amU Faculté des sciences Aix Marseille Université

Année 2025-26

Domaine Sciences et Technologies
Licence Informatique : math-info
Initiation au Génie logiciel : TP 7
Codes UE : SIN5U34
TP 7 : polygones réguliers (à rendre)

Consignes

Vous devez rendre les fichiers :

- `AngleConversion.java`
- `Point.java`
- `Circle.java`
- `RegularPolygon`

sur Ametice.

Les fichiers doivent être déposés tel quel. Le travail est individuel et donc si plusieurs rendus sont totalement identiques et identifiables comme tels la note sera divisée par autant de copie. La note de ce TP comptera pour 2 points (10%) de la note finale.

Classe `AngleConversion`

Cette classe est une peu particulière car c'est une classe utilitaire qui donnera accès à deux méthodes de classes :

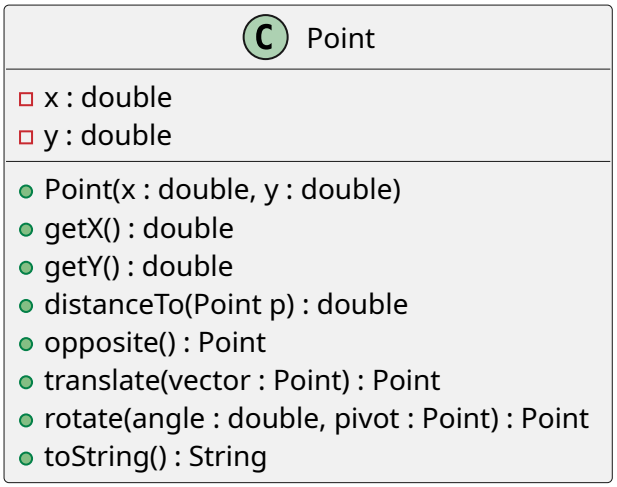
- `double toRadians(double angleInDegrees)` qui convertit un angle exprimé en degré en le même angle exprimé en radians.
- `double toDegrees(double angleInRadians)` qui convertit un angle exprimé en degré en le même angle exprimé en radians.

Vous rajouterez aussi du code pour qu'il soit impossible de construire une instance de la classe `AngleConversion` à l'extérieur de la classe.

Tâche 1 : Écrivez le code de la classe `AngleConversion`.

Classe `Point`

On considère la classe `Point` définie par le diagramme suivant :



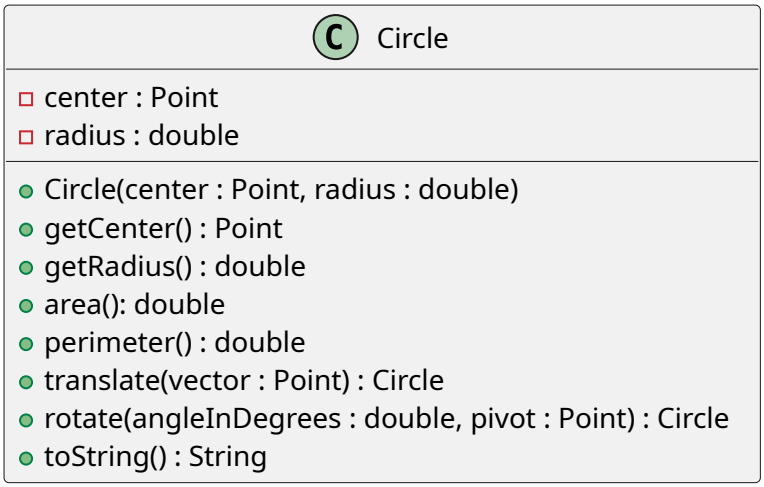
Cette classe est très similaires à la classe `Point` que vous avez déjà codé. Il y a néanmoins des différences.

