ITI 1521. Introduction à l'informatique II

Précis de **Java**

Marcel Turcotte

uOttawa

École de science informatique et de génie électrique (SIGE)
Université d'Ottawa

Version du 10 janvier 2017

Aperçu

Précis de Java

Nous donnons un exposé concis du langage Java limitant la discussion aux déclarations de types, aux structures de contrôles, et aux appels de méthodes. La programmation orientée objet, les interfaces, et les types génériques sont présentés dans les modules à venir.

Objectif général :

- À la fin de ce module, vous pourrez concevoir des programmes Java du même niveau de complexité que ceux du cours ITI1520.

Vidéo d'introduction :

- `https://www.youtube.com/watch?v=p0sFb7OlDdU`

Objectifs d'apprentissage

- **Nommer** les types primitifs de Java
- **Utiliser** les types références prédéfinis de Java
- **Déclarer** le type des variables
- **Développer** des applications utilisant des tableaux
- **Utiliser** les structures de contrôles pour la résolution de problèmes
- **Appliquer** le principe de décomposition de problème afin de **concevoir** des applications structurées

Lectures :

- Koffman & Wolfgang Appendice A (pages 597-601)

Types **primitifs**

Type	**Taille**	**Maximum**	**Exemples**
boolean	1		**true**,**false**
char	16	'\uFFFF'	'a', 'A', '1', '*'
byte	8	127	-128, -1, 0, 1, 127
short	16	32767	-128, -1, 0, 1, 127
int	32	2147483647	-128, -1, 0, 1, 127
long	64	9223372036854775807	-128L, 0L, 127L
float	32	3.4028235E38	-1.0f, 0.0f, 1.0f
double	64	1.7976931348623157E308	-1.0, 0.0, 1.0

Types **références**

- Les **types références** sont : des **classes**, des **interfaces**, des **types énumératifs**, ou des **tableaux**.
- Les types références sont présentés dans les modules à venir.

Déclaration de **type**

type identifiant

```
int age;
```

- Il faut déclarer le **type** de chaque **variable** et **paramètre**, ainsi que le type de la **valeur de retour** des méthodes.

Erreur de compilation : «cannot find symbol»

```
public class Test {
    public static void main(String[] args) {
        age = 21;
    }
}
```

- Dans l'exemple ci-haut, la variable **age** n'a pas été déclarée.

```
Test.java:3: error: cannot find symbol
        age = 21;
        ^
  symbol:   variable age
  location: class Test
1 error
```

Solution

- Il faut déclarer le **type de la variable**, ici **int** (ligne 3), avant de l'utiliser (ligne 4).

```
1 public class Test {
2     public static void main(String[] args) {
3         int age;
4         age = 21;
5     }
6 }
```

Déclaration de **type** : les méthodes

```
                type          type          type
public int sum(int a , int b) {
    return a+b;
}
```

- Il faut déclarer le **type** de chaque **paramètre**, ainsi que de la **valeur de retour** des méthodes.

Erreur de compilation : **valeur de retour** et **paramètres**

```java
public class Test {
    public sum(a, b) {
        return a+b;
    }
}
```

```
Test.java:2: error: invalid method declaration; return type required
    public sum(a, b) {
           ^
Test.java:2: error: <identifier> expected
    public sum(a, b) {
               ^
Test.java:2: error: <identifier> expected
    public sum(a, b) {
                  ^
3 errors
```

Type de la valeur de retour : **void**

- Certaines méthodes **ne retournent aucun résultat**, c'est le cas de la méthode **swap** ci-dessous, le type de la valeur de retour est alors **void** («ne retourne rien»).

```
public static void swap(int[] xs) {
    int tmp;
    tmp = xs[0];
    xs[0] = xs[1];
    xs[1] = tmp;
}
```

Définition : portée

*La **portée** d'une déclaration est la région du programme à l'intérieur de laquelle on peut référencer l'entité déclarée par la déclaration à l'aide d'un nom simple*

The Java Language Specification,
Third Edition, Addison Wesley, p. 117.

