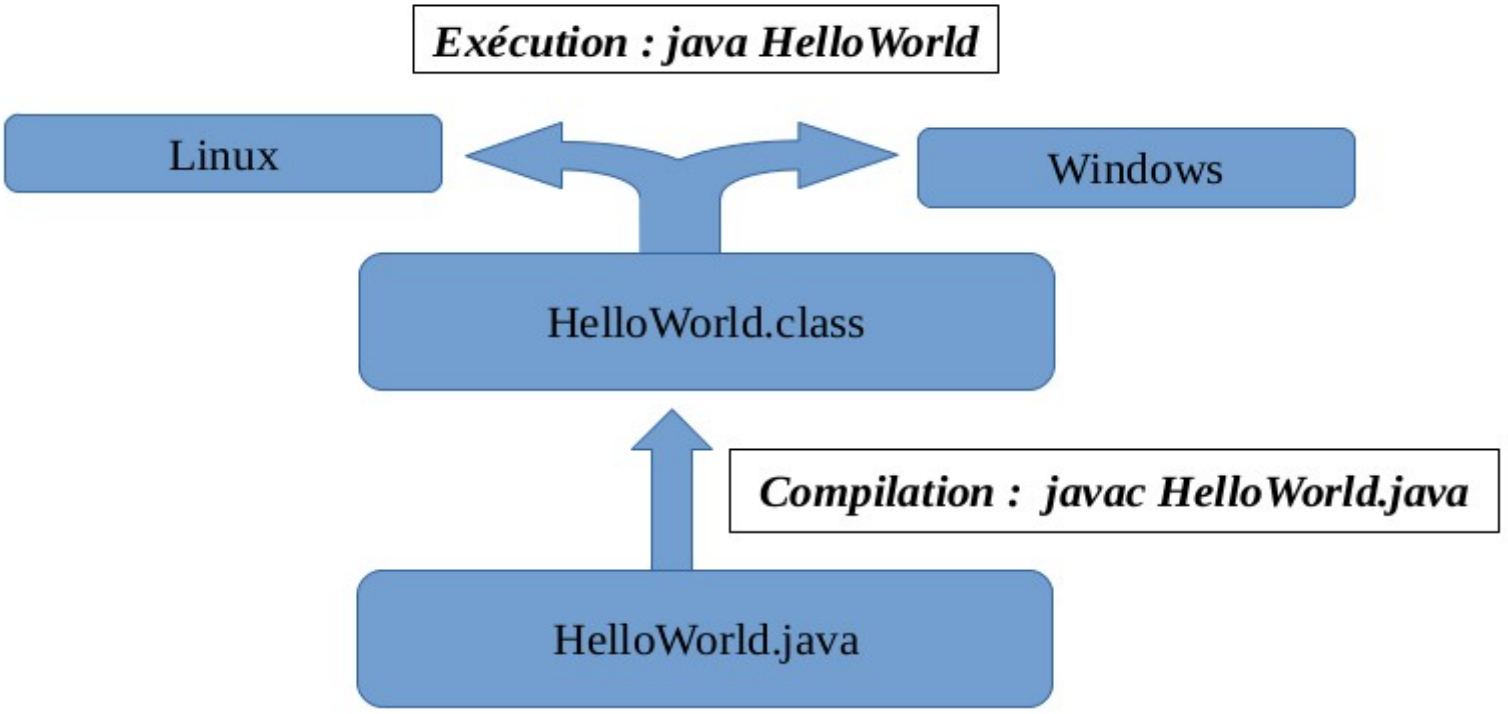
```
public class HelloWorld {

/* Un style de commentaire
sur plusieurs lignes. */

public static void main(String[] args) {

// Un commentaire sur une seule ligne

System.out.println("Bonjour à vous les informaticiens !");
}
}
```

  - Ça définit une classe, qui est une unité de compilation
  - Comme il y a une méthode main, cette classe est **«exécutable ».**

I.Variables et Types

Compilation, bytecode et JVM

- Compilation du langage source > exécution du bytecode
- Compilation (une seule fois : HelloWorld.class)
- Interprétation / exécution (***JVM Windows*** et ***JVM Linux***,…)



