Premiers pas en java

Arnaud Labourel arnaud.labourel@univ-amu.fr

Section 1

Introduction

Organisation du cours

Volume horaire

- 13 séances de 4h de cours/TD/TP intégré (52h au total)
- 4 séances de 2h de suivi de projet (8h au total)

Évaluation

- un examen sur machine (6 novembre ?) : 50% de la note
- un projet (rendu le 6 décembre) : 50% de la note

Objectifs du cours

- Apprendre à coder proprement
- Apprendre ce que permet le langage Java et ces différences avec Python
- Apprendre à programmer avec des objets
  - adopter le "penser objet"
  - connaître et savoir mettre en œuvre les concepts fondamentaux de la programmation objet

Section 2

Contenu du cours

S'initier à toutes les étapes du développement de logiciels

1. **Analyser** les besoins
2. **Spécifier** les comportements du programme
3. **Choisir** et éventuellement **concevoir** les solutions techniques
4. **Implémenter** le programme (coder)
5. **Vérifier** que le programme a le comportement spécifié (tester)
6. **Déployer** le programme dans son environnement, **fournir** une documentation dans le cas de bibliothèque
7. **Maintenir** le programme (corriger les bugs, ajouter des fonctionnalités)

Apprendre à programmer proprement

Un programme propre :

- respecte les attentes des utilisateurs
- est fiable
- peut évoluer facilement/rapidement
- est compréhensible par tous

Méthode pour programmer proprement

- nommer correctement les éléments du code
- écrire du code lisible (par un autre humain)
- relire et améliorer le code

S'initier au penser objet

Connaître les éléments de base de la programmation objet :

- maîtriser le vocabulaire de la programmation objet : *classe*, *instance*, *méthode*, *attribut*, *constructeur*, *interface*, *extension*, ...
- être capable d'utiliser les objets des classes du JDK (Java Development Kit)
- être capable de coder de nouvelles classes pour définir des nouveaux types d'objets
- savoir décomposer un problème simple en classes et objets
- savoir tester le comportement d'objets

Objectif final de l'enseignement

Être capable de développer une application graphique propre avec une interface utilisateur

Section 3

Premiers pas en Java et différences avec Python

Python

L'année précédente, vous avez codé en Python :

Exemple de code Python :

```python
def is_even(n):
    return n % 2 == 0
```

```
>>> is_even(10)
True
>>> is_even("toto")
...
    return n % 2 == 0
           ~~^~~
TypeError: not all arguments converted during
  string formatting
```

Erreur à l'exécution car `n % 2 == 0` n'est pas possible avec `n="toto"`

Java

```java
static boolean isEven(int n){
  return n % 2 == 0;
}
```

En java, il faut définir le type des arguments

```
>>> isEven(10)
True

>>> isEven("toto")
Error:
|  incompatible types: java.lang.String cannot be
|  converted to int isEven("toto")
```

En cas de type incompatible donné en argument, une erreur se produit avant l'exécution de la fonction.

Typage

Définition d'un type

Un type de données définit :

- l'ensemble des valeurs possibles pour les données du type
- les opérations applicables sur ces données

Java vs Python

- Python :
  - les données/valeurs ont un type
  - les variables/arguments **n**'ont **pas** de types
- Java :
  - les données/valeurs ont un type
  - les variables/arguments ont un type

Typage dynamique

En Python, le type d'une valeur d'une variable est défini lors de son affectation, et le type de la valeur de la variable peut changer au cours d'une exécution.

```
>>> x = 8
>>> x
8
>>> type(x)
<class 'int'>
>>> x = "toto"
>>> type(x)
<class 'str'>
```

$\Rightarrow$ Le typage est dit **dynamique**

Typage statique

En Java, les variables ont chacune un type défini à sa déclaration.

Il détermine les valeurs que peut prendre la variable et ce type ne peut pas changer.

