עבודה מס' 3: רקורסיה ותכנות מונחה עצמים

הוראות מקדימות :

1. העבודה כתובה בלשון זכר (מטעמי נוחיות), אך פונה לשני המינים
2. ניתן ומומלץ לבצע את העבודה בזוגות בהתאם להוראות המופיעות בסילבוס הקורס.
3. כל איחור בהגשת התרגיל יגרום להורדת ציון, כפי שהוגדר בנהלים שבאתר הקורס.
4. תרגיל אותו לא ניתן להדר(לקמפל) או להריץ , יקבל ציון נכשל.
5. לעבודה מצורפים שלושה קבצים Painter.java, example.java ו- Main.java. הקובץ Painter.java מספק לכם ממשק גרפי שיעזור לכם לבדוק את הקוד שתכתבו על ידי יצירת התמונה. בקובץ example.java תמצאו דוגמא לשימוש בממשק הגרפי Painter ובקובץ main.java תמצאו בדיקות לכל המחלקות היוצרות את התמונות כפי שהם מופיעות בעבודה.
6. <u>יש להגיש את קבצי ה- java , מכווצים כקובץ ZIP יחיד</u>. הקבצים להגשה הם: Pixel.java, BasicStar.java, SnowFlake.java, SuperSnowflake.java, KochCurve.java, KochSnowflake.java. יש להגיש את הקבצים האלו בלבד (ללא קבצי ה **class** וכו). כדי להגיש יש לכווץ אותם לקובץ zip. **קבצים שיוגשו בפורמט שונה מzip לא ייבדקו**.
7. את הקובץ יש להגיש ב Submission System.
8. העבודה תיבדק **באופן אוטומטי**. לכן, יש להקפיד על ההוראות ולבצע אותן **<u>במדויק</u>**. כל חוסר מעקב אחר ההוראות **יגרור פגיעה משמעותית בציון**.
9. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד ברור, מתן שמות משמעותיים למשתנים, הזחות (אינדנטציה), **<u>והוספת הערות בקוד</u>** המסבירות את תפקידם של מקטעי קוד שונים. אין צורך למלא את הקוד בהערות סתמיות אך חשוב לכתוב הערות בנקודות קריטיות המסבירות קטעים חשובים בקוד. כתיבת קוד אשר אינו עומד בסטנדרטים אלו תגרור <u>הפחתה בציון העבודה</u>. בנוסף, הערות יש לרשום אך ורק ב**אנגלית** (כתיבת הערות בכל שפה אחרת שקולה לאי כתיבת הערות).
10. בכדי לוודא שהגשתם את התרגיל באופן תקין – הורידו את קובץ הZIP שהגשתם ב Submission System למחשב שלכם, חלצו אותו ונסו להדר ולהריץ את התוכנית. הליך זה הוא חשוב, עקב מקרים שקרו בעבר בהם הקובץ שהועלה למערכת ההגשות היה פגום.
11. במידה ואינכם בטוחים מהו הפירוש המדויק להוראה מסוימת, או בכל שאלה אחרת הקשורה **<u>בתוכן</u>** העבודה, אנא היעזרו בפורום או בשעות הקבלה של האחראים על העבודה (פרוט שעות הקבלה מופיע באתר הקורס). בכל בעיה **<u>אישית</u>** הקשורה בעבודה (מילואים, אשפוז וכו'), אנא צרו את הפניה המתאימה במערכת הגשת העבודות כפי שמוסבר בסילבוס שבאתר הקורס (כאן).

12. שימו לב כי בעבודה ניתנו 5 נקודות "תמריץ" עבור מעקב אחר הוראות העבודה. במידה וההוראות לא מולאו (שכחתם להוסיף שותף במידה ובחרתם להגיש בזוגות, חוסר בדיקה של פורמט הקבצים, שמות לא נכונים וכדומה) הדבר יגרור הורדה של נקודות אלו.

