Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : TP 1
Code UE : SBICU05

TP 1 : Prise en main de Java

Introduction

Ce premier TP va vous permettre de familiariser avec la syntaxe (manière d'écrire) du langage Java via un terminal REPL (Read–Eval–Print Loop) qui est une console interactive.

Environnement de travail

- Si vous êtes sur les machines de l'université, il vous faut choisir l'image Lunix (Linux pour TOUS) en VDI.
- Sur votre machine personnelle, il vous faut un terminal et un JDK à jour :
  - **Terminal shell** si vous êtes sous windows, nous vous conseillons d'installer ubuntu pour avoir accès à un terminal shell. Sous MacOS et linux, un terminal shell est disponible de base. Si vous effectuez ce TP depuis votre machine personnelle, il vous faut installer les outils suivants :
  - **JDK version 17 ou plus** la méthode d'installation dépend de votre système d'exploitation:
    - **Windows**: Téléchargez et exécutez l'Installeur windows
    - **MacOS**: Téléchargez et exécutez l'Installeur MacOS
    - **Linux**: exécutez la commande suivante dans le terminal `sudo apt install openjdk-17-jdk`. Un mot de passe administrateur vous sera demandé.

Java en mode console

Comme pour Python il est possible d'ouvrir une console de type REPL pour le langage Java.

1. Ouvrez un terminal (Sur les machines de l'université, menu en haut à gauche et choisissez `Menu -> Outils systémes -> Terminal MATE`) et lancer la commande `jshell`. Vous devriez avoir un affichage similaire à celui ci-dessous :

```
~: jshell
|  Welcome to JShell -- Version 18.0.1.1
|  For an introduction type: /help intro

jshell>
```

Types `int` et `long` (entiers)

1. Écrivez `a = 3` dans la console (en validant avec un appui sur la touche entrée). Que se passe-t-il ? Comment expliquez-vous cela ?

2. Écrivez `int a` dans la console puis réessayer d'écrire `a = 3` Que se passe-t-il ? Comment expliquez-vous cela ? Quelle est la valeur contenue dans la variable `a` ?

3. Écrivez `int b = 8` dans la console. Que se passe-t-il ?

4. Testez le retour des instructions Java suivantes et donnez le sens des opérateurs `+=` et `++` (avant et après une variable) :
   - `int c = 0`
   - `c += 3`
   - `++c`
   - `c++`
   - `c`

5. Exécuter les deux instructions Java suivantes :
   - `int m = 2147483647`
   - `++m`

   Comment expliquez-vous la valeur retournée par la deuxième instruction ?

6. Exécuter les deux instructions Java suivantes :
   - `int m = -2147483648`
   - `--m`

   Comment expliquez-vous la valeur retournée par la deuxième instruction ?

7. Exécuter les deux instructions Java suivantes :
   - `long m = 2147483647`
   - `++m`

   Comment expliquez-vous la valeur retournée par la deuxième instruction ?

8. Quelle sont les valeurs de `Integer.MAX_VALUE`, `Integer.MIN_VALUE`, `Long.MAX_VALUE`, `Long.MIN_VALUE` ?

Types `float` et `double` (flottants)

1. Testez l'instruction : `double f = 3` Comment expliquez-vous ce retour ?

2. Testez l'instruction : `double d = 3/6` Comment expliquez-vous ce retour ?

3. Testez l'instruction : `d = 3.0/6` Comment expliquez-vous ce retour ?

4. Testez l'instruction : `int n = 3.0` Que se passe-t-il ?

5. Testez l'instruction : `long l = (long)3.5` Que se passe-t-il ?

6. Quelle est la valeur de `Math.PI` ?

7. Testez l'instruction : `0.1 + 0.1 + 0.1 == 0.3` Comment expliquez-vous ce retour ?

Type `boolean`

1. Testez l'instruction : `2 != 3` Comment expliquez-vous ce retour ?

2. Testez l'instruction : `2 == 3` Comment expliquez-vous ce retour ?

3. Testez le retour des instructions Java suivantes et donnez le sens des opérateurs `&&`, `||` et `!` :
   - `boolean t = true`
   - `boolean f = false`
   - `t && true`
   - `t && f`
   - `false && f`
   - `t || f`
   - `t || true`
   - `false || f`
   - `!t`
   - `!f`

4. Définissez un variable entière `x` et affectez-lui la valeur 3. Testez l'instruction : `boolean c = (x > 2)&& (x < 4)` Comment expliquez-vous ce retour ? Est-ce que la valeur de `c` change si vous changez la valeur de `x` ?

