amU Faculté des sciences Aix Marseille Université

Année 2025-26

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

Classe `AngleConversion`

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

```java
/**
 * The class {@Code AngleConversion} contains methods for performing angle conversions
 * from degrees to radian and vice versa.
 */

public class AngleConversion {
  private AngleConversion(){}

  /**
   * Converts an angle measured in degrees to an approximately
   * equivalent angle measured in radians.
   *
   * @param angleInDegrees an angle, in degrees
   * @return the measurement of the angle {@code angleInDegrees} in radians
   */
  public static double toRadians(double angleInDegrees) {
    return (Math.PI / 180) * angleInDegrees;
  }
  /**
   * Converts an angle measured in radians to an approximately
   * equivalent angle measured in degrees.
   *
   * @param angleInRadians an angle, in radians
   * @return the measurement of the angle {@code angleInRadians} in degrees
   */
  public static double toDegrees(double angleInRadians) {
    return (180. / Math.PI) * angleInRadians;
  }
}
```

Classe `Point`

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

```java
/**
 * A point representing a location in (x,y) coordinate space,
 * specified in double precision.
```

```
 */

public class Point {
  private final double x;
  private final double y;

  /**
   * Creates a new instance of {@code Point} with the specified coordinates.
   *
   * @param x the X coordinate of the point
   * @param y the Y coordinate of the point
   */
  public Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  /**
   * Returns the X coordinate of this {@code Point} in double precision.
   *
   * @return the X coordinate of this {@code Point}.
   */
  public double getX() {
    return x;
  }
  /**
   * Returns the Y coordinate of this {@code Point} in double precision.
   *
   * @return the Y coordinate of this {@code Point}.
   */
  public double getY() {
    return y;
  }

  /**
   * Returns the distance from this {@code Point} to a specified {@code Point}.
   *
   * @param p the specified point to be measured against this {@code Point}
   * @return the distance between this {@code Point} and the specified {@code Point}.
   */
  public double distanceTo(Point p){
    return Math.hypot(p.getX() - getX(), p.getY() - getY());
  }

  /**
   * Returns a string representation of this {@code Point} in the format (x,y).
   *
```

```
     * @return a string representation of this {@code Point}
     */
    @Override
    public String toString() {
      return "(" + getX() +"," + getY() + ')';
    }

    /**
     * Returns a new point with the coordinates of the specified point added to the
     * coordinates of this point.
     *
     * @param vector the point whose coordinates are to be added
     * @return the point with added coordinates
     */
    Point translate(Point vector){
      return new Point(getX() + vector.getX(), getY() + vector.getY());
    }
    /**
     * Returns a point that is the result of the rotation of this point around the
     * origin with the specified angle.
     *
     * @param angleInDegrees the angle of the rotation expressed in degrees
     * @return the rotated point
     */
    private Point rotate(double angleInDegrees){
      double angleInRadians = AngleConversion.toRadians(angleInDegrees);
      return new Point(x * Math.cos(angleInRadians) - y * Math.sin(angleInRadians),
              x * Math.sin(angleInRadians) + y * Math.cos(angleInRadians));
    }
    /**
     * Returns a point that is the opposite (opposite coordinates) of this point.
     *
     * @return the opposite point
     */
    public Point opposite(){
      return new Point(-getX(), -getY());
    }
    /**
     * Returns a point that is the result of the rotation of this point around
     * the specified pivot and with the specified angle.
     *
     * @param pivot the center of the rotation
     * @param angle the angle of the rotation expressed in degrees
     * @return the rotated point
     */
    public Point rotate(double angle, Point pivot){
      Point translated = translate(pivot.opposite());
```

```
    Point rotated = translated.rotate(angle);
    return rotated.translate(pivot);
  }

  /**
   * Computes the angle (in degrees) between the vector from
   * the specified point to this point and the horizontal axe.
   *
   * @param origin the origin of the vector
   * @return the angle between the vector and the horizontal axe
   * measured in degrees
   */
  public double angleFrom(Point origin){
    return AngleConversion.toDegrees(Math.atan2(getX() - origin.getX(), getY() - origin.getY()));
  }

}
```