Définition : portée d'une variable locale en Java

> *La **portée de la déclaration d'une variable locale** dans un bloc d'énoncés est le reste du bloc dans lequel cette déclaration apparaît*
>
> The Java Language Specification,
> Third Edition, Addison Wesley, p. 118.

$\Rightarrow$ A.K.A. portée statique ou lexicale

Définition : portée d'un paramètre en Java

*La **portée d'un paramètre** d'une méthode ou d'un constructeur est corps en entier de la méthode ou du constructeur*

The Java Language Specification,
Third Edition, Addison Wesley, p. 118.

$\Rightarrow$ A.K.A. portée statique ou lexicale

Référence vers un tableau

type identifiant

```
int [] xs;
```

- **Déclarer** une **variable référence** vers un **tableau**.
  - La syntaxe [] indique qu'il s'agit d'un tableau. Le type qui précède est le type des éléments du tableau.
  - Dans l'exemple ci-dessus, **xs** est une référence vers un tableau d'entiers.
  - **Important.** Déclarer une variable référence vers un tableau **ne crée pas le tableau**. Seulement la variable référence*.

*Consultez le module sur les types références

Créer un tableau

type taille

```
xs = new int[5];
```

- **Créer** un tableau d'entiers de taille 5.
  - Les **tableaux** sont des objets.
  - Le mot-clé **new** est utilisé pour créer un objet (lors de l'exécution du programme).
  - Le type de l'objet est **tableau d'entiers**.
  - La **taille** du tableau est 5.
  - La référence du tableau est sauvegardée dans une variable nommée **xs**.

Accéder au contenu d'un tableau

index

```
value = xs[0];
```

- **Accéder** au contenu d'un tableau.
  - On utilise le nom de la **variable référence** suivi de l'**index** de la position choisie entre les crochets.

Déclarer, **créer**, et **accéder** au contenu d'un tableau

- La méthode **main déclare** une variable locale de type référence vers un tableau d'entiers (ligne 3), **crée** un tableau d'entiers de taille 5 (ligne 4), et **assigne** la valeur 100 à la position 0 du tableau désigné par la variable référence **xs**.

```
1 public class Test {
2     public static void main(String[] args) {
3         int[] xs;
4         xs = new int[5];
5         xs[0] = 100;
6     }
7 }
```

Initialiser un tableau

- La ligne 4 présente un exemple où l'on **initialise un tableau** avec des valeurs données.

```
1 public class Test {
2     public static void main(String[] args) {
3         int[] xs;
4         xs = new int[] {0, 1, 2, 3, 4, 5};
5     }
6 }
```

Tableau **multidimensionnel**

- **Déclarer** une variable référence de type **tableau à deux dimensions** (ligne 2).
- **Créer** un tableau à deux dimensions (ligne 3).
- **Assigner** une valeur dans un tableau à deux dimensions (lignes 8 et 10).

```
1  int size = 4;
2  double[][] m;
3  m = new double[size][size];
4
5  for (int i=0; i<size; i++) {
6      for (int j=0; j<size; j++) {
7          if (i==j) {
8              m[i][j] = 1.0;
9          } else {
10             m[i][j] = 0.0;
11         }
12     }
13 }
```

Tableau **multidimensionnel**

- Un tableau à **plusieurs dimensions** est un tableau à une dimension dont les cellules contiennent des références vers d'autres tableaux (voir ligne 5).

```
1  double[][] m;
2  m = new double[4][];
3
4  for (int i=0; i<m.length; i++) {
5      m[i] = new double[4];
6      for (int j=0; j<m[i].length; j++) {
7          if (i==j) {
8              m[i][j] = 1.0;
9          } else {
10             m[i][j] = 0.0;
11         }
12     }
13 }
```

Tableau **multidimensionnel**

- Sachant qu'un **tableau à plusieurs dimensions** est un tableau à une dimension dont les cellules contiennent des références vers d'autres tableaux, nous pouvons créer une **matrice triangulaire** et ainsi économiser de l'espace mémoire, si cet espace n'est pas nécessaire.
- Cet exemple illustre aussi l'initialisation d'un tableau à deux dimensions.

```
double[][] m;
m = new double[][] {{1.0},{0.0, 1.0},{0.0, 0.0, 1.0}};
```