5. Définissez un variable entière `x` et affectez-lui la valeur 3. Testez l'instruction : `boolean c = (x >= 3.1)&& (x <= 4.1)` Comment expliquez-vous ce retour ?

Type `char`

1. Testez les instructions suivantes. Comment expliquez-vous ces retours ?
   - `char c=0`
   - `--c`
   - `(int) c`

2. Testez les instructions suivantes. Comment expliquez-vous ces retours ?
   - `char ch1 = 'a'`
   - `char ch2 = 'b'`
   - `char ch3 = 'z'`
   - `(int) ch1`
   - `(int) ch2`
   - `(int) ch3`

3. En déduire une expression valant `true` si une variable `ch` (qu'il vous faudra créer !) de type `char` contient une lettre de l'alphabet minuscule comprise entre `'a'` et `'z'`, `false` sinon. Est-ce que votre expression est `true` pour les lettres accentuées comme `'é'` ?

4. Aller plus loin en proposant une expression qui vaut `true` si une variable `ch` de type `char` contient une lettre de l'alphabet comprise entre `'a'` et `'z'` ou `'A'` et `'Z'`, `false` sinon.

5. Soustraire `'A'` à `'a'`. Que remarquez-vous ?

6. En déduire une expression qui, étant donné un caractère minuscule contenu dans une variable `ch`, vaut la même lettre en majuscule. Proposer une expression qui effectue l'inverse (étant donné une majuscule, donne une minuscule).

7. Se rendre sur la documentation Java consacrée à la classe `Character` et y trouver les méthodes `Character.toLowerCase(char ch)` et `Character.toUpperCase(char ch)` qui permettent de résoudre la question précédente de façon alternative. Dans la suite du cours, on préférera l'utilisation de ces méthodes à la manipulation des indices de caractères. Pourquoi est-il mieux d'utiliser ces méthodes ?

Première compilation en Java

1. Créez un fichier `Main.java` ayant le contenu ci-dessous à l'aide d'un éditeur de texte comme `gedit`.

```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

2. Compilez le fichier `Main.java` avec la commande `javac Main.java` Quel fichier a été créé dans le répertoire contenant le fichier `Main.java` ?

3. Exécutez votre programme avec la commande `java Main` Que se passe-t-il ?

4. Ajoutez au programme précédent les deux lignes suivantes après l'instruction `System.out.println("Hello World").`, puis compilez-le :

```java
  for(int index = 0; index < args.length; index++)
    System.out.println(args[index]);
```

5. Exécutez le programme précédent en ajoutant les mots bonjour, tout, le et monde (séparés par des espaces) à la suite de la commande ; vous taperez donc `java Main bonjour tout le monde` dans le terminal. Que se passe-t-il ? Dans la suite, on appellera arguments du programme les chaîne de caractères bonjour, tout, le et monde.

Tableau en Java

En java, il existe deux types de structures pour stocker des éléments indexés par des entiers :

— les tableaux dont on ne peut pas changer la taille une fois créé (le nombre de cases disponibles reste fixe)
— les listes qui peuvent changer de taille (on peut ajouter de nouvelles cases)

Dans ce TP, nous allons travailler sur les tableaux et nous verrons les listes dans le prochain TP.

Tableaux unidimensionnel

En Java, les tableaux permettent de stocker un nombre prédéfini d'objets d'un même type. Pour déclarer une variable tableau, on doit préciser le type de ses éléments, mais pas le nombre d'éléments qu'il contient : `<type elt> [] <nom variable>`.

Exemples :

- `int[] tab1`
- `boolean[] tab2`

1. Déclarer une variable de nom `array` pouvant contenir un tableau de `double`.

La construction d'un tableau se fait en utilisant `new` et en précisant sa taille (son nombre d'éléments) : `new <type elt> [<taille>]`. C'est une instruction permettant d'obtenir un tableau.

Exemples :

- `new int[10]` qui construit un tableau pouvant contenir 10 entiers.
- `new boolean[x+3]` qui construit un tableau pouvant contenir `x+3` booléens en supposant que `x` soit une variable entière.

2. Construisez un tableau pouvant contenir 5 `double`. Quel est l'affichage produit ? Comment l'expliquez-vous ?

Pour pouvoir réutiliser un tableau, il faut le stocker dans une variable. Cela se fait sans difficulté avec l'opérateur d'affectation `=` en mettant à sa gauche une variable de type tableau et à sa droite une instruction de construction de tableau (ou bien une autre variable contenant un tableau).

Exemples :

- `int[] tab1` puis `tab1 = new int[10]` (déclaration puis construction et affectation)
- `boolean[] tab2 = new boolean[x+3]` (déclaration, construction et affectation en une seule ligne)

3. Construisez un tableau pouvant contenir 5 `double` et affectez-le à la variable déjà déclaré `array`.

Les éléments d'un tableau sont numérotés (on dit indexés ou indicés) à partir de `0`. Si `nom tab` est un tableau déjà créé, alors on accède à son i-ème élément par : `<nom tab>[<expr indice>]`

Par exemple si `tab` est un tableau de 10 éléments : `tab[0]`, ... ,`tab[9]` sont des expressions correctes, mais `tab[10]` est une expression incorrecte (on sort des indices possibles car le tableau est de taille 10).

Il est possible d'affecter les cases du tableau (partie gauche d'une affectation) ou bien d'accéder à leur valeur (par exemple dans la partie droite d'une affectation).

Exemples :

