Aix*Marseille université
Initiative d'excellence
Année 2020-21

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 6
Code UE : SIN3U02

Dessins Géométriques

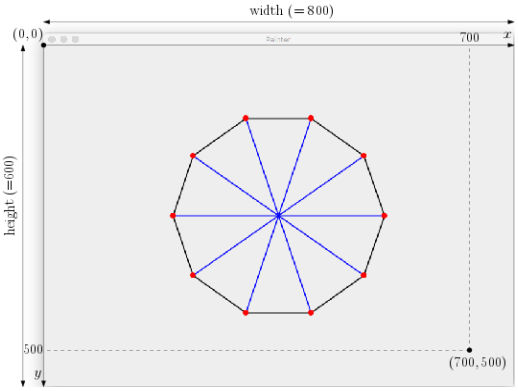
L'objectif de ce TP est de réaliser des dessins à l'écran, à l'aide de points et de lignes. Pour cela, vous utiliserez le paquet `tp2.lib`.

1 Le paquet `tp2.lib`

Le paquet `tp2.lib` est contenu dans le fichier `tp2.jar`. Le paquet `tp2.lib` possède une classe `Painter`, qui possède à son tour :

- un constructeur qui prend respectivement la largeur et la longueur en pixels de la zone de dessin.
- une méthode `void addPoint(double x, double y, Color color)` qui ajoute au dessin un point (x,y) de la couleur demandée.
- une méthode `void addLine(double x1, double y1, double x2, double y2, Color color)` qui ajoute au dessin une ligne entre les points $(x1, y1)$ et $(x2, y2)$ de la couleur donnée en argument.

L'origine du système de coordonnées se trouve dans le coin supérieur gauche de la fenêtre et l'axe des y est inversé : les points ayant une coordonnée positive en y se trouve en dessous de l'axe x.



1.1 Rappel : Comment utiliser un fichier `.jar`

Pour compiler un programme qui utilise un bibliothèque `tp2.jar`, vous devez signaler au compilateur Java de chercher dans le fichier jar pour trouver les classes de la bibliothèque. L'option "-cp" peut être utilisée pour donner le chemin vers le bibliothèque lors de la compilation avec `javac` dans le terminal :

```
javac -cp "./tp2.jar:./" Point.java
```

Si vous utilisez IntelliJ, ajoutez la bibliothèque `tp2.jar` à votre projet. Pour cela, il faut aller dans menu `File`, puis `Project Structure`. Il faut ensuite sélectionner `Libraries`, cliquer sur le `+` puis `java` et enfin sélectionner le fichier `tp2.jar`

1.2 Dessiner avec `addPoint()` et `addLine()`

Copiez le code ci-dessous dans la classe `Main` de votre projet pour tester le bon fonctionnement du paquet `tp2.lib` :

```
public static void main(String[] args){
    Painter painter = new Painter(400, 400);
    painter.addLine(100, 100, 300, 100, Color.black);
    painter.addLine(300, 100, 300, 300, Color.black);
    painter.addLine(300, 300, 100, 300, Color.black);
```

```
        painter.addLine(100, 300, 100, 100, Color.black);
        painter.addPoint(100, 100, Color.red);
        painter.addPoint(300, 100, Color.red);
        painter.addPoint(100, 300, Color.red);
        painter.addPoint(300, 300, Color.red);
    }
```

