Programmation Orientée Objets

TD-TP n°4 : Polymorphisme

Exercice 1 : Diagramme de classes

Compléter le diagramme de classes modélisant l'intégralité des données du sujet de TP, incluant le TP n°4.

Exercice 2 : Surface de polygones (fin)

Ajouter une méthode `int signeAngle(Point b, Point c)` à la classe `Point`, qui retourne le signe de l'angle formé par les 3 points, donné par le déterminant des vecteurs `ab` et `ac` (`a` étant le point courant) : $(x_b-x_a)*(y_c-y_a)-(x_c-x_a)*(y_b-y_a)$. Au lieu de (-1, 0, 1), définir des constantes statiques entières : `SAM`, `ALIGNES`, `SIAM` (`SAM` = « Sens des Aiguilles d'une Montre »).

Écrire une méthode `boolean estConvexe()` qui retourne `true` si le polygone est strictement convexe. Utiliser soit `signeAngle()`, soit la somme des angles du polygone ($\Sigma = (n-2)*\Pi$), soit la Marche de Jarvis.

Modifier `surface()` de `Polygone` : retourner -1 si le polygone n'est pas convexe.

Exercice 3 : Autres méthodes utiles

Écrire une méthode `boolean dedans(Point p)` qui vérifie si `p` est strictement inclus dans le polygone (convexe) courant.

Écrire une méthode `boolean plusGrand(Polygone poly)` qui retourne `true` si le polygone convexe courant a une surface strictement plus grande que `poly`.

Ajouter l'affichage de la surface à la méthode `toString()`.

Exercice 4 : Polymorphisme : Triangle, Rectangle et Polygone

Ajouter à la méthode `surface()` de chaque type de polygone un affichage spécifique, par exemple : « Calcul de la surface d'un triangle ».

Surclasser un `Triangle` en `Polygone` et vérifier le comportement polymorphe de `surface()` : quelle méthode est appelée ? Vérifier le comportement polymorphe de `plusGrand(Polygone poly)` en comparant la surface d'un triangle et celle d'un rectangle.

Exercice 5 : Polymorphisme : classe « EnsemblePolygoneAlea »

Écrire la classe `EnsemblePolygoneAlea` avec un constructeur qui créé `nbPoly` polygones aléatoires, dont le type est choisi aléatoirement parmi `Polygone` et ses sous-classes. Dans le cas de `Polygone`, le nombre de côtés est choisi aléatoirement entre 5 et 20.

Dans le cas de `Rectangle`, on pourra réarranger *a posteriori* les sommets tirés aléatoirement afin que les 4 sommets définissent un rectangle avec les sommets 0 et 2 opposés (haut gauche et bas droite).

Écrire la méthode `sommeSurface()` qui retourne la somme des surfaces des polygones de `EnsemblePolygone` (seulement des polygones convexes).

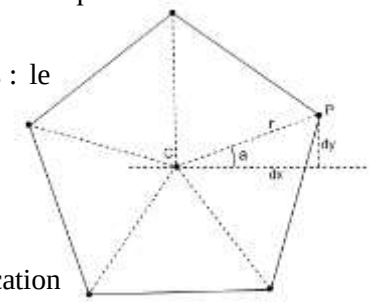
Vérifier le comportement polymorphe des méthodes `sommeSurface()` et `affiche()` avec `EnsemblePolygoneAlea`, puis de la méthode `plusGrand(Polygone poly)` en comparant la surface de polygones de l'ensemble 2 à 2 sans connaître leur type effectif.

(facultatif) Compléter l'exercice 3 du TD3 en modifiant le constructeur de `Polygone` afin de créer uniquement des polygones aléatoires convexes ayant un périmètre >= 600.

Exercice 6 : Méthode equals()

Redéfinir la méthode `equals()` dans `Polygone`. On examinera les polygones indépendamment de leur position dans l'espace et de la rotation des sommets dans le tableau, mais pas de leur rotation dans l'espace.

Exercice 7 : Classe « PolygoneRegulier » (facultatif)



Écrire la classe `PolygoneRegulier` comportant un constructeur avec 3 paramètres : le nombre n de sommets, le centre $c = (x_c, y_c)$ et un des sommets $p = (x_p, y_p)$.

Les n sommets du polygone sont alors les points (x_i, y_i) définis par :

$x_i = x_c + r \times \cos(a+i\times 2\times\pi/n)$ avec : $dx = x_p - x_c$, $dy = y_p - y_c$

$y_i = y_c + r \times \sin(a+i\times 2\times\pi/n)$ $r = \text{sqrt}(dx^2+dy^2)$

$a = \text{atan2}(dy, dx)$

Vérifier que la classe `PolygoneRegulier` fonctionne correctement et sans modification avec la « vieille » méthode `sommeSurface()` même sans recompilation de `EnsemblePolygone`.

Modifier `EnsemblePolygoneAlea` afin de permettre la génération aléatoire de polygones réguliers (avec n, c et p aléatoires).