```
1  tab[0] = 1;
```

```
int x = 5;
tab[x-2] = 4;
tab[9] = 10;
```

4. Changez les valeurs contenues dans le tableau `array` pour qu'il contiennent `1.1`, `2.2`, `3.14`, `4.2` et `100.0` dans cet ordre.

La taille d'un tableau fait partie des attributs fournies par Java : si `<nom tab>` est un tableau déjà créé, alors `<nom tab>.length` est son nombre d'éléments. Par exemple, `tab.length` vaut `10` si `tab` est un tableau de 10 éléments.

5. Accéder à la taille du tableau `array` et vérifiez qu'elle correspond bien à la valeur donnée lors de sa construction.

6. Testez l'instruction `array[5]`. Que se passe-t-il ? Faites de même pour les instructions `array[-1]` et `array[1.1]`. Que se passe-t-il ?

Il existe des variantes pour la construction de tableau qui permettent d'affecter directement des valeurs dans les cases.

7. Testez l'instruction `int[] ints = {1, 2, 3, 4, 5}` Quel est la taille du tableau `ints` ?

8. Testez l'instruction `ints = new int[]{1, -2, 4, -8, 16, -32}` Quel est la taille du tableau `ints` ?

9. Comment expliquez-vous que la taille dans le tableau contenu dans la variable `ints` a changé ?

10. Déclarez une variable `integers` de type `int[]`. Affectez le tableau contenu dans la variable `ints` dans la variable `integers`. Que se passe-t-il ?

11. Déclarez une variable `doubles` de type `double[]`

12. Affectez le tableau contenu dans la variable `ints` dans la variable `doubles`. Que se passe-t-il ?

13. Déclarer une variable `longs` de type `long[]` (sans construire de tableau). Accéder à la taille de longs et à l'élément d'indice `0`. Que se passe-t-il ?

14. Déclarer deux variables `floats1` et `floats2` toutes deux de type `float[]`. Construisez un tableau de `float` de taille `10` et affectez-le à la variable `floats1`. Affectez le contenu de la variable `floats1` dans la variable `floats2`.

15. Changez la valeur de l'élément d'indice 3 de `floats1` en `12.3`. Que se passe-t-il pour l'élément d'indice 3 de `floats2` ? Comment l'expliquez-vous ?

16. Changez la valeur de l'élément d'indice 4 de `floats2` en `1.2`. Que se passe-t-il pour l'élément d'indice 4 de `floats1` ? Comment l'expliquez-vous ?

Tableaux multi-dimensionnels

Un tableau à deux dimensions peut être vu comme un tableau dont chaque élément est lui-même un tableau à une dimension. Et on peut généraliser pour les tableaux à trois (ou plus) dimensions. La déclaration et la construction de tableaux à plusieurs dimensions suivent le schéma des tableaux à une dimension avec `<type elt>` qui est un type tableau :

— déclaration : `<type elt> []...[] <nom tab>` (autant de `[]` que de dimensions voulues pour le tableau)
— construction : `<nom tab> = new <type elt> [<expr nb1>] ... [<expr nbn>];` avec `<expr nbi>` la taille voulue pour la i-ème dimension.

Exemple :

```
1 int[][] matrix;
2 matrix = new int[3][2];
3 matrix[1][0] = 4;
4 int[][] gameBoard = new int [4][4];
```

Le tableau construit avec `new int[3][2]` est un tableau regroupant 3 valeurs (3 cases), chacune contenant un tableau de 2 entiers (2 cases).

Il faut obligatoirement fixer la première dimension à la création. Mais les autres dimensions peuvent être fixées ensuite. Conséquence : un tableau n'est pas forcément rectangulaire.

Exemple :