- La méthode `toString()` devra renvoyer la chaîne de caractères `(0.0,0.0)` pour un point $(0, 0)$.
- La méthode `translate` renvoie un nouveau point qui est issu de la translation par le vecteur (dx, dy) (représenté par le `Point vector` passé en paramètre) du point initial avec lequel la méthode est appelée.
- La méthode `opposite()` qui, appelée avec un point (x, y), renvoie un nouveau point $(-x, -y)$.
- La méthode `rotate` renvoie un nouveau point qui est issu de la rotation d'angle `angleInDegrees` (angle exprimé en degré) et de centre `pivot` du point initial avec lequel la méthode est appelée. Pour réaliser une rotation d'`angle` de centre `pivot` (x, y), il faudra appliquer au point les trois transformations suivantes dans cet ordre :
  - une translation de vecteur $(-x, -y)$,
  - une rotation d'`angle` ayant comme pivot l'origine (l'image d'un point de coordonnées (x, y) par une rotation d'angle θ ayant comme pivot l'origine est le point de coordonnées $(x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$)
  - une translation de vecteur (x, y).

Tâche 2 : Écrivez le code de la classe `Point`.

Classe `Circle`

On considère la classe `Circle` définie par le diagramme suivant :



— La méthode `toString()` devra renvoyer la chaîne de caractères `Circle{center=(0.0,0.0), radius=1.0}` pour un cercle de centre `(0.0,0.0)` et de rayon `1.0`.
— La méthode `translate` renvoie un nouveau cercle qui est issu de la translation par le vecteur (dx, dy) du cercle initial avec lequel la méthode est appelée.
— La méthode `rotate` renvoie un nouveau cercle qui est issu de la rotation d'angle `angle` et de center `center` du cercle initial avec lequel la méthode est appelée.

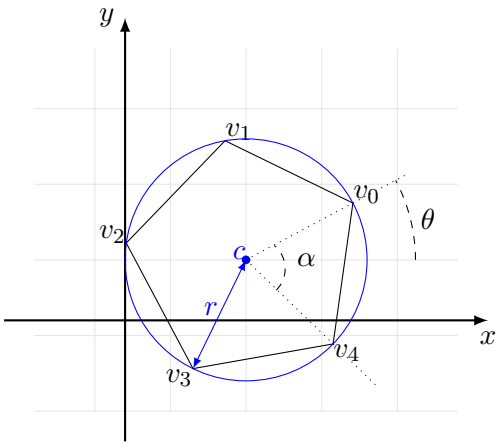
Tâche 3 : Écrivez le code de la classe `Circle`.

Classe `RegularPolygon`

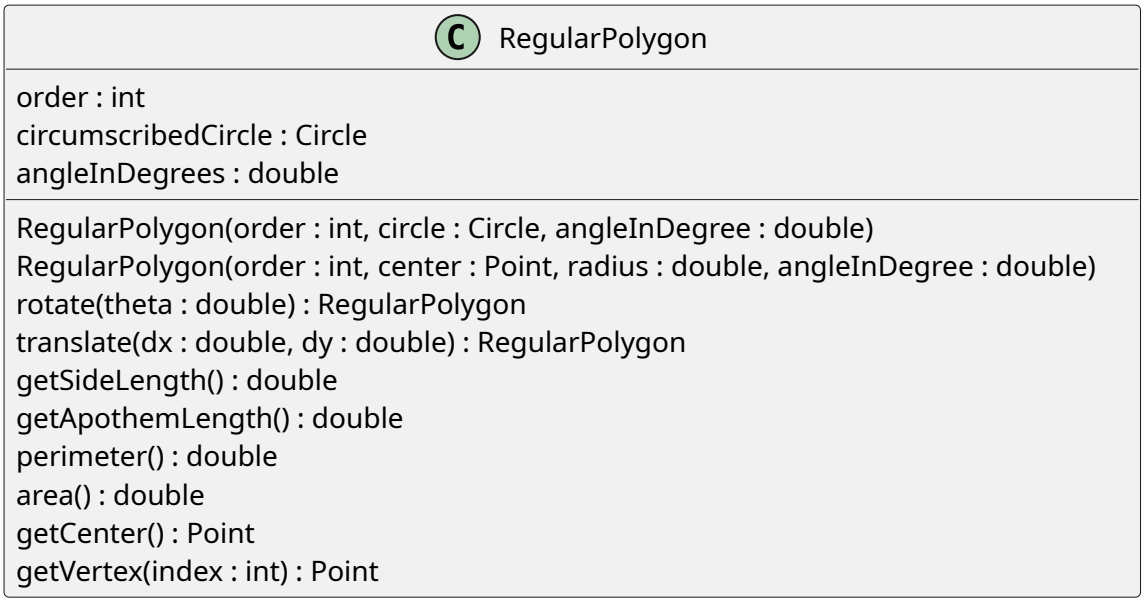
On souhaite écrire une classe permettant de représenter les polygones réguliers. Un polygone régulier est défini par :