Opérateurs **arithmétiques**

Priorité	Opérateur	Description
2	++, --	Incrémente, décrémente postfixe
3	++, --	Incrémente, décrémente préfixe †
3	+, -	Signe unaire
4	*, /, %	Multiplication, division, modulo
5	+, -	Addition, soustraction

† Associativité **droite-à-gauche** pour les opérateurs préfixes.

Exemples :

```
int i=0, sum, a=4, b=6, rest;
i++;
sum = a + b;
rest = sum % 2;
```

Opérateurs **logiques**

Priorité	Opérateur	Description
3	!	Négation †
12	&&	Et logique
13	||	Ou logique

† Associativité `droite-à-gauche` pour les opérateurs préfixes.

Exemple :

```
if (! hasPrize && ! isSelected) {
    isOpen = true;
}
```

Opérateurs **relationnels**

Priorité	Opérateur	Description
7	<, <=	Plus petit, plus petit ou égal
7	>, >=	Plus grand, plus grand ou égal
8	==, !=	Égal, non égal

Exemples :

```
if (age < 3) {
    price = 0.0;
} else if (age <= 13 || age >= 65) {
    price = 11.99;
} else {
    price = 14.50;
}
```

Énoncés

- **Un énoncé** se termine par le **point-virgule** (;)
- L'**énoncé vide** ne comprend que le **point-virgule** (;)
- **Zéro ou plusieurs énoncés** sont regroupés dans un **bloc** à l'aide de parenthèses.

Énoncés : corps d'une méthode

```
public int sum(int a, int b) {
    int value;
    value = a+b;
    return value;
}
```

- Le **corps d'une méthode** est un **bloc d'énoncés**

Énoncés : if

```
if (age >= 18) {
    voters = voters + 1;
    System.out.println("Can vote");
}
```

- Si la branche-vrai de l'énoncé **if** comprends plus d'un énoncé, il faut utilisé un **bloc d'énoncés**.
- Il est **recommandé** de toujours utiliser un bloc.

Énoncés : if-else

```
if (100.0*exams/65.0 < 50.0) {
    grade = 100.0*exams/65.0;
} else {
    grade = laboratories + assignments + exams;
}
```

- Si la branche-faux de l'énoncé **if-else** comprend plus d'un énoncé, il faut utiliser un **bloc d'énoncés**.
- Dans l'exemple ci-haut, les parenthèses ne sont pas nécessaires, mais il est **recommandé** de les mettre.

Énoncés : for

initialisation test incrément

```
for (int i=0; i<10; i++) {
    System.out.println(i);
}
```

- La boucle **for** comprend une expression d'**initialisation**, un **test de boucle**, ainsi qu'une expression d'**incrément**.
- Il est **recommandé** de déclarer le type de la variable qui contrôle la boucle dans l'énoncé d'initialisation afin de limiter sa portée au corps de la boucle.

Énoncés : while

```
int i;
i = 0;

while (i<10) {
    System.out.println(i);
    i++;
}
```

- La boucle **while**.
  - Le corps de la boucle est exécuté tant que la condition, ici **i<10**, est vrai.
- Cet exemple produit le même résultat que l'exemple précédent.

Enoncé : return

```
1  public static int indexOf(String word, String[] words) {
2      for (int i=0; i<words.length; i++) {
3          if (word.equals(words[i])) {
4              return i;
5          }
6      }
7      return -1;
8  }
```

- Toute méthode dont le type de la valeur de retour **n'est pas void** doit posséder un énoncé **return**.
- La méthode termine et retourne la valeur de l'argument de l'énoncé **return**.