I.Variables et Types

La machine virtuelle (JVM)

- Son rôle est d'abstraire le comportement d'une machine
- Son comportement est défini par la « JVM Spec » édité par Sun Oracle
- Une « JVM » est une implémentation de cette spécification
- Pour le rendre le plus possible indépendant de la plateforme
- Qui peut être adaptée à une plateforme d'accueil (Windows, Linux, Mac...)
- Qui peut être développée par Sun (HotSpot: open source GPL depuis 2006) ou par d'autres: IBM, Jikes, etc.
- Une « JVM » traduit le « bytecode » dans le langage machine de la plateforme d'accueil.

I.Variables et Types

Java: un langage et une plateforme

- Dans la technologie Java, on a donc besoin :
  - Du langage de programmation et du compilateur
  - De la « JVM » et des « APIs » (Application Programming Interfaces) regroupées dans une « plateforme »:
  - Et plein de commandes bien utiles: jar, javap, javadoc, etc
  - Java SE (Java Platform, Standard Edition): Java SE pour applications classiques, desktop
  - Java EE (Java Platform, Enterprise Edition): Java EE pour développer et déployer des applications serveur, Web services, etc.
  - Java ME (Java Platform, Micro Edition): J2ME pour les applications embarquées, PDA, téléphones, etc.
- Si on veut juste exécuter, il suffit du JRE (Java Runtime Execution) par opposition au JDK (Java Developpement Kit)

I.Variables et Types

Identificateurs

- Pour désigner les éléments d'un programme, donc pour donner un nom à quelque chose dans un algorithme ou un programme (une variable, une fonction, ...), on utilise en Java un identificateur.
- Un identificateur en Java est une lettre (le caractère _ est une lettre) suivie d'une suite de lettres ou de chiffres de longueur quelconque.
- **Exemples :** x, y, x1, _var, CONST.

Typage

- Un type correspond à la définition de :
  - un ensemble de valeurs,
  - un ensemble d'opérations applicables aux éléments de cet ensemble et la sémantique de ces opérations.
- Toute variable a un type, toute expression a un type.

I.Variables et Types

Typage

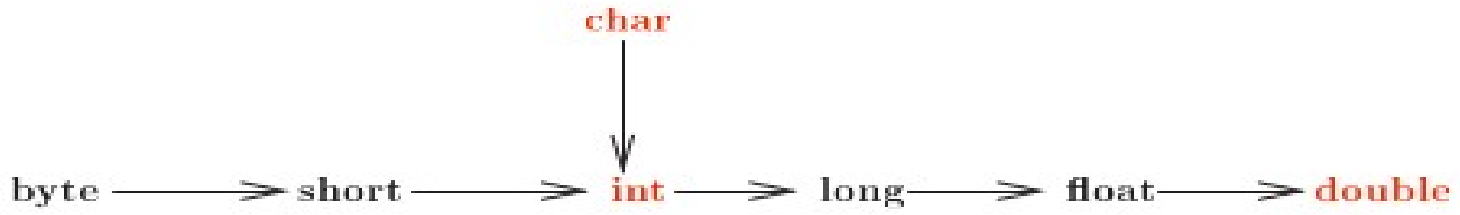
- En Java on distingue :
  - **les types primitifs** (caractères, entiers, réels, booléens).
  - **les types références** (qu'on appelle aussi classes) : les tableaux, les nombreuses classes fournies par les nombreux paquetages (package) Java (comme par exemple la classe « ***String*** **»** qui permet de représenter des chaînes de caractères), et les nouvelles classes définies par les utilisateurs.
  - Types primitifs

Type primitif	Nombre d'octets	Opérations
Entiers signés	byte (1 octet), short (2 octets), int (4 octets), long (8 octets)	arithmétiques : + - * / (division entière) % (reste) comparaison : égal (==), différent (\! =) plus grand (> ou >=), plus petit (< ou ≤) Exemples : 231, 243, 2543L
flottants	float (4 octet), double (8 octets).	arithmétiques : + - * / comparaison : égal (==), différent (!=) plus grand (> ou >=), plus petit (< ou ≤) Exemples : 123.345, 0.12345
char	4 octets	Exemple : char c = 'c', et c+3 est le code de la lettre f.

