ITI 1521. Introduction à l'informatique II

Types de données : types primitifs et les types références

by

Marcel Turcotte

Version du 9 janvier 2020

Préambule

Préambule

Aperçu

Types de données : types primitifs et les types références

Nous examinons les avantages des langages fortement typés. Nous comparons les types primitifs et les types références. Nous introduisons les diagrammes de mémoire.

Objectif général :

- Cette semaine, vous serez en mesure de contraster les types primitifs et les types référence.

Vidéo d'introduction :

- `https://www.youtube.com/watch?v=9y3TNuz3kWA`

Préambule

Objectifs d'apprentissage

Objectifs d'apprentissage

- **Nommer** des types primitifs et références prédéfinis.
- **Illustrer** les associations entre objets à l'aide de diagrammes de mémoire.

Lectures :

- Pages 545-551 de E. Koffman et P. Wolfgang.

Préambule

Plan du module

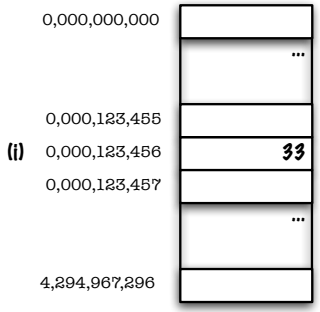
Plan

Théorie

Définition : Variable

Qu'est-ce qu'une **variable** ?

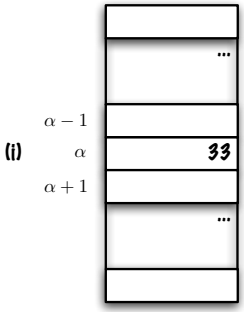
- Une variable est abstraction pour un **emplacement de la mémoire** auquel on réfère à l'aide d'une **étiquette**, dans les langages de haut niveau




```
i = 33;
```

Convention

J'utiliserai des **lettres grecques** pour désigner les emplacements (adresses) en mémoire puisqu'en Java on ne connaît pas l'emplacement des objets et on ne devrait pas s'en préoccuper.




```
i = 33;
```

Définition : Types de données

Qu'est-ce qu'un **type de données** ?

- Un **type de données** fournit des informations sur la **représentation en mémoire** des données (intervalle de valeurs possibles, par exemple) ainsi que sur les **opérations** qui sont définies pour ces données.

Discussion : Types de données

Mais encore, à **qui** servent les types données ?

- Au **compilateur** afin qu'il réserve l'espace mémoire nécessaire pour les données.
- Au **compilateur**, mais aussi au **programmeur**, afin de détecter certaines erreurs au moment de la compilation — appliquer une opération non définie pour un type de données en particulier.

Discussion : Types de données

Donnez des **exemples** de types données ?

- byte, short, int, long
- float, double
- boolean
- char

Types prédéfinis

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

- `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`
- `https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html`

Java : type de données

- Java est un langage **fortement typé**. Ce qui signifie que chaque **variable** et chaque **expression** ont un type connu au moment de la **compilation**.
- 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»

```java
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;
}
```

Type de la valeur de retour (erreur de compilation)

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

```
> javac Test.java
Test.java:2: error: invalid method declaration; return type required
    public static swap(int [] xs) {
                  ^
1 error
```

Type et affectation d'une valeur (erreur de compilation)

```java
public class Test {
    public static void testTypes() {
        boolean b;
        b = "true";
    }
}
```

```
> javac Test.java
Test.java:4: error: incompatible types: String cannot be converted to boolean
        b = "true";
            ^
1 error
```

Solution

```
public class Test {
    public static void testTypes() {
        boolean b;
        b = true;
    }
}
```

Type et expressions

```java
public class Test {
    public static void testTypes() {
        if (3 < 4 && 0) {
            System.out.println("Bingo!");
        }
    }
}
```

```
> javac Test.java
Test.java:3: error: bad operand types for binary operator '&&'
        if (3 < 4 && 0) {
                  ^
  first type:  boolean
  second type: int
1 error
```

Solution

```
public class Test {
    public static void testTypes() {
        if (3 < 4 && 'a' == 'a') {
            System.out.println("Bingo!");
        }
    }
}
```

Java : Types de données (suite)

- On distingue les **types de primitifs** et les **types référence**.
- Qu'est-ce qu'un **type référence** ? Qu'est-ce qu'une **type primitif** ?

