

DEV1 – Laboratoires Java I**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

1	Créer et manipuler des structures	2
1.1	Définition et utilisation	2
1.2	Affichage	3
2	Exercices : créer et manipuler des structures	5

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 pas indexées et ne sont pas 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 la classe qui permet leur instantiation. Avant de réellement mettre le pied à l'étrier et de vous initier réellement à l'orienté objet ¹, nous allons utiliser ces objets de manière simplifiée afin de coller aux structures et de vous permettre de vous familiariser avec l'utilisation et la création de nouveaux types.

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 :

```
1 package esi.dev1.td12;
2
3 public class Student {
4     int id;
5     String firstName;
6     String lastName;
7
8     public static void main(String[] args) {
9         Student s = new Student();
10        s.id = 101010;
11        s.firstName = "Alan" ;
12        s.lastName = "Turing";
13
14        System.out.println(s.id + " : " + s.firstName + " " + s.lastName);
15    }
16 }
```

Student.java

Essayez de définir un nouvel élément (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 bonnes valeurs.

```
1 package esi.dev1.td12;
2
3 public class StudentWithConst {
4     int id;
5     String firstName;
6     String lastName;
7
8     public StudentWithConst(int anId, String aFirstName, String aLastName) {
9         id = anId;
10        firstName = aFirstName;
11        lastName = aLastName;
12    }
13 }
```

1. Cela sera fait dans un autre cours.

```

14 public static void main(String[] args) {
15     StudentWithConst s = new StudentWithConst( 101010, "Alan", "Turing");
16
17     System.out.println(s.id + " : " + s.firstName + " " + s.lastName);
18 }
19 }

```

StudentWithConst.java

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

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

Un bon étudiant n'ayant pas fait de java avant d'avoir commencé ce cours proposera certainement d'écrire une méthode pour permettre cette affichage. Nous allons nommer cette méthode `toString`.

```

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

```

StudentWithSTS.java

Encodez à nouveau votre étudiant. Si son implémentation se complexifie un peu à chaque étape, son utilisation est elle 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

Dans la manière d'interagir avec ce nouvel élément, vous avez peut-être remarqué que nous avons déjà manipulé quelque chose de semblable précédemment. Il s'agit des `Scanners`. Vous étiez en réalité en train de manipuler des instances d'objet de type `Scanner`.

Nous avons précédemment dit que chaque instance est une valeur. Il serait commode de pouvoir imprimer sur la sortie standard cette valeur sans avoir à la manipuler. D'ailleurs, 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 externe qui permet d'afficher une instance passée en paramètre; ce

que nous avons fait précédemment. Modifions pour cela la méthode `toString`. Quand vous l'utilisez, vous devez fournir l'instance à représenter sous forme de chaîne de caractères. Supprimons le paramètre. Il n'y a maintenant plus de valeur à afficher. Pour rappel, on souhaitait faire en sorte qu'une instance puisse être affichée. Cela est possible en retirant le mot clé `static`. Une instance de la classe que vous définissez porte toujours le même nom `this`. Cela donne le code suivante :

```
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 }
```

StudentWithTS.java

N'est-ce pas beaucoup mieux ? En fait, pas réellement. Au lieu de demander à afficher `toString(s)` il faut maintenant faire `s.toString()`. En fait, nous ne l'avons pas montré, mais la partie `.toString()` est facultative. 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.

Le code nettoyé donne :

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

StudentCleared.java

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 :

1. de tester le bon fonctionnement de votre code ;
2. penser à la modularité et à la clareté de celui-ci ;
3. pour chacune des structures, permettre de la représenter par une chaîne de caractère est représentée fidèlement un élément de la structure.

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 paramètre²

Exercice 2 Point

- ▷ Écrivez une structure qui permet de représenter un point par ses coordonnées x et y .
- ▷ Écrivez une classe **Points** 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 :

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

- ▷ 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 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 du 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.

2. Tout type que vous définirez sera un type référence. 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.

- ▷ `boolean haveIntersectionPoint(Circle c1, Circle c2)` qui retourne vrai si les deux cercles ont au moins un points d'intersection, sinon faux.

Exercice 4 **Date**

- ▷ Écrivez une structure 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.