Enoncé : return

```
1  public class ArrayUtils {
2      public static int indexOf(String word, String[] words) {
3          for (int i=0; i<words.length; i++) {
4              if (word.equals(words[i])) {
5                  return i;
6              }
7          }
8          return -1;
9      }
10 }
```

- Dans l'exemple ci-haut, si le mot recherché est dans le tableau, alors l'énoncé à la ligne 5 sera exécuté. La valeur de retour sera l'index du mot dans le tableau. Si le mot est absent du tableau, alors l'énoncé de la ligne 8 sera exécuté et la valeur -1 sera utilisée afin de signifier l'absence du mot dans le tableau.

Appel de méthode : ArrayUtils.indexOf("charlie",words)

```
public class Test {

    public static void main(String[] args) {

        int result;

        String[] words;
        words = new String[] {"alpha", "bravo", "tango"};

        for (int i=0; i<words.length; i++) {
            result = ArrayUtils.indexOf(words[i],words);
            System.out.println(result);
        }

        result = ArrayUtils.indexOf("charlie",words);
        System.out.println(result);
    }

}
```

Méthode principale : **main**

- Un programme Java comprend une ou plusieurs classes, mais généralement plusieurs classes.
- L'exemple précédent comprend deux classes : **ArrayUtils** et **Test**.
- L'une des classes comporte une méthode principale.
  - Sa signature est toujours la même :

```
public static void main(String[] args) {
    ;
}
```

  - La méthode principale est la première méthode exécutée lorsqu'on lance une application Java.

Lire les données de l'utilisateur

La **lecture des données** en Java nécessite bien souvent l'utilisation de plusieurs classes et la connaissance des mécanismes d'exceptions.

- Pour les applications simples, la classe **Scanner** peut faire l'affaire.

`https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

Lire les données de l'utilisateur

- On passe à un objet **Scanner** la référence de l'objet associé au clavier, **System.in**.

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

- L'objet **Scanner** possède un ensemble de méthodes permettant notamment de lire le prochain entier, **nextInt**, le prochain nombre en point flottant de précision double, **nextDouble**, ou encore la prochaine ligne, **nextLine**.

`https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

```java
import java.util.Scanner;
import static System.out;

public class Test {
    public static void main(String[] args) {

        Scanner sc; int age; String answer;

        sc = new Scanner(System.in);

        out.print("How old are you? ");
        age = sc.nextInt();

        sc.nextLine();

        out.println("What is your favorite color?");
        answer = sc.nextLine();

        out.print("You are " + age + " years old");
        out.println(" and you like the color " + answer);

    }
}
```

Convention d'écriture

- Le nom des variables, attributs, et méthodes débute par une lettre **minuscule** : `door`.
- Lorsqu'un symbole est **composé de plusieurs mots**, la première lettre de chaque mot est en **majuscule** : `pickOneDoor`.
- Le nom des classes, interfaces, et types énumératifs débute par une **majuscule** : `Statistics`.
- Le nom des constantes (`static final`) est en **majuscule** et les différents mots du symbole sont séparés par le souligné (_) : `MAX_VALUE`.

Résumé

- En Java, il faut **déclarer le type** des **variables** et **paramètres**, ainsi que le type de la **valeur de retour** des méthodes.
- Nous avons présenté la syntaxe afin de **déclarer** une variable référence de type tableau, **créer** un tableau, et **accéder** aux valeurs du **tableau**.
- Les énoncés se terminent par un **point-virgule**.
- Plusieurs énoncés sont regroupés en un **bloc** à l'aide de **parenthèses**.
- Nous avons présenté les principaux **opérateurs** et **énoncés de contrôle**.

N'ont pas été vus

- Les **modificateurs de visibilité** : `public` et `private`.
- Le mot clé `static` associé aux **variables et méthodes de classe**.
- La **notation pointée** : `words.length`.

Ces concepts sont présentés dans le module sur la **programmation orientée objet**.

Prochain module

- Introduction à la **programmation orientée objet**

References I

P. Sestoft.
Java Precisely.
The MIT Press, second edition edition, August 2005.

E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 2e edition, 2010.

Pensez-y !

L'impression de ces notes n'est probablement pas nécessaire !