I.Variables et Types

Typage

- Ordres des types primitifs



Variables

- Une variable est un emplacement en mémoire pouvant contenir une valeur d'un type donné.
- Elle est désignée par un identificateur (son nom) valide (ne doit pas commencer par un chiffre).
- Toute variable doit être déclarée avant d'être utilisée.
- Exemple :

```
final int i=3;
i=4;
```

I.Variables et Types

Expressions

- une expression est composée de variables et/ou de constantes combinées avec des opérateurs.
- une expression a un type et une valeur qui dépendent des éléments qui la constituent.
- Pour un opérateur mélangeant des types compatibles entre eux, c'est celui de type le plus grand qui détermine le type de l'expression.
- Exemple : si i et j sont entiers, i/j est une division entière alors que si l'un des deux ne l'est pas la division est flottante.

```
System.out.println(4/3);
System.out.println(4/3.0);
```

II.Instructions de Contrôle

II.Instructions de Contrôle

```
if(condition){
...
}
else if(condition){
...
}
else{
...
}
```

```
while (test d'arrêt) {
...
}
```

et sa variante :

```
do {
...
}
while (test d'arret)
```

```
switch (x) {
case 0 :
    // instructions
    break;
case 1:
    // instructions
    break;
default :
    // instructions
    break;
}
```

```
for (initialisation;test d'arret;incrementation){
..
}
```

III. Tableaux et chaînes de caractères

III.Tableaux et chaînes de caractères

Généralités

- Un tableau (array) est un regroupement ordonné d'objets (on dit une collection ordonnée), tous du même type et auxquels on accède par un indice entier.
- Si le tableau contient « n » éléments, le premier est d'indice « 0 », le second, d'indice « 1 »,... , et le dernier d'indice « n – 1 ».
- Toute classe ou type primitif peut être utilisé comme entrée d'un tableau :

```
MaClasse[] t;
int[] tableauDEntiers;
String[][] tableauDeTableauDeString;
```

III.Tableaux et chaînes de caractères

Tableaux à une dimension

Un tableau est un objet et n'est donc manipulé qu'à travers une référence (même si ce qu'il contient est un type primitif).

« int[] t » ne définit pas un tableau (même si on le dit fréquemment par abus de langage), ***« t est une référence vers un tableau»***.

Pour créer le tableau il faut le demander explicitement via, par exemple : « new int[10] », qui créé un objet tableau de « 10 int ».

Il faut bien entendu associer l'objet créé à la référence :

```
int []t = new int[10];
```

III.Tableaux et chaînes de caractères

Tableaux à plusieurs dimensions

```
int[][] t=new int[10][10]
for(int i=0;i<10;i++){
    for(int j=0;j<10;j++){
        t[i][j]=i*j;
        }
}
```

- Pour savoir le nombre de lignes, on pourra alors écrire « int l = t.length; »
- Et pour le nombre de colonnes, demander la taille d'une quelconque des entrées de t « int c = t[0].length ».

III.Tableaux et chaînes de caractères

La Classe String

- Cette classe fournie avec la librairie standard de Java permet de manipuler des chaînes de caractères.
- Si on place une chaîne de caractères entre guillemets, c'est un « String » égal à cette chaîne de caractères.
- L'opérateur « + » correspond à la concaténation.
- Exemple :

```
String s =''bonjour'';
s += "toto";
System.out.println(s);
```

- Scanner clavier = new Scanner(System.in);
- String s = clavier.nextLine();

III.Tableaux et chaînes de caractères

Méthodes (Fonctions)

- Souvent on veut écrire nos propres fonctions et leur donner un nom, de façon a pouvoir la réutiliser sans avoir à en réécrire le code.
- En programmation objet, et donc en Java, on appelle en général une fonction une méthode.
- La définition de nouvelles méthodes s'écrit dans le corps de la classe.
- Exemple
- Si une fonction est dans un fichier contenu dans le même répertoire) on peut l'appeler avec son nom complet : NomClasse.nomMethode (par exemple ici ***Calcul.puissance).***

```
class Calcul{
public static int puissance(int x, int n){
int val=1;
    for(int i=1;i<=n;i++){
      val*=x;
       }
    return val;
    }
public static void main(String[] args){
System.out.println("11*11=" +puissance(11,2));
}
}
```

