TP 1 — Classes et objets

Introduction

Placez-vous dans votre répertoire `Documents▸BUT_INFO`, et créez les sous-dossiers `R201_POO▸TP0`. Dans `Apache Netbeans`, créez un projet nommé `r201_tp1_<nomDeFamille>`.

Travail à rendre

Déposer sur EPREL le répertoire de votre projet, placé dans une archive ZIP nommée `r201_tp1_<nomDeFamille>.zip`

Tous les codes sources en Java ont l'extension `.java`. Ils doivent contenir une classe (et une seule !) dont le nom commence par une majuscule est doit être le même que le nom du fichier source.

Comme en langage C, tout programme Java exécutable doit contenir une méthode nommée `main`. Cette méthode doit être contenue dans une classe. Il est d'usage, comme en langage C, que la classe qui contient la méthode `main` ne contiennt que cette méthode.

Dans votre projet, créez le fichier `HelloWorld.java` :

HelloWorld.java

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Le fichier `HelloWorld.java` forme donc la classe principale. Lorsque l'on compile, cela a pour effet de créer le fichier `HelloWorld.class`. Il contient *du bytecode*, qui est une sorte de code intermédiaire entre du code machine et le code source. Ce bytecode est lu et compris par la machine virtuelle Java, dans laquelle le code s'exécute.

1 La classe `Point2D`

On se propose, dans ce TP, de créer un programme qui gère et effectue des opérations élémentaires sur les points du plan (distance entre deux points, translation, etc.). Chaque point est caractérisé par ses deux coordonnées : son abscisse et son ordonnée.
On considère le code source ci-dessous, à copier et à intégrer à votre projet, avec le nom `Point2D.java` :

Point2D.java

```java
import java.lang.Math;
/* Classe Point2D ****************************************/
class Point2D {
// Attributs
    double x;
    double y;

    // Constructeur trivial, x et y sont initialisés à 0
    Point2D(){
    }
    // Un autre constructeur avec initialisation des coordonnées
    Point2D(double x, double y) {
        this.x = x;
        this.y = y;
    }
    // Un troisième constructeur avec un autre point
    Point2D(Point2D p) {
        this.x = p.x;
        this.y = p.y;
    }
    // Affichage
    public String toString() {
        return "(" + this.x + "," + this.y + ")";
    }
}
```

Exercice 1. **1)** Depuis `Apache Netbeans`, renommez le fichier `HelloWorld.java` en `TP1.java`.

2) Depuis le fichier `TP1.java` à l'intérieur de la méthode `main`, créez un nouveau point avec l'instruction `Point2D p0 = new Point2D();`.

3) Quel constructeur va être appelé ? Quelles seront les coordonnées du point ?

4) Créer un point `p1` de coordonnées $(1, 2)$.

5) Ajouter la méthode ci-dessous à la classe `Point2D` :

Point2D.java

```java
void translate(double dx, double dy) {
    this.x += dx;
    this.y += dy;
}
```

Appliquer cette méthode aux points précédemment créés. Que fait-elle ?

6) Programmer une méthode `distance(Point2D p)` qui reçoit un point `Point2D p` en paramètre et calcule la distance entre ce point et l'instance du point qui exécute la méthode.

7) Depuis la méthode `main`, calculer la distance entre les points `p0` et `p1`.

8) Créez une méthode `boolean estEgal(Point2D p)` qui renvoie vrai si le point appelant et le paramètre ont les mêmes coordonnées, et faux sinon.

Classes, objets et instances

- Les différents points qui ont été créés sont de *classe* `Point2D`.
- On dit que ce sont des *instances* de la classe `Point2D`.
- Les méthodes définies dans le fichier `Point2D.java` sont *appelées par un objet* et peuvent en *modifier les attributs*[a].

a. Ce qui revient en fait à *agir* sur l'objet.

Visibilité des attributs et des méthodes

Dans l'exercice, les visibilités des attributs et des méthodes ont pu être omises. On rappelle que, par principe :

- Les attributs doivent avoir la visibilité **`private`**, et ne peuvent être modifiés que depuis *l'intérieur* de la classe.
- Les méthodes sont en visiblité **`public`** pour pouvoir être appelées depuis l'extérieur de la classe.

Exercice 2. **1)** Modifier le code de la classe `Point2D` pour affecter aux attributs et méthodes la bonne visibilité.

2) Ajouter l'intruction `System.out.println(p0.x);` dans le programme principal. Que remarque-t-on ?

Cette erreur est normale et même *souhaitable* : on ne peut pas lire, et encore moins modifier, directement les valeurs des attributs avec la visibilité **`private`**. Pour contourner cela, on ajoute des méthodes permettant d'accéder et de modifier les attributs :

Point 2D.java

```
public getx() { // Getter
    return this.x;
}
public setx(double x_) { // Setter
    this.x = x_;
}
```

3) Ajouter ce code à la classe `Point2D`.

4) Ajouter maintenant le *getter* et le *setter* pour l'attribut `y`.

5) Quel est le rôle du mot-clé **`this`** dans le code source ?

6) Modifier le programme principal pour accéder aux attributs et les modifier en utilisant uniquement les *getters* et *setters*.

2 La classe `Rectangles`

À présent, on souhaite définir une nouvelle classe, la classe `Rectangle`. Cette classe doit permettre de représenter informatiquement des rectangles dans le plan. Dans un premier temps, on suppose pour simplifier que les côtés des rectangles sont parallèles aux axes. Un rectangle est donc complètement déterminé par le point (objet de classe `Point2D`) en bas et à gauche et par ses longueurs

horizontale et verticale (deux valeurs réelles de type `double`).