Le compilateur vérifie le typage.

```
int x = 8;
x; /* 8 */
x = "timoleon"; // interdit ! ne compile pas
```

$\Rightarrow$ Le typage est dit **statique**

Résumé typage

Java est un langage **typé statiquement** :

- les variables, arguments et retours de méthode ont des types
- une vérification de compatibilité des types est effectué à la compilation

Python est un langage **typé dynamiquement** :

- les variables, arguments et retours de méthode **n'**ont **pas** de types
- les valeurs ont des types
- des erreurs liés au types se produisent à l'exécution

Analyse du typage statique

Avantages du typage statique

- séparation entre erreurs de types à la compilation et autres erreurs à l'exécution
- aide à la programmation (surtout avec la complétion automatique)
- aide à l'utilisation de bibliothèque (les types aident à comprendre l'usage des fonctions)
- exécutable généralement plus rapide (code plus facile à optimiser)

Désavantages du typage statique

- verbeux (code plus long à écrire)
- généricité plus difficile (difficile de définir des fonctions acceptant plusieurs types)
- code plus difficile à modifier

Autre exemple Python

```
def even_integers(bound):
    list_even_integers = []
    for i in range(0, bound):
        if is_even(i):
            list_even_integers += [i]
    return list_even_integers
```

```
>>> print(even_integers(10))
[0, 2, 4, 6, 8]
```

Même exemple en Java

```
static List<Integer> evenIntegers(int bound) {
  List<Integer> evenIntegers = new ArrayList<>();
  for(int i = 0; i < bound; i++){
    if(isEven(i)){
      evenIntegers.add(i);
    }
  }
  return evenIntegers;
}
```

```
>>> System.out.println(evenIntegers(10));
[0, 2, 4, 6, 8]
```

Section 4

Types primitifs

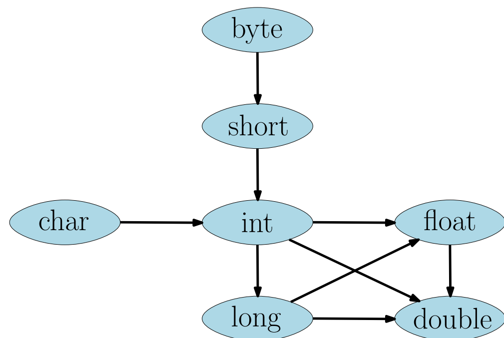
Les types primitifs

En java, il existe des types **primitifs** :

type	catégorie	taille	valeurs possibles	affichage
byte	entier	8 bits	-128 à 127	0
short	entier	16 bits	-32768 à 32767	0
int	entier	32 bits	-2^{31} à $2^{31}-1$	0
long	entier	64 bits	-2^{63} à $2^{63}-1$	0
float	flottant	32 bits		0.0
double	flottant	64 bits		0.0
char	caractère	16 bits	caractère unicode	'\000'
boolean	booléen	non définie	`false` ou `true`	`false`

Conversion implicite

Java convertit implicitement les valeurs dans le sens des flèches.



Exemples :

```
double d = 3;
float f = 10;
```

Conversion explicite

Il est possible de forcer la conversion d'un type à l'autre avec la syntaxe suivante : (*type souhaité*) *valeur*

Exemples :

```
int n = (int) 2.4;
long l = (long) 3.6;
```

C'est utile pour faire un division fractionnaire

```
double d = 1/2; // => 0 car division d'entiers
float f = 1/((float) 2) // => 0.5 car division flottante
```

Section 5

Compilation

Compilation/interprétation

- En Java, le code est généralement compilé puis ensuite exécuté.
- En Python, le code est généralement directement interprété.

Compilation

code source (langage de programmation) → code exécutable (langage machine)

Plusieurs phases :

- analyse lexicale (découpe le code en *tokens*)
- analyse syntaxique (vérifie la syntaxe)
- analyse sémantique (vérifie le typage)

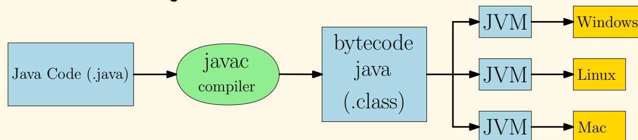
Commandes

- `javac MonFichier.java` pour compiler le fichier `MonFichier.java`
- `java MonFichier` pour exécuter le code de `MonFichier.java`

Compilation et machine virtuelle

Compilation Java

- le compilateur java génère du **bytecode java** = langage machine virtuel
- le code java s'exécute dans une **Java Virtual Machine (JVM)**



Machine virtuelle Java

- la JVM interprète le *bytecode* pour l'exécuter
- la JVM rend le code java indépendant de l'OS et de la machine hôte

$\Rightarrow$ *Compile once, run everywhere*

Exemple compilation → exécution

Pour être exécuté un code `Java` doit définir une méthode `main`.

```
public class App {
  public static void main(String[] args) {
    System.out.println("Hello world !");
  }
}
```

- la fichier doit avoir le même nom que la classe (ici `App` et donc fichier `App.java`)
- La commande `javac App.java` compile le fichier et crée l'exécutable `App.class`.
- La commande `java App` exécute `App.class` et affiche donc `Hello world !`.

