Bases de la conception orientée objet
Classes, objets, messages

Steven Costiou
Stéphane Ducasse
Inria

September 23, 2020

1 Exercice 1 : points et rectangles

Dans cet exercice, nous allons créer et afficher des rectangles sous forme d'objets. Pour cela, nous allons modéliser le concept de rectangle sous forme de classe : les variables d'instance de la classe représenteront la structure d'un rectangle et les méthodes les opérations pour calculer les propriétés du rectangle (aire, périmètre).

Un rectangle est représenté par un couple de points origin (point supérieur gauche) et corner (point inférieur droit). Les propriétés du rectangle sont ensuite calculées à partir de ces deux points.

1.1 Mise en bouche : une petite classe Point

Avant de pouvoir modéliser un rectangle, nous devons modéliser le concept de point. Programmer une classe Point dans un fichier Point.java :

1. Un point possède une variable d'instance x qui est un entier représentant sa position sur l'axe x d'un repère orthonormé,
2. une variable d'instance y qui est un entier représentant sa position sur l'axe y d'un repère orthonormé,
3. des méthodes d'accès en lecture (*getters*) et en écriture (*setters*) sur ces deux variables, appelées *accesseurs*.

Pour instancier des points, vous aurez besoin d'un constructeur :

4. Écrire un constructeur pour notre classe Point qui prend en paramètre deux entiers i et j puis les affecte aux variables d'instance x et x du point en cours d'instantiation via ses accesseurs,
5. écrire une méthode de test qui vérifie qu'un nouveau point possède bien les coordonnées passées à son constructeur.

1.2 La classe Rectangle

Nous pouvons maintenant programmer la classe Rectangle, dans un fichier Rectangle.java :

1. Un rectangle est représenté par un couple d'objets Point : origin (point supérieur gauche) et corner (point inférieur droit).

2. Programmer un constructeur paramétré par les coordonnées des 2 points origin et corner, on veut pouvoir instancier des rectangles comme suit: Rectangle r = new Rectangle(4, 2, 3, 1).

Nous allons programmer en TDD les opérations du rectangle. Cela signifie que nous allons écrire d'abord des méthodes de test qui décrivent et testent le comportement attendu. Ces tests seront en échec car le comportement décrit n'existe pas encore. Dans un second temps, nous mettrons alors en oeuvre ce comportement dans la classe Rectangle afin de se conformer aux tests.

1. Créer une classe de test, et programmer les tests suivants :

   (a) Tester que l'instanciation d'un rectangle via le constructeur de Rectangle produit bien un objet rectangle avec deux points origin et corner possédant des coordonnées correctes.

   (b) Tester les opérations du rectangle *largeur*, *longueur*, *aire*, et *périmètre*, mises en oeuvre respectivement par les méthodes width(), height(), area(), perimeter(). À chaque test, implémenter le code adéquet pour que le test passe.

2. Écrire un test pour une opération *impression*, mise en oeuvre par la méthode print(), qui imprime les propriétés du rectangle dans la console. Par exemple, pour un rectangle instancié comme Rectangle r = new Rectangle(4, 2, 3, 5), envoyer le message r.print() affiche la chaine de caractères suivantes sur la console : *Rectangle: (<4@2> , <3@5>)*. Vous utiliserez les méthodes getX() et getY() des points du rectangle pour obtenir leurs valeurs.

3. Modifier le code de print() sans toucher au test unitaire : au lieu d'utiliser getX() et getY() des points du rectangle, vous demanderez directement aux points de s'imprimer sur la console en leur envoyant le message print(). Votre modification est complète une fois que le test unitaire s'exécute avec succès.

2 Exercice 2 : formes géométriques

Nous allons maintenant abstraire pour modéliser d'autres formes géomètriques, comme des cercles, triangles, carrés.

2.1 La classe abstraite GeometricShape

Cette classe définit de manière abstraite comment les formes géométriques se structurent et se comportent. Les sous classes de GeometricShape, elles, mettent en oeuvre cette structure et ce comportement.

1. Écrire la classe abstraite GeometricShape, quels sont ses éléments structurels et comportementaux devant être partagés par toutes les formes géométriques ?
2. Modifier la classe Rectangle pour hériter de GeometricShape. Les tests doivent continuer à passer.
3. Écrire la classe Circle :
   (a) Pour chaque classe, hériter de la classe qui vous semble la plus judicieuse et proposer un constructeur,
   (b) écrire les méthodes devant être redéfinies (en TDD).

2.2 Dessin des formes géomètriques

Nous allons utiliser une classe de test PainterApp, qui utilise un objet Painter dédié à l'affichage de formes graphiques. La méthode paint(Graphics g) de Painter est appelée automatiquement lors de l'affichage. Cette méthode doit utiliser l'objet de dessin Graphics g pour afficher des objets graphiques.

Récupérer les classes PainterApp[1] et Painter[2]. Compléter la classe Painter avec les éléments suivants :

1. Écrire une méthode public void paintRectangle(Rectangle r, Graphics g)) qui dessine le rectangle r sur le médium graphique g. Vous utiliserez la méthode de dessin offerte par g pour dessiner des rectangles : drawRect (int x1, int y1, int x2, int y2).
2. Écrire une méthode public void paintCircle(Circle c, Graphics g) qui dessine le cercle c sur le médium graphique g. Vous utiliserez la méthode de dessin offerte par g pour dessiner des cercles drawOval(int centerX, int centerY, int radius, int radius).

[1] https://kloum.io/costiou/teaching/PainterApp.java
[2] https://kloum.io/costiou/teaching/Painter.java

3. Dans la méthode paint(), parcourir la liste des formes géométriques contenues dans le tableau shapes, et appeler la méthode de dessin appropriée en fonction de la classe de chaque objet. Pour cela vous pouvez utiliser le code suivant qui renvoie true si la forme géométrique s est un rectangle et false sinon : s instanceof Rectangle.

L'utilisation de instanceof n'est pas *objet*. En programmation objet, on va laisser les objets décider comment ils répondent à un message plutôt que de contrôler et sélectionner impérativement les messages qu'on leur envoie.

Nous allons introduire une méthode de dessin dans chaque objet géométrique afin de les laisser décider comment ils doivent être affichés.

4. Écrire dans la classe abstraite GeometricShape une méthode abstraite public abstract void drawOn(Graphics g).

5. Dans chaque sous classe de GeometricShape, implémentez la méthode drawOn(Graphics g) en conséquence.

6. Dans la classe Painter, supprimer toutes les instructions conditionnelles et les tests instanceOf et remplacer cela par un envoi du message drawOn(g) aux objets géométriques. Supprimer le code devenu inutile.