13. בעבודה זו ניתן לצבור עד 110 נקודות:

1. 5 נקודות עבור מעקב אחר הוראות (סעיף 12)
2. 15 נקודות עבור הבדיקה הידנית - כתיבת הערות (7 נק'), הזחה (4 נק') ושמות משתנים (4 נק').
3. 80 נקודות על כתיבת הקוד בהתאם לדרישות התרגיל.
4. 10 נקודות בונוס על חלק Von Koch Snowflake.

לתרגיל מצורפים 3 קבצים:

1. **Painter.java:**מספק לכם ממשק גרפי שיעזור לכם לבדוק את הקוד שתכתבו על ידי יצירת התמונה.
2. **example.java**: מספק דוגמא לשימוש בממשק הגרפי Painter.
3. **Main.java** : מכיל בדיקות לכל המחלקות היוצרות את התמונות כפי שהם מופיעות בעבודה.

שימו לב: <u>אין צורך</u> לקרוא ולהבין את הקוד בקובץ **Painter.java.**

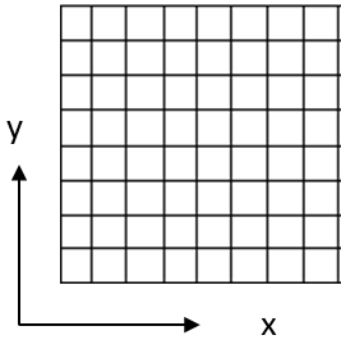
- תוכלו להוסיף פונקציות עזר נוספות לבחירתכם. <u>שימו לב</u>: קריאות לפונקציות עזר שכתבתם, ייעשו **רק** מתוך השיטות אותן אתם מממשים, ולא מתוך פונקציית ה-main שנועדה לבדיקה.

Assignment Description:

In this assignment you will create graphic images by using recursive Java methods.

Graphic Images

Graphic images are drawn in a rectangle. The rectangle has known *width* and *height*. Its origin is in the bottom left at coordinates (0,0).



The Painter

Use the **provided** Painter class (i.e., Painter.java) to open a window frame and draw images. In main.java you will find an example for using the Painter class. The Painter has the following useful methods:

- **draw(String shape)**– Creates a frame and displays the shape name.
- **getFrameWidth()–** Returns the width of the frame(integer).
- **getFrameHeight()–** Returns the height of the frame(integer).
- **drawLine(Pixel p1, Pixel p2)-** Draws a straight line from point p1 to point

p2.

Part I – Starry Night (55 Points)

In this part you will write recursive methods for drawing stars and snowflakes.

Pixel (10 Points)

The pixel class represents a single point. It should have two fields (i.e., x and y), two constructors, getter methods and functions translate, rotateRelativeToAxesOrigin, and rotateRelativeToPixel.
Please see the class's structure below.

```
public class Pixel{

      private double x;
      private double y;

      public Pixel(){
            x=0;
            y=0;
      }

      public Pixel(double x, double y){
            // Your code here
      }

      public double getX(){
            // Your code here
      }

      public double getY(){
            // Your code here
      }
      public void translate(Pixel p){
            // Your code here (see appendix)
      }
      public void rotateRelativeToAxesOrigin(double theta){
            // Your code here (see appendix)
      }
      public void rotateRelativeToPixel(Pixel p1,double theta){
            // Your code here
      }
}
```

The Fields

- The field x describes the x-coordinate of the pixel.
- The field y describes the y-coordinate of the pixel.

The Constructor

- The empty constructor initializes the fields of the pixel (0,0).
- The second constructor receives two parameters and initializes the class fields from the given arguments.

The Getters

The getter methods will help you access the fields of the Pixel class (see example in the main function or of similar things done in class).

translate (Pixel p) method

You will have to implement this method.

The method translate the pixel using the given pixel p. See the appendix for a formal definition of translation and the way it is calculated.

rotateRelativeToAxesOrigin (double theta) method

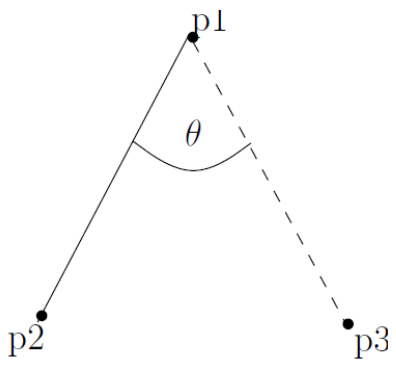
You will have to implement this method.