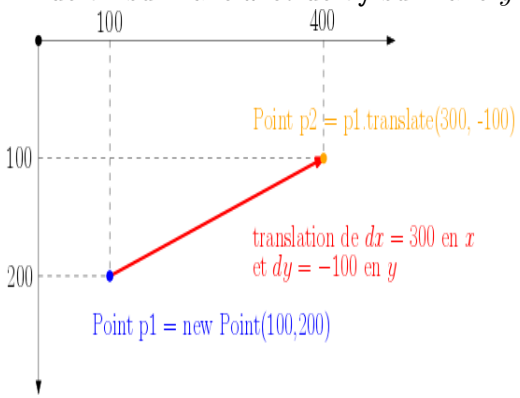
2 Classe `Point`

Écrivez une classe Point possédant :
- deux attributs `x` et `y` de type double ;
- un constructeur `Point(double x, double y)` ;
- une méthode `void draw(Painter painter, Color color)` qui dessine un point `(this.x,this.y)` de couleur `color` dans le `painter` ;
- une méthode `void drawLine(Point p, Painter painter, Color color)` qui dessine une ligne de couleur `color` entre le point `(this.x, this.y)` et le point `(p.x, p.y)` dans le `painter` ;

Vérifier le bon fonctionnement de votre code en appellant la méthode `test1_Point()` de la classe `Point`.

2.1 Ajout de fonctionnalités

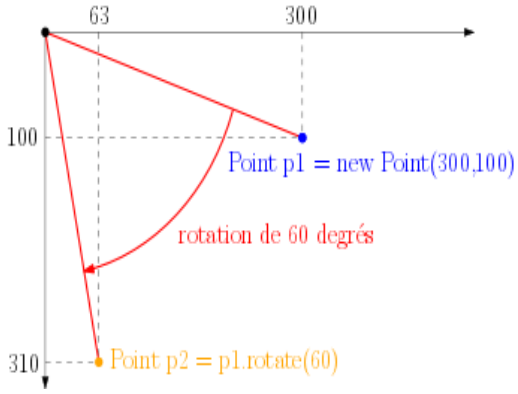
Ajoutez la méthode suivante à la classe `Point` :
- `Point translate(double dx, double dy)` qui renvoie un nouveau point issu de la translation du point de `dx` sur l'axe x et de `dy` sur l'axe y (voir image ci-dessous).



Vérifier le bon fonctionnement de cette méthode en appellant la méthode `test2_Point()` de la classe `Point`. La méthode doit avoir le même comportement que le test précédent.

2.2 Rotation d'un Point

Ajoutez la méthode `rotate(double angle)` qui crée un nouveau point en effectuant une rotation d'angle `angle` (exprimé en degrés) dans le sens des aiguilles d'une montre et de centre $(0, 0)$.



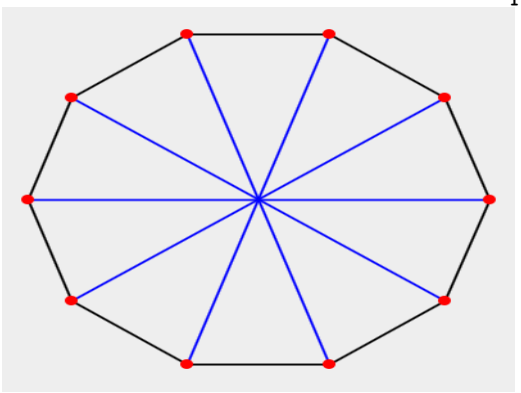
Vous devez utiliser les méthodes et propriétés statiques suivantes de la classe `Math` :

— `Math.PI` : valeur de π ;
— `Math.cos(x)` le cosinus de `x` (avec `x` exprimé en radians) ;
— `Math.sin(x)` le sinus de `x` (avec `x` exprimé en radians).

Rappel : pour effectuer une rotation, vous pouvez utiliser les formules suivantes où $\alpha = (\pi \times angle)/180$ pou obtenir les coordonnées (x', y') d'un point obtenu par rotation du point de coordonnées (x, y) :

— $x' = x\cos(\alpha) - y\sin(\alpha)$
— $y' = x\sin(\alpha) + y\cos(\alpha)$

Dans la classe `Main` écrivez une méthode `public static void dessin_Roue()` qui dessine la roue suivante :



3 Shape : Rectangle et Triangle

On désire maintenant créer des classes permettant de dessiner des formes géométriques, tel que des rectangles et des triangles. Les classes `Rectangle` et `Triangle` doivent implementer l'interface `Shape` qui contient les méthodes suivants :