```java
int [][] brokenGameBoard;
brokenGameBoard = new int[4][];
brokenGameBoard[0] = new int[4];
brokenGameBoard[1] = new int[3];
...
brokenGameBoard[3] = new int[4];
```

1. Déclarez une variable `matrix` pouvant contenir un tableau bi-dimensionnel de `double`.

2. Construisez un tableau bi-dimensionnel de `double` de taille 10×8 et affectez-le à la variable `matrix`.

3. Changez les valeurs de la case d'indices 3 et 2 de `matrix` et vérifiez que sa valeur a bien changé.

4. Est-ce qu'il est possible de changer la valeur de la case d'indices 10 et 0 ou de la case d'indices 0 et 10 de `matrix`?

5. Est-ce qu'il est possible de faire en sorte que `matrix[0].length` ait la valeur 10 sans changer le reste de la matrice ?

6. Créez un tableau bi-dimensionnel d'entiers (`int`) de sorte qu'il contienne 3 cases : celle d'indice 0 contenant un tableau de taille 3, celle d'indice 1 contenant un tableau de taille 2 et celle d'indice 2 contenant un tableau de taille 1.

Liste en Java

En Java, il est possible de créer une liste d'entiers grâce à l'instruction suivante :

```java
List<Integer> l = new ArrayList<Integer>();
```

Les listes en Java sont des collections d'objets ordonnés. Nous allons seulement utiliser quelques méthodes de

cette classe. Pour plus de détails sur les listes, nous vous conseillons (comme dans la plupart des cas pour Java) de vous référer à la documentation officielle de `List`.

Pour cet exercice, nous allons seulement considérer des listes d'`Integer`. `Integer` est une classe encapsulant le type primitif `int`. Cela permet de manipuler des entiers en tant qu'objet, car les `int` ne sont pas des objets en Java.

Dans cet exercice, on va utiliser les méthodes suivantes de `List` :

— `add(Integer e)` : ajoute l'entier e à la fin de la liste
— `equals(Object o)` : compare la liste avec l'objet `o` et renvoie `true` si `o` est une liste contenant les éléments de la liste dans le même ordre.

But de l'exercice

— Comprendre l'utilisation des méthodes de `List` et d'`Object`.
— Comprendre le concept de référence

Déroulé de l'exercice

Vous devez réaliser les tâches suivantes :

1. Lancez la commande jshell dans un terminal (si ce n'est pas déjà fait).
2. Définissez et créez une liste `l1` en `ArrayList<Integer>()` en tapant l'instruction `List<Integer> l1 = new ArrayList<>();`
3. Vérifiez que la liste est bien vide en tapant `l1`.
4. Ajoutez `1` à la fin de la liste `l1` à l'aide de la méthode `add`.
5. Vérifiez que la liste `l1` contient bien `1` en tapant `l1`.
6. Ajoutez `2` à la fin de la liste `l1` à l'aide de la méthode `add`.
7. Vérifiez que la liste `l1` contient bien `1` et `2` en tapant `l1`.
8. Définissez une liste `l2` en tapant l'instruction `List<Integer> l2 = l1`.
9. Quels sont les entiers contenus dans `l2` ? Pourquoi `l2` contient ces entiers ?
10. Ajoutez `3` à la fin de la liste `l1` à l'aide de la méthode `add`.
11. Vérifiez le contenu de la liste `l2`. Est-ce qu'il a changé ?
12. Définissez et créez une liste l3 en tapant l'instruction `List<Integer> l3 = new ArrayList<>();`.
13. Ajoutez `1` puis `2` puis `3` à la fin de la liste `l3` à l'aide de la méthode `add`.
14. Vérifiez que la liste `l3` contient bien `1`, `2` et `3` en tapant `l3`.
15. Comparer les listes `l1` et `l2` à l'aide de l'opérateur `==` en tapant `l1 == l2`.
16. Faites de même pour comparer `l1` et `l3` ainsi que `l2` et `l3`.
17. Comparer les listes `l1` et `l2` à l'aide de la méthode `equals` en tapant `l1.equals(l2)`.
18. Faites de même pour comparer `l1` et `l3` ainsi que `l2` et `l3`.
19. Comment expliquez-vous le résultat des comparaisons ?
20. Arréter jshell en tapant /exit.