The method rotateRelativeToAxesOrigin gets a rotation angle θ and rotates the pixel relative to the axes origin. See the appendix for a formal definition of rotation and the way it is calculated.

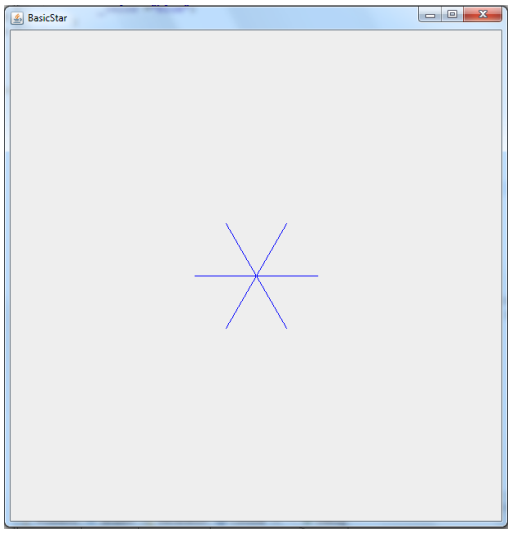
rotateRelativeToPixel (Pixel p1, double theta) method

You will have to implement this method.

The method rotateRelativeToPixel gets pixels p1 and a rotation angle θ. The method returns a new pixel p3 that is given by rotating the pixel by angle θ (counter clockwise) relative to pixel p1 (as depicted in the figure to the right). The way to calculate this is not specified in the appendix and is up to you to understand.



BasicStar (10 Points)



This class creates a Basic Star, which is made of 6 lines that originate from a central point. This class's structure is as follows:

```
public class BasicStar {
   private Pixel center;
   private double radius;


   public BasicStar(){
         double height = Painter.getFrameHeight()/2;
         double width = Painter.getFrameWidth()/2;
         this.center = new Pixel (width, height);
         double maxRadius = Math.min(width, height)/2;
         this.radius = maxRadius/4;

   }

   public BasicStar(Pixel center, double radius){
         // Your code here
   }

   public Pixel getCenter(){
         // Your code here
   }

   public double getRadius(){
         // Your code here
   }

   public void draw(){
         // Your code here
   }


}
```

empty constructor

This method initializes a star in the middle of the frame with radius relative to the width and height of the frame:

- The width and height are needed to determine the **radius** of the star that fits in the frame. To draw a symmetrical star, it finds its radius, which is the minimum of the frame's width and height, divided by 2. Let us further decrease the radius by a factor of 4 (dividing the radius by 4) so that there is

room for the rest of the snowflake.

- The star is drawn in the **center** of the rectangle.

draw() method

You will have to implement this method.

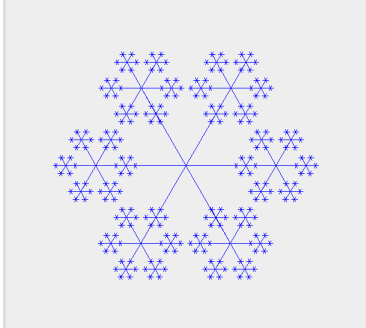
This method draws a star given the point of its center and its radius:

1. **Each line (of the six) starts at the center p**, and ends at a point on the circle of the given radius.
2. A circle has $2 \cdot \pi$ radians, so one sixth of a circle is $2\pi/6$.

Other methods

Complete the specified getter methods and the non-empty constructor.

Snowflake- Recursive star (15 Points)



Create a Snowflake class, similar to BasicStar, that draws a snowflake. A snowflake will be drawn by adding another star centered at the tip of each of the six lines,and continuing the recursive **process** to create a snowflake image.

The Fields

The Snowflake class has a two fields, a center BasicStar Object and depth.

