Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2019 – 2020
DEV1 – JAVL

DEV1 – JAVL – Laboratoires Java

TD 12 – Les structures

Dans ce TD vous trouverez une introduction aux structures.
Les codes sources et les solutions de ce TD se trouvent à l'adresse :
`https://git.esi-bru.be/dev1/labo-java/tree/master/td12-structures/`

Table des matières

Version du document
27 septembre 2019

1 Créer et manipuler des structures

À l'instar d'un tableau, une structure représente une collection d'éléments. Cependant, contrairement aux tableaux, ces données ne sont ni indexées ni nécessairement du même type. L'intérêt d'utiliser une structure est de pouvoir accéder et manipuler des données qui sont liées entre-elles de manière à former un nouveau type composite.

Dans la grammaire de Java, il n'existe pas réellement de structure. Nous allons en réalité définir des classes qui permettent d'*instancier* (créer) des objets. Ces objets sont des valeurs dont le type est celui défini par leur classe.

Avant de réellement mettre le pied à l'étrier et de vous initier à l'orienté objet [1], nous allons utiliser ces objets de manière simplifiée afin de coller aux structures.

1.1 Définition et utilisation

Imaginons vouloir utiliser un nouveau type qui permet de représenter un étudiant. Un étudiant est composé d'un matricule, d'un nom, d'un prénom.

Ce type peut-être défini et utilisé de la manière suivante :

Student.java

```java
package esi.dev1.td12;

public class Student {
   int id;
   String firstName;
   String lastName;

   public static void main(String[] args) {
      Student s = new Student();
      s.id = 101010;
      s.firstName = "Alan" ;
      s.lastName = "Turing";

      System.out.println(s.id + " : " + s.firstName + " " + s.lastName);
   }
}
```

Essayez de définir une nouvelle valeur (objet) qui vous représente en tant qu'étudiant. Affichez cette valeur.

Si vous avez bien fait l'exercice, il est probable que cela vous ait déjà semblé laborieux. Nous allons introduire un *constructeur* qui va vous permettre, en une instruction, d'instancier votre étudiant avec les valeurs souhaitées.

1. Cela sera fait dans un autre cours.

StudentWithConst.java

```java
package esi.dev1.td12;

public class StudentWithConst {
   int id;
   String firstName;
   String lastName;

   public StudentWithConst(int anId, String aFirstName, String aLastName) {
      id = anId;
      firstName = aFirstName;
      lastName = aLastName;
   }

   public static void main(String[] args) {
      StudentWithConst s = new StudentWithConst( 101010, "Alan", "Turing");

      System.out.println(s.id + " : " + s.firstName + " " + s.lastName);
   }
}
```

Instanciez à nouveau une instance d'un étudiant avec vos propres valeurs.

Le code s'améliore, mais nous pouvons aller un peu plus loin. En effet, il reste assez verbeux de représenter une valeur et de l'afficher sur la sortie standard.

Nous vous proposons d'écrire une méthode qui permet de « traduire » une instance `Student` en une chaîne de caractères. Nommons cette méthode `toString` ; la signature de la méthode sera donc `String toString(Student s)`.

StudentWithSTS.java

```java
package esi.dev1.td12;

public class StudentWithSTS {
   int id;
   String firstName;
   String lastName;

   public StudentWithSTS(int anId, String aFirstName, String aLastName) {
      id = anId;
      firstName = aFirstName;
      lastName = aLastName;
   }

   public static String toString(StudentWithSTS s) {
      return s.id + " : " + s.firstName + " " + s.lastName;
   }

   public static void main(String[] args) {
      StudentWithSTS s = new StudentWithSTS( 101010, "Alan", "Turing");

      System.out.println(toString(s));
   }
}
```

À nouveau, ajoutez une instance d'étudiant qui vous représente. Si son implémentation se complexifie un peu à chaque étape, son utilisation se voit simplifiée.

Notez qu'après création (instanciation), il est toujours possible de modifier votre instance. Modifiez son prénom en 'Adams' et son nom en 'Douglas'.

1.2 Affichage

Nous avons déjà instancié des éléments précédement. Il s'agit des scanner (`Scanner`). Pour être précis, vous manipuliez jusqu'à maintenant des instances d'objet de type `Scanner`.

Nous avons précédement dit que chaque instance est une valeur. Il serait commode de pouvoir imprimer sur la sortie standard cette valeur sans avoir à utiliser de méthodes. Si vous essayez avec une instance de `Scanner` cela va afficher sa valeur (essayez !).

```
Scanner clavier = new Scanner(System.in);
System.out.println(clavier);
```

Pour permettre cela, il faut expliciter comment une instance doit être affichée, et non plus définir une méthode qui permet d'afficher une instance passée en paramètre ; ce que nous avons fait précédemment. Modifions pour cela la méthode `toString`.

1. Actuellement vous devez fournir l'instance à représenter sous forme de chaîne de caractères. Supprimons le paramètre de la méthode `toString`. N'essayez pas de compiler, ça ne fonctionnera pas.
2. Retirons le mot clé `static`. Une méthode non statique est en fait liée à l'instance. Cela veut dire que lorsque vous utilisez une méthode non statique, celle-ci a accès aux valeurs propres à l'instance sur laquelle la méthode est appelée. Pour faire référence à cette instance, utilisez le mot clé `this`.

