Labo POO avec Java *TP4*

Points et droites en 2 dimensions

Maxime Burri - Stéphane Malandain – Michael Polla – 2016/2017 – filière ITI

Objectifs

Ce laboratoire répond aux objectifs suivants:

1) Création d'une classe
2) Instanciation d'un objet et référencement
3) Utiliser un paquetage pour l'importation
4) Notion de référence
5) Distinguer objet / référence
6) Surcharge des méthodes `equals()` et `toString()`
7) Développement complet d'une classe

Reddition

Ce laboratoire est à terminer pour le labo suivant.

I. Descriptif

Nous allons créer nos propres objets. Remarquons qu'il s'agit là d'un passage difficile car si créer un objet ne pose pas de problèmes, il faut savoir le créer de telle façon qu'il soit réutilisable. Ceci est lié à la POO qui doit toujours penser à la réutilisabilité des objets sans quoi tout serait bloqué.

II. Enoncé

Nous allons définir une classe PointD2 pour créer et utiliser des points du plan cartésien à deux dimensions.

III. Implémentations

Etape A : une classe PointD2 dans un paquetage

Nous allons créer une classe capable de générer des objets immuables : des points. Ces objets vont constituer un nouveau type pour Java. En fait un nouveau type que vous allez mettre à disposition d'autres développeurs par le biais d'un nouveau paquetage. Une bonne idée de désignation du paquetage serait : `ch.hepia.it2.geomD2`. Mais, pour des questions de simplicité, nous nommerons ce paquetage `geomD2`.

Ici, l'application sera constituée de deux fichiers.

Programme 1 Le paquetage `geomD2`

```
// PointD2.java
package geomD2;
public class PointD2 {

    // PointD2 a ses variables membres
    private Integer x = null;
    private Integer y = null;

    // PointD2 a ses constructeurs
    public PointD2() { }
    public PointD2(int a,int b) {
       x = a; // conversion implicite de int en Integer
       y = b;
    }

    // PointD2 a ses méthodes d'instance
    public boolean defini() {
        return (x != null) && (y != null);
    }
}
```

Programme 2 Utilisation du paquetage `geomD2`

```
// TP3a.java
import geomD2.PointD2;

public class TP3a {
    public static void main(String[] args) {
        PointD2 p0 = null;
        System.out.println(p0);
        PointD2 p1 = new PointD2();
        System.out.println(p1);
        System.out.println(p1.defini());
        PointD2 p2 = new PointD2(3,4);
        System.out.println(p2);
        System.out.println(p2.defini());
    }
}
```

QUESTIONS

1. Que se passe-t-il si vous omettez le modificateur public pour `PointD2` ?
2. Que cela signifie-t-il pour la visibilité (*scope*) ?
3. Pourquoi n'avons-nous pas pu écrire `p0.defini()` ?
4. Qu'affiche l'impression `System.out.println(p1)` ?
5. Comment implémenter le constructeur `PointD2(int a,int b)` tout en respectant le typage fort ?

Note : l'impression de la référence d'un objet est automatiquement faite par la méthode `toString()` de la classe `Object` de Java (nous utilisons ici une capacité de l'héritage qui sera discutée par la suite).

Etape B : ajout d'autres méthodes à la classe PointD2

Ajouter ensuite des constructeurs `PointD2(PointD2 pt)`, `PointD2(Integer[] pt)`, des accesseurs `getX()`, `getY()` et des modificateurs `setX()`, `setY()` ainsi qu'une méthode `dist()` qui calcule et retourne la distance à un autre point. Tester chacune des méthodes implémentées.

QUESTIONS

1. Faites un croquis (comme bon vous semble !) des composants du projet.
2. Si vous comparez deux variables d'instance, en quoi doivent-elles différer ?
3. Si vous comparez deux constructeurs, qu'ont-ils en commun ?