The Empty Constructor

The Snowflake class has an empty constructor that initializes the snowflake in the center, like a star.

draw() method

This method draws a snowflake centered at pixel with given radius (both are fields of the class), and does so recursively depth times.

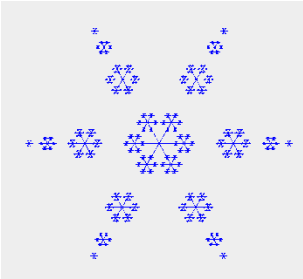
- Draw the central star using the given radius and recursively draw smaller snowflakes at the tips of the central star.
- The central stars of the new snowflakes need to be smaller than the central star. Thus, the **radius** of the new stars should be radius/3.

Other methods

Write getter methods for all the fields above, complete the code of the non-empty constructor and add methods which returns the radius and center of the center BasicStar.

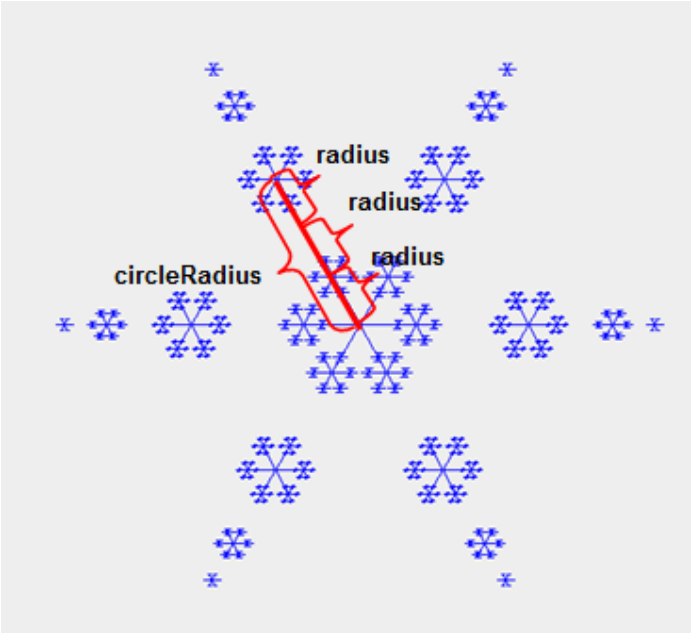
* When constructing a new snowflake using the empty constructor, and then calling the draw method, the resulting image should be as seen below. This is tested by running the given Java template files.

Super Snowflake (20 Points)



Create a SuperSnowflake class, that draws a super snowflake. The super snowflake image contains the original snowflake and additional circles of snowflakes.

- There is a central snowflake and around it there are 3 circles, each with 6 snowflakes.
- Each circle has 6 snowflakes with central stars that are 2 times smaller in radius than the central star of the previous snowflake .
- Each circle has a circleRadius which is larger than the previous circleRadius by 3***radius,** where **radius** is the radius of the snowflake's central star in the previous circle.



The Fields

The SuperSnowflake class has two field, a Snowflake Object and depth.

The Empty Constructor

The SuperSnowflake class has an empty constructor that initializes a new Snowflake object.

The draw() method

The draw() method in the SuperSnowflake is similar to the draw() method in Snowflake. It should draw the central snowflake and then the circles of snowflakes. The radius of the first circle is 3 times the radius of the Snowflake's central star. The radius of the second circle is calculated by adding to the first circle radius a distance that is 3 times the radius of the central star of the second snowflake (refer to the figures above).

Other methods

Write getter methods for all the fields above and complete the code of the non-empty constructor.

* When constructing a new super snowflake using the empty constructor, and then calling the draw method, the resulting image should be as seen above. This is tested by running the given Java template files.

Part II – The Von Koch Curve- Recursion in Fractal Geometry (25 Points + 10 Bonus)

In this part you will write recursive methods for drawing Koch Snowflake.