IV.Classes et Objets

IV.Classes et Objets

Classes et objets

- Une classe Toto représente plusieurs choses:
  - Une unité de compilation
    - La compilation d'un programme qui contient une classe Toto produira un fichier Toto.class
  - La définition du type Toto
    - Il peut servir à déclarer des variables comme Toto t;
  - Un moule pour la création d'objets de type Toto
    - Cela nécessite en général la définition d'un ensemble de champs (fields) décrivant l'état d'un objet de ce type et d'un ensemble de méthodes définissant son comportement ou ses fonctionnalités
- Chaque objet de la classe Toto
  - Dispose de son propre état (la valeur de ses champs)
  - Répond au même comportement (via les méthodes de la classe)

IV.Classes et Objets

Structure d'une classe

- Une classe est définie par son nom et son package d'appartenance (ex: java.lang.String)
  - En l'absence de directive, les classes sont dans un package dit « par défaut » (i.e., pas de packge).
- Une classe peut contenir trois sortes de membres
  - Des champs (fields) ou attributs
  - Des méthodes (methods) et constructeurs
  - Des classes internes
- Les membres statiques (static) sont dits membres de classe
  - Ils sont définis sur la classe et non sur les objets
- Les membres non statiques (ou d'instance) ne peuvent exister sans un objet

IV.Classes et Objets

```java
public class Pixel {
    public final static int ORIGIN = 0;
    private int x;
    private int y;
    public Pixel(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void reset() {
        x = ORIGIN;
        y = ORIGIN;
    }
    public void printOnScreen() {
        System.out.println("("+x+","+y+")");
    }
    public static boolean same(Pixel p1, Pixel p2) {
        return (p1.x==p2.x) && (p1.y==p2.y);
    }
    public static void main(String[] args) {
        Pixel p0 = new Pixel(0,0);
        Pixel p1 = new Pixel(1,3);
        p1.printOnScreen();                  // (1,3)
        System.out.println(same(p0,p1)); // false
        p1.reset();
        System.out.println(same(p0,p1)); // true
    }
}
```

Exemple

Constante

Champs

Constructeur

Méthodes d'instances

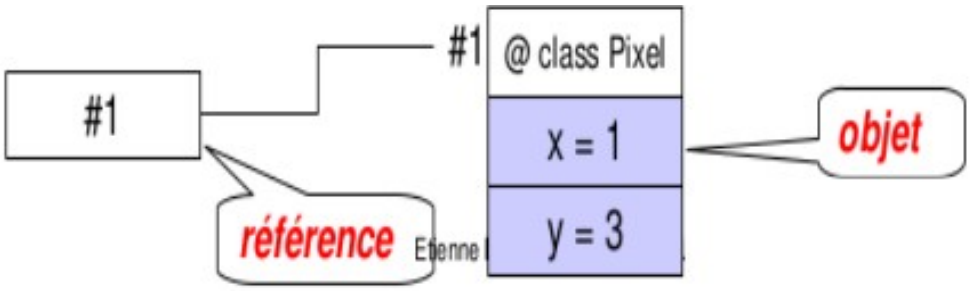
Méthode de classe

Variables locales à la méthode main et objets de la classe Pixel

IV.Classes et Objets

La nature des variables en Java

- Les variables locales comme les champs des classes et des objets ne peuvent être que de deux natures
- De type « primitif »
  - Dans ce cas, la déclaration de la variable réserve la place mémoire pour stocker sa valeur (qui dépend de son type)
- De type « objet », ou référence
- Dans ce cas, la déclaration de la variable ne fait que réserver la place d'une référence (une sorte de pointeur) qui permettra d'accéder à l'endroit en mémoire où est effectivement stocké l'objet en lui même (vaut null si référence inconnue)
- Pixel p1;
-
- p1=new Pixel(1,3)



Questions ?