Classe `Circle`

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

```
/**
 * The Circle class creates a new circle with the specified radius and center location.
 */
public class Circle {
  private final Point center;
  private final double radius;

  /**
   * Creates a new instance of Circle with a specified center and radius.
   *
   * @param center the center of the circle
   * @param radius the radius of the circle
   */
  public Circle(Point center, double radius) {
    this.center = center;
    this.radius = radius;
  }

  /**
   * Gets the value of the center of this circle.
   *
   * @return the value of the center
   */
  public Point getCenter() {
```

```
  return center;
}
/**
 * Gets the value of the radius of this circle.
 *
 * @return the value of the radius
 */
public double getRadius() {
  return radius;
}
/**
 * Computes the area defined by this circle.
 *
 * @return the area of this circle
 */
public double area(){
  return Math.PI * radius * radius;
}
/**
 * Computes the perimeter of this circle.
 *
 * @return the perimeter of this circle
 */
public double perimeter(){
  return Math.PI * radius * 2.;
}
/**
 * Returns a new circle which is the result of the translation
 * of this circle by the specified vector.
 *
 * @param vector the point whose coordinates defined the coordinates of the translation
 * @return the translated circle
 */
public Circle translate(Point vector){
  return new Circle(getCenter().translate(vector), getRadius());
}
/**
 * Returns a circle that is the result of the rotation of this circle around the
 * origin with the specified angle.
 *
 * @param angleInDegrees the angle of the rotation expressed in degrees
 * @return the rotated circle
 */
public Circle rotate(double angleInDegrees, Point pivot){
  return new Circle(getCenter().rotate(angleInDegrees, pivot), getRadius());
}
```

```
  /**
   * Returns a string representation of this circle in the format
   * Circle{center=(X.X,Y.Y), radius=R.R}.
   *
   * @return a string representation of this circle
   */
  @Override
  public String toString() {
    return "Circle{" +
            "center=" + center +
            ", radius=" + radius +
            '}';
  }
}
```

Classe `RegularPolygon`

```
/**
 * The Circle class creates a new regular polygon with the specified number of vertices,
 * radius, center location and the angle in degrees between the vector from the center
 * to the first vertex and the horizontal axe.
 */
public class RegularPolygon {
  private final int order;
  private final Circle circumscribedCircle;
  private final double angleInDegrees;

  /**
   * Creates a new regular polygon with the specified number of vertices, radius,
   * circumscribed circle and the angle in degrees between the vector from the center
   * to the first vertex and the horizontal axe.
   *
   * @param order the number of vertices of the polygon
   * @param circumscribedCircle the circumscribed circle of the polygon
   * @param angleInDegree the angle in degrees between the vector from the center
   *                      to the first vertex and the horizontal axe.
   */
  public RegularPolygon(int order, Circle circumscribedCircle, double angleInDegree) {
    this.order = order;
    this.circumscribedCircle = circumscribedCircle;
    this.angleInDegrees = angleInDegree;
  }

  /**
   * creates a new regular polygon with the specified number of vertices, radius,
   * center location and the angle in degrees between the vector from the center
```

```
 * to the first vertex and the horizontal axe.
 *
 * @param order the number of vertices of the polygon
 * @param center the center of the polygon
 * @param radius the distance between the center and the vertices
 * @param angleInDegree the angle in degrees between the vector from the center
 *                      to the first vertex and the horizontal axe.
 */
public RegularPolygon(int order, Point center, double radius,  double angleInDegree) {
  this(order, new Circle(center, radius), angleInDegree);
}
/**
 * Returns the center of the polygon.
 *
 * @return the center of the polygon
 */
public Point getCenter(){
  return circumscribedCircle.getCenter();
}
/**
 * Returns the vertex of the specified index.
 *
 * @param index the index of the vertex
 * @return the vertex of the specified index
 */
public Point getVertex(int index){
  return new Point(circumscribedCircle.getCenter().getX()+circumscribedCircle.getRadius(),
          circumscribedCircle.getCenter().getY())
          .rotate(angleInDegrees + (order / 360.) * index, getCenter());
}
/**
 * Returns the length of a side of the polygon.
 *
 * @return the length of a side of the polygon
 */
public double sideLength(){
  return 2 * circumscribedCircle.getRadius() * Math.sin(Math.PI/order);
}
/**
 * Returns the length of the apothem, i.e., a line segment from the center
 * of the polygon to the midpoint of one of its sides.
 *
 * @return the length of the apothem
 */
public double getApothemLength(){
  return circumscribedCircle.getRadius() * Math.cos(Math.PI/order);
}
```

```
  /**
   * Computes the perimeter of this regular polygon.
   *
   * @return the perimeter of this regular polygon
   */
  public double perimeter(){
    return order * sideLength();
  }
  /**
   * Computes the area of this regular polygon.
   *
   * @return the area of this regular polygon
   */
  public double area(){
    return perimeter() * getApothemLength() / 2.;
  }
  /**
   * Returns a new regular polygon that is the result of the rotation of this
   * regular polygon around its center with the specified angle.
   *
   * @param angleInDegrees the angle of the rotation expressed in degrees
   * @return the rotated regular polygon
   */
  public RegularPolygon rotate(double angleInDegrees){
    return new RegularPolygon(order, circumscribedCircle,
             this.angleInDegrees+angleInDegrees);
  }
  /**
   * Returns a new regular polygon which is the result of the translation
   * of this regular polygon by the specified vector.
   *
   * @param vector the point whose coordinates define the coordinates of the translation
   * @return the translated regular polygon
   */
  public RegularPolygon translate(Point vector){
    return new RegularPolygon(order, circumscribedCircle.translate(vector), angleInDegrees);
  }

}
```