

Auteur : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Résumé

Le but de ce TP est de concevoir une classe en utilisant la notion d'association vue en cours. Cette classe sera également implantée en Java en utilisant une collection.

1 Objectifs

Les objectifs de ce TP sont les suivants :

- concevoir une classe à un niveau d'abstraction « élevé » en utilisant les associations ;
- représenter cette conception avec UML ;
- proposer une solution d'implantation avec UML ;
- implanter cette classe en Java en une collection ;
- utiliser un packaging ;
- écrire des tests avec JUnit.

2 Présentation informelle

Un polygone au sens géométrique est un ensemble de points du plan dont le nombre est supérieur à 3 (on considère qu'un polygone « définit » un plan). On doit pouvoir le translater, mais également récupérer son périmètre et son aire.

3 Conception de la classe Polygone

1. définir dans un premier temps la relation qui existe entre Polygone et Point. On pourra utiliser une association simple avec multiplicité, puis une relation d'agrégation ou de composition. Écrire le diagramme UML correspondant.
2. il faut maintenant réfléchir à une version plus « implantation » de la classe. En particulier, il faut déterminer les attributs de Polygone. Pour cela, réfléchir à la construction du polygone : dispose-t-on du nombre exact de points, les connaît-on etc.

Pour pouvoir gérer l'ensemble de points constituant le polygone, on utilisera une liste de points via une instance de `java.util.ArrayList<Point>`.

On choisit (arbitrairement!) d'utiliser un constructeur pour Polygone qui prend en argument un objet de type `java.util.ArrayList<Point>`.

Les services pouvant être rendus par le polygone seront représentés par les méthodes suivantes :

- deux méthodes permettant d'ajouter et de retirer un point du polygone à une certaine position ;
- une méthode permettant de récupérer le nombre de sommets du polygone ;
- une méthode `toString` retournant une représentation du polygone sous la forme « Point1 ... Pointn » ;
- une méthode de translation ;
- une méthode retournant le périmètre du polygone ;
- une méthode calculant l'aire du polygone.

Proposer un diagramme UML à un niveau plus « implantation » décrivant la classe Polygone.

3. écrire un diagramme de séquence représentant le scénario suivant :
 - le programme de test crée un point de coordonnées (0,0)
 - le programme de test crée un point de coordonnées (2,2)
 - le programme de test crée un point de coordonnées (4,5)
 - le programme de test crée une instance de `ArrayList` à partir de ces points (on ne représentera pas les ajouts des points dans la liste pour ne pas alourdir le diagramme)
 - le programme de test crée un polygone à partir de la liste
 - le programme de test translate le polygone avec un vecteur $(-2, 3)$
 - le programme de test affiche le périmètre du polygone

4 Implantation de la classe Polygone

Planter et documenter la classe Polygone en Java en utilisant au maximum les services rendus par la classe Point. Écrire en même temps une classe de test PolygoneTest qui permettra de tester les services rendus par la classe Polygone. Vous utiliserez le *framework* JUnit (cf. [1, 2, 3]). Utilisez également la documentation javadoc de la classe ArrayList fournie sur le site d'Oracle pour comprendre comment utiliser ses méthodes (en particulier cherchez le domaine de valeurs du paramètre de la méthode get : les indices commencent-ils à 0 ou à 1 ?).

Vous choisirez de créer vos classes dans un paquetage fr.supaero.figure, paquetage auquel appartiennent les classes qui vous sont fournies.

Vous pourrez également écrire les contrats nécessaires pour Polygone.

Vous réfléchirez particulièrement à la construction du polygone : doit-on copier « proprement » l'ensemble de points passés en paramètre ou doit-on utiliser une affectation par référence ?

Enfin, vous pourrez utiliser le visualisateur graphique disponible sous l'onglet « Ressources » du site. Il faut utiliser la classe Ecran qui possède des méthodes permettant de dessiner des figures. Sa documentation Javadoc vous permettra de comprendre son utilisation.

Références

- [1] JUnit. URL : <http://www.junit.org>.
- [2] K. BECK et E. GAMMA. *Test Infected : Programmers Love Writing Tests*. URL : <http://junit.sourceforge.net/doc/testinfected/testing.htm>.
- [3] K. BECK et E. GAMMA. *JUnit Cookbook*. URL : <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>.