— `getPerimeter()` pour calculer le périmètre
— `getArea()` pour calculer le périmètre
— `translate(int dx, int dy)` pour le translation du objet de `dx` sur l'axe x et de `dy` sur l'axe y (comme on a vu pour `Point`)
— `draw(Painter painter, Color color)` pour dessiner l'objet géométrique dans le `painter`

La classe `Rectangle` est représentée par deux points `p1` et `p2` qui correspondent respectivement aux points situés en haut à gauche et en bas à droite du rectangle. Cette classe contient les méthodes `height()` et `width()` pour calculer le longueur and largeur du rectangle à partir de points `p1` et `p2`. Elle doit de plus implementer toutes les méthodes de l'interface `Shape`.

Vérifier le bon fonctionnement de votre code en appellant la méthode `test_rectangle()` de la classe `Rectangle`.

La classe `Triangle` permet de représenter un triangle par une liste `Point[]` de trois points. Cette classe

doit implementer les méthodes de l'interface `Shape`. Ajoutez aussi une méthode `test_triangle()` pour vérifier le bon fonctionnement de votre code (vous pouvez vous inspirer par la méthode `test_rectangle`)

4 Classe `Turtle` (Tortue)

La classe `Turtle` permet de simuler les déplacement d'une tortue. Cette dernière est un robot qui se situe au centre de la fenêtre au début du programme et qui regarde vers le haut. Elle possède un crayon qui lui permet de dessiner une ligne lorsqu'elle avance. Il est de possible de lui donner les ordres suivants :

- **[forward x]** fait avancer la tortue de **x** pas.
- **[turnLeft x]** fait tourner la tortue de **x** degrés sur sa gauche.
- **[turnRight x]** fait tourner la tortue de **x** degrés sur sa droite.
- **[penUp]** lève le crayon.
- **[penDown]** pose le crayon sur la surface de dessin.
- **[setColor color]** choisit le crayon de couleur **color**.

La suite suivante d'ordres permet de dessiner un carré rouge de 40 pas de côté :

```
[setColor red]
[penDown]
[forward 40]
[turnLeft 90]
[forward 40]
[turnLeft 90]
[forward 40]
[turnLeft 90]
[forward 40]
```

La suite d'ordres précédente doit pouvoir être simulée par la classe `Turtle` en écrivant le code suivant :

```
Turtle turtle = new Turtle();
turtle.setColor(Color.red);
turtle.setPenDown();
turtle.moveForward(40);
turtle.turnLeft(90);
turtle.moveForward(40);
turtle.turnLeft(90);
turtle.moveForward(40);
turtle.turnLeft(90);
turtle.moveForward(40);
```

La classe `Turtle` possède :

- les attributs suivants :
  - `penColor` de type `Color` qui correspond à la couleur du crayon de la tortue.
  - `angleDirection` de type `double` qui correspond à l'angle entre l'axe x et la direction courante de la tortue (compté dans le sens des aiguilles d'une montre).
  - `position` de type `Point` qui correspond à la position courante de la tortue.
  - `penDown` de type `boolean` qui vaut vrai si le stylo est posé (la tortue dessine un trait correspondant à sa trajectoire lorsqu'elle avance) et faux sinon (la tortue ne dessine rien lorsqu'elle avance).
  - `painter` : l'objet de la classe `Painter` dans lequel la tortue dessine.
- Un constructeur :
  - `public Turtle(int width, int height)` : crée une tortue en initialisant les attributs de la manière suivante :