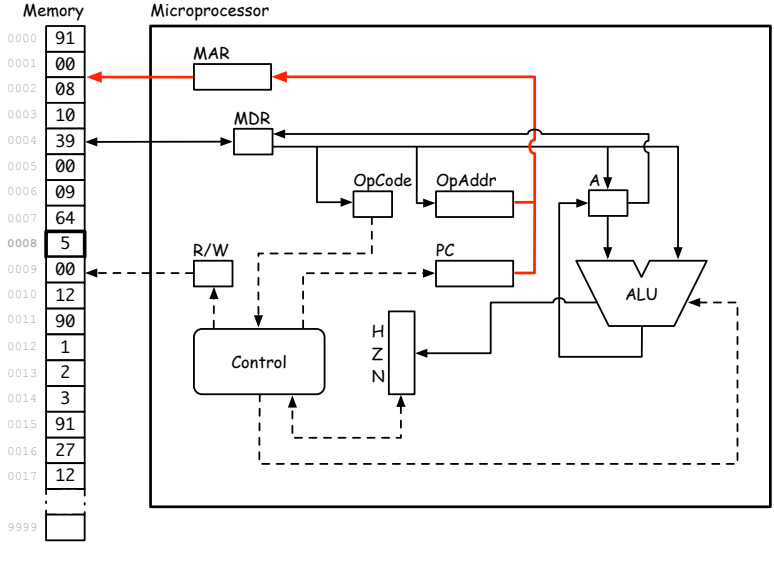
Java : types de données (suite)

- Les types **primitifs** sont :
  - nombres (**byte**, **int**, **long**, **float**, **double**), les caractères (**char**, mais pas les chaînes) et les booléens (**booleanskim**)
  - **la valeur d'une variable d'un type primitif se trouve à l'adresse désignée par l'étiquette (identificateur)**
- **Références** :
  - Prédéfines :
    - Arrays (tableaux)
    - Strings (chaînes de caractères)
  - Les types définis par l'usager, référence vers un objet
  - **La valeur d'une variable de type référence est l'adresse de l'emplacement mémoire de l'objet désigné par la variable — on dit que la variable pointe, désigne ou référence l'objet**

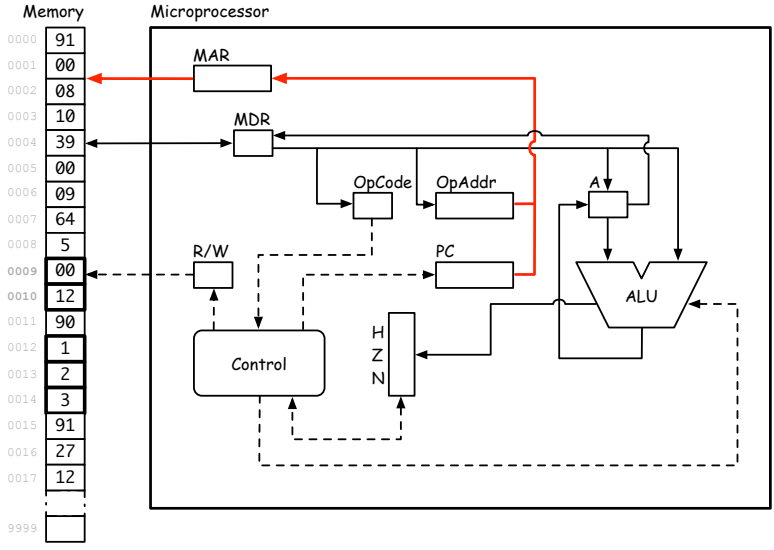
Primitif vs référence et le TC-1101

```
int pos;
pos = 5;

int[] xs;
xs = new int[] {1, 2, 3};
```

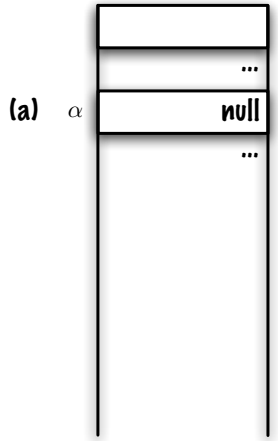


- La variable **pos** est de type **int**, un type primitif, si **pos** désigne l'adresse **00 08**, alors la valeur **5** est sauvegardée à l'adresse **00 08**.