— le nombre n de côtés ou de sommets (l'*ordre* du polygone),
— son centre c,
— le rayon r de son cercle circonscrit,
— l'angle θ entre la droite définie par le centre et le premier sommet du polygone (d'indice 0) avec l'axe horizontal.

La figure ci-dessous fait le lien entre le cercle de centre c et de rayon r en bleu et le polygone régulier d'ordre 5 associé avec un angle θ.



Voici le diagramme (incomplet) de la classe `RegularPolygon`.



— La méthode `translate` renvoie un nouveau polygone qui est issu de la translation par le vecteur (dx, dy) du polygone initial avec lequel la méthode est appelée.

— La méthode `rotate` renvoie un nouveau polygone qui est issu de la rotation d'angle `angle` et de center `center` du cercle associé au polygone avec lequel la méthode est appelée.

— La méthode `getSideLength()` renvoie la longueur ℓ d'un côté du polygone qui est donnée par la formule $\ell = 2r\sin(\pi/n)$.

— La méthode `getApothemLength` renvoie la longueur h de l'apothème du polygone qui est donnée par la formule $h = r\cos(\pi/n)$.

— La méthode `perimeter()` renvoie le périmètre p du polygon qui est donnée par la formule $p = \ell n$.

— La méthode `area()` renvoie l'aire a du polygon qui est donnée par la formule $a = \frac{ph}{2}$.

— La méthode `getVertex(int i)`, pour `i` compris entre 0 et l'ordre moins 1, permet d'obtenir le sommet d'indice i du polygone. Les sommets sont indicés à partir de 0 en commençant par celui dont l'angle à l'horizontale est donné par la propriété `angle`. Ils sont ordonnés dans le sens trigonométrique (sens inverse des aiguilles d'une montre illustré par v_0, v_1, v_2, v_3 et v_4 dans la figure).

Tâche 4 : Écrivez le code de la classe `RegularPolygon` :

— Commencez par les attributs, dont vous donnerez le type.
— Continuez avec les deux constructeurs.
— Finissez par les méthodes.

C'est à vous de choisir les droits d'accès (`private` ou `public`) des attributs et des méthodes.

Documentation

Pour toute les classes, il est utile de rajouter au début de la classe une documentation décrivant à quoi sert la classe. Il est aussi possible de mettre le nom complet de l'auteur de la classe (donc votre nom).

```
/**
 * Text describing the class.
 *
 * @author First name Last name
 */
```

Tâche 5 : Rajouter de la documentation pour toutes les trois classes que vous avez écrites.

Pour toutes les méthodes publiques d'une classe, il est utile de rajouter avant la définition de la méthode un texte expliquant à quoi sert la méthode.

```
  /**
   * Text describing the method.
   *
   * @param parameterName text describing {@code parameterName}.
   * @return text describing the returned value.
   */
```

Tâche 6 : Rajouter de la documentation pour toutes les méthodes classes que vous avez écrites.