Cela donne le code suivante :

StudentWithTS.java

```
1  package esi.dev1.td12;
2
3  public class StudentWithTS {
4     int id;
5     String firstName;
6     String lastName;
7
8     public StudentWithTS(int anId, String aFirstName, String aLastName) {
9        id = anId;
10       firstName = aFirstName;
11       lastName = aLastName;
12    }
13
14    public String toString() {
15       return this.id + " : " + this.firstName + " " + this.lastName;
16    }
17
18    public static void main(String[] args) {
19       StudentCleared s = new StudentCleared( 101010, "Adams", "Douglas");
20
21       System.out.println(s.toString());
22    }
23 }
```

C'est mieux... enfin, pas vraiment. Heureusement, il ne faut plus rien faire pour simplifier notre code. En effet, l'appel à la méthode `.toString()` est facultatif. On peut donc écrire ce que l'on souhaitait depuis le départ. C'est-à-dire, `System.out.println(s);`. Essayez ! Tant qu'on est à parler de choses facultative. Le mot clé `this` n'est pas obligatoire lui non plus.

Le code nettoyé donne :

StudentCleared.java

```java
package esi.dev1.td12;

public class StudentCleared {
   int id;
   String firstName;
   String lastName;

   public StudentCleared(int anId, String aFirstName, String aLastName) {
      id = anId;
      firstName = aFirstName;
      lastName = aLastName;
   }

   public String toString() {
      return id + " : " + firstName + " " + lastName;
   }

   public static void main(String[] args) {
      StudentCleared s = new StudentCleared( 101010, "Adams", "Douglas");

      System.out.println(s);
   }
}
```

Maintenant que vous savez comment manipuler les « structures ». Faites des exercices.

2 Exercices : créer et manipuler des structures

Pour tous les exercices qui vous sont proposés ici, aucune méthode principale (`main`) ne doit être implémentée. Il est par contre requis de :

1. tester le bon fonctionnement de votre code ;
2. penser à la modularité et à la clarté de celui-ci ;
3. pour chacune des structures, permettre de représenter fidèlement un élément par une chaine de caractères.

Dans ce tp, chaque structure (objet) est associée à une classe dite *helper* ou *utilitaire*. Celle-ci a pour objectif de fournir certaines fonctionnalités à la structure. **Par convention**, une telle classe porte le même nom que la classe associée à ceci près qu'un 's' est ajouté à la fin du nom.

Exercice 1 Moment

▷ Créer une nouvelle structure `Moment` qui définit un moment d'une journée par un nombre de secondes, minutes et heures.

▷ Dans une classe `Moments`, créer une méthode `void addASecond(Moment m)` qui va permettre d'incrémenter d'une seconde, un moment donné en paramètre [2]

Exercice 2 Point

▷ Écrivez une structure `Point` qui permet de représenter un point par ses coordonnées x et y.

2. Tout type que vous définirez sera un type référence (à l'inverse des types primitifs). Cela implique qu'il est possible de modifier une valeur en la passant en paramètre d'une méthode. Pour plus d'explication, demandez à votre enseignant.

- Écrivez une classe `Points` dans laquelle deux méthodes doivent être implémentées :
  - La méthode `double distance(Point p1, Point p2)` qui retourne la distance entre deux points. Pour rappel, la distance entre deux points $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$ est donnée par la formule suivante :

$$\text{distance}(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

  - La méthode `Point middlePoint(Point p1, Point p2)` qui retourne le point à mi-chemin entre les deux points $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$.

Exercice 3 Circle

- Écrivez une structure `Circle` qui permet de représenter un cercle par son centre et son rayon.
- Écrivez une classe `Circles` et implémentez les méthodes suivantes :
  - `double area(Circle c)` qui retourne l'aire délimitée par le cercle passé en paramètre.
  - `double perimeter(Circle c)` qui retourne le périmètre du cercle.
  - `Circle middleCircle(Circle c1, Circle c2)` qui retourne le cercle dont le diamètre est le segment reliant les centres des cercles donnés en paramètre.
  - `boolean isADiskMember(Circle c, Point p)` qui indique si le point p est un point du disque délimité par le cercle donné.
  - `boolean haveIntersectionPoint(Circle c1, Circle c2)` qui retourne vrai si les deux cercles ont au moins un point d'intersection, sinon faux.

Exercice 4 Date

- Écrivez une structure `Date` qui permet de représenter une date par le jour, le mois et l'année.
- Écrivez dans une classe `Dates` la méthode `int compare(Date d1, Date d2)` qui retourne un nombre positif si `d1` est une date ultérieure à `d2`, négatif si `d1` est antérieur et nul si les deux dates sont les mêmes.

Exercice 5 Holiday

- Écrivez une structure qui permet de représenter un jour férié par son libellé (exemple : `"jour de l'an"`) et par sa date.
- Écrivez dans une classe `Holidays`, la méthode `void sort(Holiday[] holidays)` qui permet d'ordonner, par date, les jours fériés contenus dans un tableau.