  - `penColor` : initialisé à `null`
  - `angleDirection` : initialisé de sorte que la tortue pointe vers le Nord.
  - `painter` : initialisé en créant un nouveau `Painter` dont la taille (`width` et `height`) correspond aux attributs du constructeur.
  - `penIsDown` : initialisé à `false`.
  - `position` : initialisé en créant un nouveau `Point` au centre de la fenêtre.
- les méthodes :
  - `public void moveForward(double distance)` : attend 500 millisecondes (en utilisant `sleep` expliqué ci-dessous) et fait avancer la tortue dans sa direction courante de la `distance` donnée (traçant un trait si le crayon est posé).
  - `public void setColor(Color color)` : change la couleur du crayon par `color`.
  - `public void turnLeft(double angle)` : change la direction courante de la tortue en la faisant tourner de `angle` degrés dans le sens **inverse** des aiguilles d'une montre.
  - `public void turnRight(double angle)` : change la direction courante de la tortue en la faisant tourner de `angle` degrés dans le sens des aiguilles d'une montre.
  - `public void setPenDown()` : pose le stylo (met `penDown` à `true`)
  - `public void setPenUp()` : lève le stylo (met `penDown` à `false`)

Pour dessiner à l'écran, votre tortue doit déléguer le dessin des lignes à la classe `Point` de la première partie. Notez que vous pouvez également faire dormir le programme pendant `n` millisecondes en utilisant la fonction statique `sleep(n)` de la classe `Tools` du paquet `tp2.lib`.

Une fois votre classe écrite, testez-là avec le code suivant :

```
public static void drawSquare(Turtle turtle, int size) {
  for (int i = 0; i < 4; i++) {
    turtle.moveForward(size);
    turtle.turnLeft(90);
  }
}

public static void test_turtle() {
  Turtle turtle = new Turtle(800,600);
  turtle.setColor(Color.black);
  turtle.setPenDown();
  int n = 20;
  for (int i = 0; i < n; i++) {
    turtle.turnRight(360.0/n);
    drawSquare(turtle, 100);
  }
}
```

5 Fractales

1. Ajouter à la classe `Turtle` une méthode `void drawString(String sequence, double length, double angle)` permettant de faire exécuter à la tortue la séquence d'ordres codée dans une chaîne de caractères `sequence`. La chaîne de caractères doit être lue de la gauche vers la droite et chacun des caractères doit être interprété de la façon suivante :
   - le caractère **'A'** fait avancer la tortue de `length` pas ;
   - le caractère **'+'** fait tourner la tortue de `angle` degrés à droite ;
   - le caractère **'-'** fait tourner la tortue de `angle` degrés à gauche ;
   - les autres caractères doivent être ignorés.

Pour tester votre méthode, dessinez un carré avec la séquence `"A+A+A+A"`, une `length` de `100` et un `angle` de 90°.

2. Nous allons appliquer plusieurs fois un ensemble de règles de réécriture sur une séquence de caractères afin d'obtenir une séquence d'ordres qui dessine une fractale à l'écran. Une règle de réécriture est un couple $(c \to S)$. Écrivez une classe `Rule` modélisant une règle de réécriture. Cette classe possède :
   — deux attributs privés `symbol` (de type `char`) et `sequence` (de type `String`) ;
   — un constructeur qui permet d'initialiser les deux attributs ;
   — deux accesseurs `char getSymbol()` et `String getSequence()` permettant d'obtenir les valeurs des deux attributs.
3. Écrivez la classe `SetOfRules` modélisant un ensemble de règles. Cette classe possède :
   — un attribut privé `rules` de type `Rule[]` ;
   — un constructeur permettant d'initialiser l'attribut `rules` ;
   — une méthode publique `String apply(String sequence)` qui applique les règles du tableau `rules` à la chaîne de caractères sequence et qui retourne le résultat. Appliquer un ensemble de règles $\{(c_1 \to S_1), \ldots, (c_n \to S_n)\}$ sur la séquence `sequence` consiste à remplacer chaque caractère c_i de la chaîne `sequence` par la chaîne `S_i`. Par exemple, l'application des règles $\{('A' \to' AB'), ('B' \to' BA')\}$ sur `"AB"` doit produire `"ABBA"`.
4. Testez votre programme en exécutant la fonction drawFractale() dans la classe `Main` (il faut enlever le commentaire).