Les questions suivantes doivent vous permettre de définir une classe `Rectangle` afin de pouvoir manipuler de tels objets. On donne ci-dessous le squelette de la classe :

Rectangle.java

```
class Rectangle {
    // Attributs

    // Constructeurs

    // Getters, setters


    // Méthodes
}
```

Exercice 3 (Création de la classe `Rectangle` et opérations élémentaires). **1)** Dans le code fourni, ajouter la visibilité de la classe `Rectangle`, acuellement non précisée.

2) Définir les attributs de la classe `Rectangle`. Définir 3 constructeurs prenant respectivemeent en paramètre un point et deux longueurs, deux points (supposés être diagonalement opposés), ou un rectangle.

3) Écrire une méthode translate qui déplace un rectangle en déplaçant le point en bas et à gauche.

4) Ajouter tous les *getters* et *setters* nécessaires.

5) Écrire une méthode `sameAs` qui teste l'égalité de deux rectangles.

6) Écrire une méthode `translate` qui reçoit deux nombres réels en paramètre (un pour le déplacement en abscisse, l'autre pour le déplacement en ordonnées) et déplace un rectangle en déplaçant le point en bas à gauche.

On souhaite maintenant pouvoir *compter* le nombre de rectangles qui ont été instanciés. Cette information n'est pas relative à un rectangle particulier, donc ne peut pas être l'attribut d'un objet de classe `Rectangle` C'est au contraire une information relative à la *classe* elle-même : elle doit être stockée dans un *attribut de classe*. En Java, les attributs de classe sont spéficiés avec le mot-clé `static`. Ajouter la signe suivante à votre code `Rectangle.java` :

Rectangle.java

```
private static int nbRectangle = 0;
```

7) Modifier les constructeurs pour incrémener le compteur à chaque nouvelle fois qu'un nouveau rectangle est insancié.

8) Ajouter le *getter* associé.

Exercice 4 (Tests d'appartenance). **1)** Écrire une méthode `contains` qui teste si un point donné (en paramètre) est à l'intérieur du rectangle.

2) Écrire une autre méthode `contains` qui teste si un rectangle donné en paramètre est à l'intérieur du rectangle qui appelle la méthode.

3) Écrire une méthode statique nommée `hull` qui calcule le rectangle englobant d'un ensemble de rectangles. Cette méthode prend en paramètre un tableau de rectangles et retourne le plus petit rectangle qui contient tous les rectangles du tableau.

Les tableaux en Java

Nous étudierons plus tard les tableaux et les familles d'objets itérables [a]. Pour créer un tableau de rectangle, on pourra s'inspirer du code suivant :

Java

```java
// Déclaration d'un tableau (vide) recevant des objets de classe Rectangle
Rectangle liste[];
int size = 10;
liste = new Rectangle[size]; // Instanciation du tableau en mémoire
// Remplissage du tableau par des objets de classe Rectangle
for (int i = 0; i < size; i++) {
    liste[i] = new Rectangle(...);
}
```

a. Appelés *collections*.

3 Rectangles obliques et héritage

On considère des rectangles obliques donc les côtés ne sont plus nécessairement parallèles aux axes. Un tel rectangle oblique est vu comme un rectangle aux côtés parallèles qui aurait été incliné d'un certain angle θ par rapport à l'horizontale.

En d'un utilisant l'héritage, on définit le squelette la classe `RectangleOblique` permettant de manipuler de tels objets de la manière suivante :

RectangleOblique.java

```java
public class RectangleOblique extends Rectangle {
    // Attributs

    // Constructeurs
    RectangleOblique(/* … */) {
        super(/* … */);
        /* … */
    }

    // Méthodes
}
```

Exercice 5. **1)** Définir le ou les attributs nécessaires.

Visibilité des attributs et héritage

La visibilité étant `private` dans la classse `Rectangle.java`, on ne peut pas manipuler directement les attributs définis dans la classe `Rectangle` pour les objets de classe `RectangleOblique`.

Dans le code `Rectangle.java`, changer la visibilité les attributs qui le nécessitent, en remplaçant `private` par `protected`.

2) Définir des constructeurs appropriés.

3) Définir une méthode rotate dans l'esprit de la méthode translate.

4) De quelles méthodes hérite la classe `RectangleOblique` ?

5) Redéfinir celles qui le nécessitent.

6) Depuis la méthode `main`, créer des instances de `RectangleOblique` pour tester votre code.

À retenir

- Les noms de classes commencent par une *majuscule*, et les constructeurs portent le même nom que la classe.
- Par principe, les attributs sont en visibilité `private` s'il n'y a pas d'héritage. Ils sont en visibilité `protected` si les classes filles doivent hériter des attributs.
- Les méthodes sont en visibilité `public`, sauf celles qui ne réalisent que des opérations *internes* à l'objet, qui ne peuvent pas être utilisées depuis l'extérieur.
- On lit les attributs *via* des méthodes appélées *getters*. On les modifie *via* les *setters*.
- Les attributs et méthodes de classe (par opposition aux attributs *d'instance*) sont indiqués avec le mot-clé `static`.
- À l'intérieur d'une même classe, il est possible que plusieurs méthodes aient le même nom, à condition qu'elles aient des paramètres de types différents.
- En particulier, il est possible de créer plusieurs constructeurs, pour pouvoir instancier les objets de différentes manières.