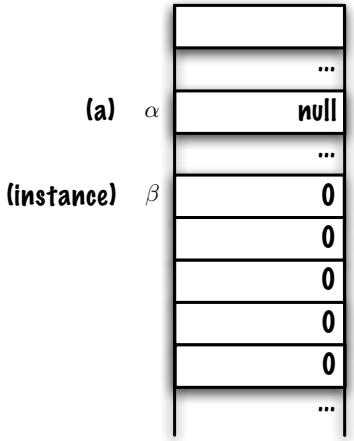
- La variable **xs** est de type **référence** vers un tableau d'entiers, si **xs** désigne l'adresse **00 09**, alors, la valeur des cellules **00 09** et **00 10**, est l'adresse où le tableau a été sauvegardé en mémoire, **00 12**. À l'adresse **00 12** se trouve le tableau, avec ses trois valeurs **1**, **2**, et **3**.

```
> int[] a;
  a = new int[5];
```



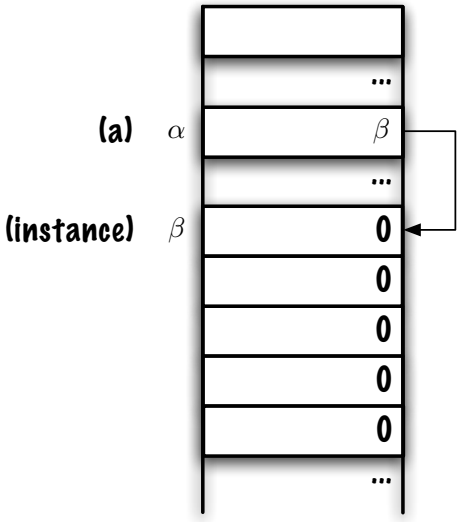
La déclaration d'une variable de type référence **ne crée pas l'objet (instance)**, le compilateur réservera un espace suffisant pour contenir la référence (pointeur), **null** est un littéral qui signifie : ne désigne aucun objet.

```
   int[] a;
> a = new int[ 5 ];
```



La création d'un objet, **new int[5]**, réserve une portion de la mémoire pour 5 entiers (et la gestion interne — *housekeeping*). Chaque cellule du tableau se comporte comme une variable de type **int** et reçoit la valeur initiale 0.

```
    int a[];
>   a = new int[ 5 ];
```



Finalement, la référence du nouvel objet est sauvegardée à l'adresse désignée par l'étiquette **a**.

Théorie

Représentation en mémoire

Diagramme de mémoire

Puisqu'on ne connaît pas l'emplacement des objets en mémoire (et qu'on ne devrait pas s'en préoccuper), nous utiliserons des diagrammes de mémoire (image la plus à droite)

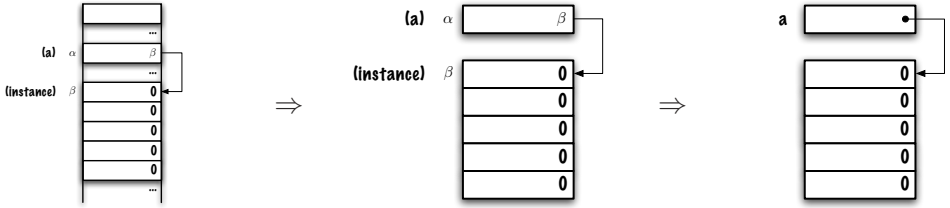


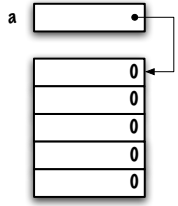
Diagramme de mémoire

Consignes pour vos diagrammes de mémoire :

- Une boîte pour chaque variable **référence** et une **flèche** vers l'objet désigné
- Une boîte pour chaque variable de type **primitif** et sa **valeur** dans la boîte même

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

⇒



Prologue

Résumé

- Une variable est abstraction pour un **emplacement de la mémoire** auquel on réfère à l'aide d'une **étiquette**.
- Un **type de données** fournit des informations sur la **représentation en mémoire** des données (intervalle de valeurs possibles, par exemple) ainsi que sur les **opérations** qui sont définies pour ces données.
- La valeur d'une variable d'un **type primitif** se trouve **à l'adresse désignée par l'étiquette** (identificateur).
- La valeur d'une variable de **type référence** est l'adresse de l'emplacement mémoire de l'objet désigné par la variable.
- En Java, il faut déclarer le type de variables.

Prochain module

- **Types de données** (partie 2)

References I


E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa