TP 2 : Agrégation, Composition, Héritage

©2022 Ghiles Ziat
ghiles.ziat@epita.fr

Dans les exercices qui suivent, vous êtes libres de répondre aux questions en utilisant le langage de votre choix parmi Java et C++, sauf lorsqu'il est explicitement demandé d'utiliser un des deux langages ou les deux.

Exercice I : Agrégation, Composition

Q1 – Proposez une architecture de classes pour la spécification suivante : On veut manipuler des formes géométriques en deux dimensions : Triangle, Cercle, Rectangle et Polygone. Toutes nos formes auront comme point commun d'avoir un nombre de sommets et une façon de calculer leurs aires.

Q2 – Créez une classe `Point` avec deux champs `double x,y` et un constructeur avec paramètres.

Q3 – Ajoutez une méthode `Point vector(Point p)` qui retourne une nouvelle instance de point correspondant au vecteur [1] entre l'instance appelante et le point passé en paramètre. Par exemple, le vecteur entre le point $(0, 1)$ et le point $(3, 5)$ donnera le point $(3, 4)$.

Q4 – Ajoutez une méthode `Point translate(Point v)` qui retourne une nouvelle instance de point correspondant à la translation de l'instance appelante, par le vecteur passé en paramètre. On rappelle qu'une translation consiste à additionner les composantes x et les composantes y entre elles. Par exemple, la translation du point $(0, 1)$ par le vecteur $(3, 5)$ donnera le point $(3, 6)$.

Q5 – Ajoutez une méthode `Point scal(double coef)` qui retourne une nouvelle instance de point ou les composantes x et y ont été multipliées par le coefficient passé en paramètre.

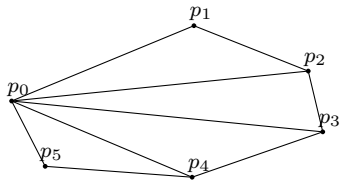
Q6 – Proposez une modélisation et créez une classe pour les `Triangle`.

Q7 – Ajouter une méthode `double area()` qui calcule l'aire du triangle. **Rappel :**

- l'aire d'un triangle (ABC) est $\frac{1}{2} \cdot |det(\overrightarrow{AB}, \overrightarrow{AC})|$ où *det* est le déterminant de deux vecteurs.
- le déterminant de deux vecteurs $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ et $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$ est donné par : $x_1y_2 - y_1x_2$

1. Un vecteur a, comme un point, des champs x et y. Pour aller vite, on ne définira pas de classe `Vector` et on réutilisera la classe `Point` pour modéliser des vecteurs.

Q8 – Créez une classe `Polygon` avec un champ `vertices` contenant les points de l'enveloppe convexe du polygone, et un champ `asTriangles` contenant un tableau/vecteur de triangles.



Q9 – Ajoutez un constructeur qui prend un tableau/vecteur de points $(p_0, p_1, p_2, p_3, ..., p_{n-1}, p_n)$ correspondants aux sommets du polygone. Le constructeur stockera de plus un tableau/vecteur de triangles recouvrant le polygone dans le champs `asTriangles`. Effectuez pour cela un parcours deux à deux des sommets pour construire la liste des triangles $(p_0, p_1, p_2), (p_0, p_2, p_3), ..., (p_0, p_{n-1}, p_n)$. On supposera que l'on dispose de plus de 2 sommets et que ceux-ci sont donnés dans le sens des aiguilles d'une montre, de sorte à pouvoir les connecter dans l'ordre dans lequel ils sont fournis.

Q10 – Ajoutez à la classe `Polygon` une méthode `double area()` qui calcule l'aire d'un polygone.

Q11 – **Bonus** Définissez une méthode `Point random()` dans la classe `Triangle` qui tire un point aléatoirement dans un triangle. Une façon [2] de faire est de tirer des coefficients aléatoires r_1 et r_2 puis,

- si $r_1 + r_2 < 1$ d'effectuer translation du point A par les vecteurs $r_1 \cdot \overrightarrow{AB}$ et $r_2 \cdot \overrightarrow{AC}$
- sinon, d'effectuer translation du point A par les vecteurs $(1 - r_1) \cdot \overrightarrow{AB}$ et $(1 - r_2) \cdot \overrightarrow{AC}$

Q12 – **Bonus** On désire maintenant tirer aléatoirement et uniformément un point dans un polygone. Une façon de faire est d'abord de tirer aléatoirement un triangle, puis de tirer un point dans ce triangle. On fera attention à l'uniformité : les triangles ayant la plus grande aire ont plus de chances d'être tirés. Définissez la méthode `Point random()` qui tire un point aléatoirement dans un polygone.

Q13 – **Bonus** On vous donne en annexe un script bash *show.sh* qui va vous permettre de tester visuellement l'uniformité de votre tirage : dans votre `main`, affichez sur la sortie standard 1000 points tirés aléatoirement au format "x y" et affichez sur la sortie d'erreur les sommets du polygone. Vous pouvez à présent lancer votre programme en préfixant la commande par *./show.sh* comme suit :

`./show.sh java Main` (ou l'équivalent *c++* : `./show.sh ./a.out`)

Assurez vous d'avoir *gnuplot* installé sur votre machine et d'avoir donné les droits d'exécution au script au préalable (en lançant `chmod +x show.sh`)

Exercice II : Héritage

Q1 – Proposez une architecture de classes pour la spécification suivante : On veut manipuler une structure générale de gestion de stock, avec une capacité maximum, un nombre de produits stockés, des méthodes pour dire si la structure est vide ou pleine, et des méthodes d'ajout et de retrait de produit. On veut de plus avoir deux structures plus spécifiques, une structure FIFO (first-in first-out)

2. Pour les curieux, une interprétation géométrique de cette formule est donnée en annexe.

et une structure LIFO (last-in first-out) avec des méthodes de retrait différentes, mais avec la même méthode d'ajout.

Q2 – Définissez une classe abstraite `Storage` dans laquelle on utilisera un tableau `memory` pour stocker des entiers, un entier qui comptera le nombre courant d'éléments stockés dans le tableau, ainsi qu'un entier qui correspondra à l'indice de la prochaine valeur à écrire.

Q3 – Ajoutez un constructeur qui prend en paramètre une taille maximum, ainsi que des méthode `boolean isFull()` et `boolean isEmpty()` qui vérifient respectivement si le tableau est plein ou vide.

Q4 – Ajoutez à la classe une méthode `void add(int i)` qui ajoute un entier à la mémoire. L'ajout d'élément se fera de façon cyclique dans le tableau : on ramènera l'indice d'ajout du prochain élément dans l'intervalle $[0; CAPACITY]$ à l'aide de l'opérateur %. Si le tableau est plein, la méthode `add` ne fera rien.

Q5 – Nous allons maintenant créer deux sous-classes qui hériteront de `Storage` et qui implémenteront respectivement des structures FIFO (first-in first-out) et LIFO (last-in last-out).

Annexes

Script d'affichage

Copiez-collez le code suivant dans un fichier *show.sh*

```
#!/bin/bash
# we run the arguents of the script as a standalone command
$@ > pts.txt 2> vert.txt
# we rewrite the first vertices at the end of the file to "close" the
# drawing of polygon
echo $(head -n 1 vert.txt) >> vert.txt
# we ask gnuplot to draw the border of the polygon and to plot the dots
gnuplot -persist -e "reset; unset key; set terminal pngcairo size 500,400;\
set output 'uniform.png'; set style line 1 linetype 1 pointtype 4;\
plot 'pts.txt' with points pointtype 3, 'vert.txt' with linespoints linestyle 1"
# we remove the files we have generated and open the image
rm vert.txt pts.txt; open uniform.png
```

Tirage aléatoire dans un triangle

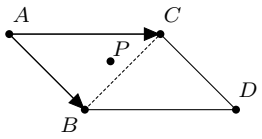
Pour tirer un point aléatoirement dans un triangle ABC on va construire un parallélogramme $ABDC$ (ou D est le symétrique de A par rapport au milieu M de BC), tirer un point aléatoirement dans $ABDC$ et ramener ce point dans ABC. Tirons à présent un point P aléatoirement dans $ABDC$:

$$P = r_1 \cdot \overrightarrow{AB} + r_2 \cdot \overrightarrow{AC}$$

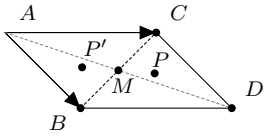
ou r_1 et r_2 sont deux coefficients aléatoires entre 0 et 1.

On a ensuite deux possibilités :

1. Si P est dans ABC alors on le garde
2. s'il est dans BCD, on retourne son symétrique par rapport à M



1) On garde P



2) On prend son symétrique par rapport à M

On peut finalement remarquer que si $r_1 + r_2 \leq 1$, alors P sera forcément dans ABC, et sinon P est dans BCD.