Compléments de POO en Java

L2-MIM4A1

Année 2019/2020

Yann Mathet yann.mathet@unicaen.fr

Types simples

Les types entiers

- `byte` (1 octet)
- `short` (2 octets)
- `int` (4 octets)
- `long` (8 octets)

Les types réels

- `float` (4 octets)
- `double` (8 octets)

Booléen : `boolean (1 bit)`

Caractère : `char` (2 octets)

Structures itératives (boucles)

boucle "for"

- On connaît à l'avance le nombre d'itérations
- utilise un compteur qu'elle incrémente à chaque itération

boucle "while"

- On ne connaît pas à l'avance le nombre d'itération
- la boucle est répétée tant qu'une certaine condition est vraie (booléen).

boucle "for"

```
for (initialisation ; condition de continuation ; incrémentation)
        instruction(s) à répéter
```

RAPPEL :
une seule instruction :
instuction;
Plusieurs instructions :
{
instruction1;
instruction2;
etc;
}

Exemple :

```
for (int i=0 ; i<10 ; i++)
        System.out.println("valeur de i = "+i);
```

opérateur de CONCATENATION (mise bout à bout de chaînes de caractères)

boucle "while"

```
while (condition booléenne)
        instruction(s) à répéter
```

Attention :
il est nécessaire de rendre possible l'évolution de la condition booléenne au sein de la boucle, faute de quoi, la boucle est infinie (boucle "folle").

Exemple :

```
while (x!=y)
        {
        System.out.println("Perdu");
        x=Lecture.readInt();
        }
```

CQFD :
x peut changer, donc la condition peut devenir vraie

Similitudes entre "for" et "while"

- En fait, toute boucle "for" peut être remplacée par une boucle "while", et réciproquement.
- Le choix de la boucle se fait donc sur le critère pratique : concision du code, facilité à interpréter
- Exercice : reprendre les deux boucles vues en exemple et les transformer en leur équivalent

Exemple de boucle for :

```
for (int i=0 ; i<10 ; i++)
            System.out.println("valeur de i = "+i);
```

Solution :

```
int i=0;
while (i<10)
    {
    System.out.println("valeur de i = "+i);
    i++;
    }
```

Exemple de boucle while :

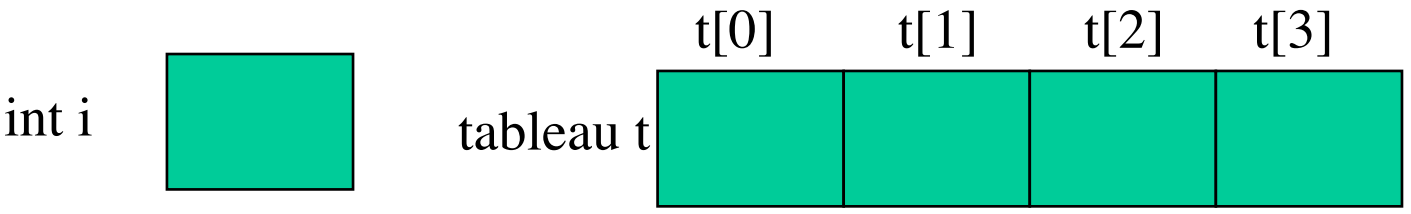
```
while (x!=y)
            {
            System.out.println("perdu");
            Lecture.readInt(x);
            }
```

Solution :

```
for (y=36 ; x!=y ; Lecture.readInt(x) )
            System.out.println("perdu");
```

Tableaux

- Une variable classique (simple ou d'instance) permet de mémoriser une seule valeur ou référence.
- Lorsque l'on veut grouper sous un même nom un ensemble de variables, on utilise un tableau.
- Chaque élément du tableau est accessible par son indice, compris entre 0 et n-1, n étant la taille du tableau



Tableaux (2)

Pour créer un tableau, on utilise l'opérateur [].

2 Syntaxes équivalentes : typeDuTableau nomDuTableau[]
typeDuTableau[] nomDuTableau

Exemple :

int tabEntier[]; ou int[] tabEntier ;

Ceci définit une référence sur un tableau d'entiers

➔ pas de place réservée pour mettre les éléments du tableau.