Exemple compilation → exécution avec arguments

```
public class App2 {
  public static void main(String[] args){
    for(String arg : args)
      System.out.println(arg);
  }
}
```

- `String[] args` correspond aux arguments de l'appel d'exécution.
- La commande `java App2 to ta ti` exécute `App2.class` avec les arguments `to`, `ta` et `ti` et affiche donc :

```
to
ta
ti
```

Section 6

Tableaux

Tableaux unidimensionnels

En Java, les tableaux sont des objets (et donc des références).

Déclaration d'une variable de type "référence vers un tableau" :

```
int[] arrayOfInt;
double[] arrayOfDouble;
```

Construction d'un tableau :

```
arrayOfInt = new int[10]
arrayOfDouble = new double[3];
```

Utilisation d'un tableau :

```
arrayOfInt[0] = 5;
arrayOfInt[9] = 10;
arrayOfDouble[2] = arrayOfInt[0] / arrayOfInt[9];
system.out.println(arrayOfDouble.length) // 3
```

Tableaux multidimensionnels

Déclaration :

```
int[][] matrixOfInt;
```

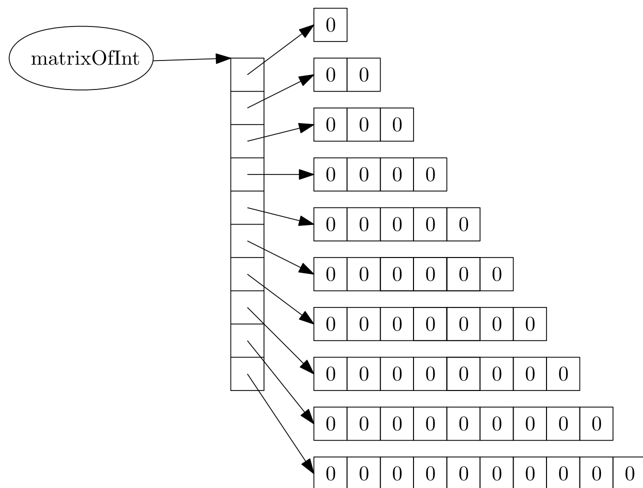
Construction :

```
matrixOfInt = new int[10][];
for(int row = 0; row < matrixOfInt.length; row++)
    matrixOfInt[row] = new int[5];
/* ou directement */
matrix = new int[10][5];
```

Que produit le code suivant (réponse au transparent suivant) ?

```
matrixOfInt = new int[10][];
for(int row = 0; row < matrixOfInt.length; row++)
    matrixOfInt[row] = new int[row + 1];
```

Tableau de tableaux



Règles pour les tableaux en Java

```
TypeOfTheElements[][][][] array
    = new TypeOfTheElements[x][y][z][];
```

Règles

- Le type des éléments d'un tableau peut être n'importe quel type (primitif ou objet).
- Le nombre de [] définit la dimension du tableau.
- On peut construire un tableau en définissant une taille (positive ou nulle) pour au moins une dimension.

Section 7

Résumé Java vs Python

Points communs Java/Python

- concept de fonctions similaire :
  - une fonction est définie par un bloc d'instruction
  - une fonction a des arguments
  - une fonction peut retourner une valeur (instruction `return` similaire dans les deux langages)
- concept de variables/arguments similaire (sauf au niveau du typage)
- concept de bloc d'instructions similaire
- concept de structures de contrôle similaire : `for`, `if`, `while` présents en Java et Python

Différences Java/Python

- Même si les concepts sont similaires, la syntaxe (manière d'écrire est souvent différente) :
  - les blocs en pythons sont définis par l'indentation alors qu'en java ils le sont par les accolades `{ }`.
  - instructions séparées par des `;` en java.
- Typage statique vs typage dynamique $\Rightarrow$ il faut préciser le type des éléments d'une liste en Java.
- Très peu d'opérateurs disponibles sur les objets en Java
  - pas de `+` sur des listes
  - pas de `*` sur les `String` (mais quand même le `+`)
- Les `for` sur les entiers *old school* en Java.
- Normes de nommage différentes pour les variables/arguments/fonctions :
  - `snake_case` en Python
  - `camelCase` en Java
- Compilé en Java vs interprété en Python.