Etape C : la méthode `toString()` pour la classe `PointD2`

QUESTIONS

1. Que produit l'appel `System.out.println(p.toString())` ?
2. A quel est emplacement dans la hiérarchie des classes, la méthode `toString()` est-elle déclarée ?
3. Fort de votre compréhension des questions précédentes, implémenter `toString()` pour la classe PointD2.

Etape D : la méthode `equals()` pour la classe `PointD2`

La méthode `equals()` est définie dans la classe `Object`. Consulter sa spécification dans l'API et la redéfinir au sein de la spécification de `PointD2`.

Programme 1 Expérimentez le programme

```
// TP3b.java
import geomD2.PointD2;

public class TP3b {
    public static void main(String[] args) {
        PointD2 p1 = new PointD2(3,4);
        PointD2 p2 = p1;
        System.out.println(p2.equals(p1));
        p2 = new PointD2(3,4);
        System.out.println(p2.equals(p1));
    }
}
```

Exécuter ce programme et en observer les impressions.

Programme 2 Expérimentez le programme avec les ajouts

```
PointD2 p1 = new PointD2(3,4);
PointD2 p2 = p1;
System.out.println(p2.equals(p1));
System.out.println("avec == : "+(p2 == p1));
p2 = new PointD2(3,4);
System.out.println(p2.equals(p1));
System.out.println("avec == : "+(p2 == p1));
```

Par défaut, `equals(Object o)` compare les références des objets et agit comme l'opérateur `==` ; pour tester l'égalité des contenus, `equals(Object o)` doit toujours être redéfini. Maintenant implémentez `equals()`.

QUESTIONS

1. Pourquoi `equals()` n'est pas prototypé selon `boolean equals(Object,Object)` ;
2. Que pouvez-vous dire de `equals()` (avec un argument) pour deux `PointD2` affectés de `null` ?
3. Comment modifiez-vous le corps de votre méthode afin qu'elle retourne immédiatement `false` si l'argument n'est pas un `PointD2` ? Vous pouvez utiliser l'opérateur `instanceof` pour savoir si un objet appartient à une classe.

Etape E : la classe `DroiteD2`

Voici ce qui constitue un énoncé : le fichier avec tous les prototypes que nous vous demandons d'implémenter.

Programme 3 Squelette de `DroiteD2`

```
// DroiteD2.java
public class DroiteD2 {

    private PointD2 p = null;
    private Double  m = null; // m est la pente de la droite

    // Les constructeurs :
    // un point et une pente
    public DroiteD2(PointD2 p, Double m);
    // une pente et une ordonnee a l'origine
    public DroiteD2(Double m, Double ord);
    // deux points
    public DroiteD2(PointD2 p1, PointD2 p2);

    // Les modificateurs :
    // changer le point
    public void setPoint(PointD2 p);
    // changer la pente
    public void setPente(Double m);
    // changer tout
    public void setPointPente(PointD2 p, Double m);

    // Les accesseurs purs :
    // obtenir le point
    public PointD2 getPoint();
    // obtenir la pente
    public Double getPente();

    // Les accesseurs qui effectuent un calcul
    // abscisse d'intersection avec axe X
    public Double getIntersectionX();
    // ordonnée d'intersection avec axe Y
    public Double getIntersectionY();

    public boolean contient(PointD2 p);
    public boolean estParallele(DroiteD2 d);

    // Les utilitaires :
    public boolean equals(Object o); // redéfinition
    public String toString();        // redéfinition
    public StringBuilder toString2();
}
```

QUESTIONS

1. Les objets `String` ou appartenant aux classes enveloppes des types primitifs (`Integer`, `Double`,...) sont immuables. Qu'économisons-nous en utilisant des objets mutables ?
2. Pourquoi avons-nous deux méthodes de renvoi de chaîne?
3. Pourquoi avoir appelé la deuxième méthode de renvoi de chaîne `toString2()` et non pas aussi `toString()`?