Allocation mémoire : l'opérateur new

int[] tabEntier = new int[7] ; // 7 éléments dans le tableau

int tabEntier[ ]= new int[7];

Les tableaux ont un indice qui commence à 0.

Initialisation d'un tableau d'un type primitif : int[] table_entier = {1,5,9};

(Ou évidemment avec une boucle for)

Héritage, Polymorphisme

- Une classe peut servir de modèle à d'autres classes, par le mécanisme d'héritage
- On part du général (classe mère) vers le particulier (classe fille)
- Tout ce qui est déclaré et défini dans la classe mère est présent dans la classe fille
- Mais on peut faire des ajouts et des modifications dans la classe fille

Classe mère, classe fille

- Attention, le vocabulaire « mère » et « fille » est relatif à une relation d'héritage, et non une propriété figée
- Par exemple, la classe B peut être la classe mère de la classe C, et en même temps la classe fille de la classe A
- Vocabulaire : classe mère, super-classe, classe fille, sous-classe, classe dérivée.

Relation d'hériatage = « est un »

- La relation d'héritage correspond à la sémantique « est un » entre la classe fille et la classe mère : toute instance de la classe fille doit pouvoir être considéré comme un élément de la classe mère
- Exemple : Voiture « est un » Véhicule, Rectangle « est une » Forme
- Contre exemple : Cercle n'est pas un Point

Polymorphisme

- La relation « est un » est à la base du polymorphisme
- Le polymorphisme est le fait de pouvoir considérer une entité selon plusieurs types : son type natif, ou n'importe lequel de ses types
- On peut voir une Voiture comme une Voiture mais aussi comme un Véhicule (si Voiture extends Véhicule)

Classes, Instances, Références

- Le polymorphisme opère entre instances et références, à partir d'une relation d'héritage entre classes
- Une instance a TOUJOURS son type natif, quelle que soit la façon dont on y accède
- Mais des références de différents types peuvent pointer vers un objet d'un type donné.

Classes, Instances, Références

- Exemple :
  - Voiture voiture = new Voiture();
  - Véhicule véhicule = voiture; // OUI
  - véhicule.avancer(); // OUI
  - véhicule.ouvrirCoffre(); // NON
  - Véhicule véhicule2=new Voiture(); // OUI

Polymorphisme : utilité ?

- Offrir des traitements génériques
- Exemple :
  - Voiture extends Véhicule
  - Camion extends Véhicule
  - public void garer(Véhicule v) permet de garer des Voitures, des Camions, etc.
  - Sans polymorphisme, il faudrait une méthode par type

Polymorphisme : restrictions

- Par polymorphisme, on ne peut accéder qu'à une partie générique de l'objet.
- Toutes les méthodes plus spécifiques, bien qu'elles existent au sein de l'objet, ne sont pas accessibles par une référence plus générique
- Lors de la définition d'un traitement, il faut donc viser juste entre trop général (classe Object) et trop spécifique…

Polymorphisme et redéfinition

- Lorsqu'une méthode est redéfinie dans une sous classe, et que l'on y accède par une référence plus générique, c'est malgré tout la méthode redéfinie qui est invoquée !
- Exemple :
  - Véhicule v = new Vélo();
  - v.tourner(); // c'est la méthode redéfinie dans Vélo qui est invoquée (par ex. tourner le guidon)

Polymorphisme : conclusion

- Concerne les références et non les objets
- Un objet possède toujours son type natif et ses méthodes éventuellement redéfinies
- Le polymorphisme permet d'accéder à différents types d'objets dans un même traitement (ex. une méthode), ou de stocker ces différents objets dans une même structure de données (ex. une liste)

Méthodes virtuelles, liage dynamique

- En raison du polymorphisme, il n'est pas toujours possible de savoir quelle méthode va être réellement appliquée à l'exécution lors de la compilation
- En effet, cela dépend du type réel de l'objet, pas du type de la référence. Ce type n'est connu que lors de l'exécution
- On parle alors de méthode virtuelle, dont le code à exectuer est trouvé dynamiquement, i.e. à l'exécution)