Swedish mathematician Helge von Koch introduced the "Koch curve" in 1904. Starting with a line segment, recursively replace the line segment as shown below:

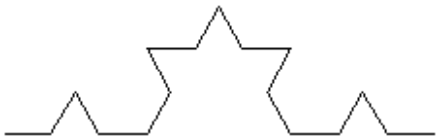
0

1

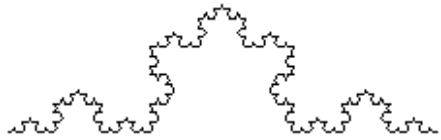
The single line segment in Step 0 is divided into 3 parts of equal length and the central part is further broken into two segments, such that the resulting 4 segments all have equal length. This same "rule" is applied an infinite number of times resulting in a figure with an infinite perimeter.

here are the next few steps:

2



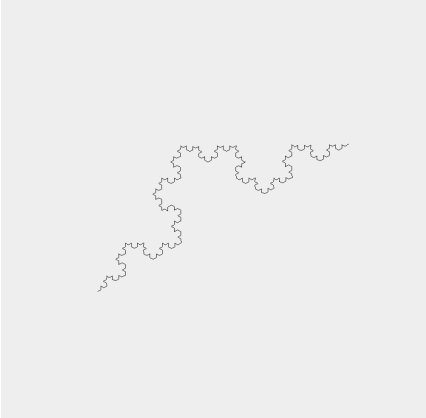
3

4 

...

For more information and further reading,
http://en.wikipedia.org/wiki/Koch_snowflake

The Von KochCurve (25 Points)



Create a KochCurve class, which draws the curve shown in step 4.

The Fields

The KochCurve class has 2 pixels (start and end) and the depth.

The Empty Constructor

The KochCurve class contains an empty constructor for initialization.
The strat pixel should be at the center of the frame and the end pixel should be at the center of the upper right quarter.

draw() method

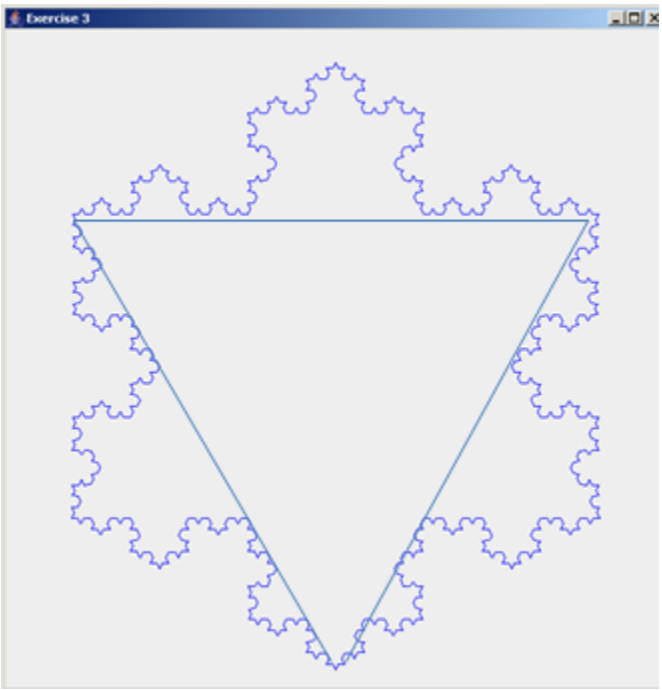
Write a recursive method **draw()** that draw the koch curve from its start and end points, and does so recursively according to its depth.

- Find 2 new pixels p3=(x3,y3) and p4=(x4,y4) which equally divide the segment p1p2 into 3 identical segments.
- Find a pixel p5=(x5, y5) that creates the equilateral triangle with p3 and p4.
- Add a recursive call for each of the 4 segments.

* When constructing a KochCurve using the empty constructor, and then calling the draw method, the resulting image should be as seen above. This is tested by running the given Java template files.

Bonus: The Von Koch Snowflake (10 Points)

Create a KochSnowflake class, which draws the KochSnowflake shown in the image below.



The Fields

The KochSnowflake class contains the 3 points of the triangle that describes it.

The Constructor

The KochSnowflake class has an empty constructor that initializes a new KochCurve object. The triangle's center point is the center of the frame (x=FrameWidth/2, y=FrameHeight) and the y value (the height) of the triangle's upper base is 40.

draw() method

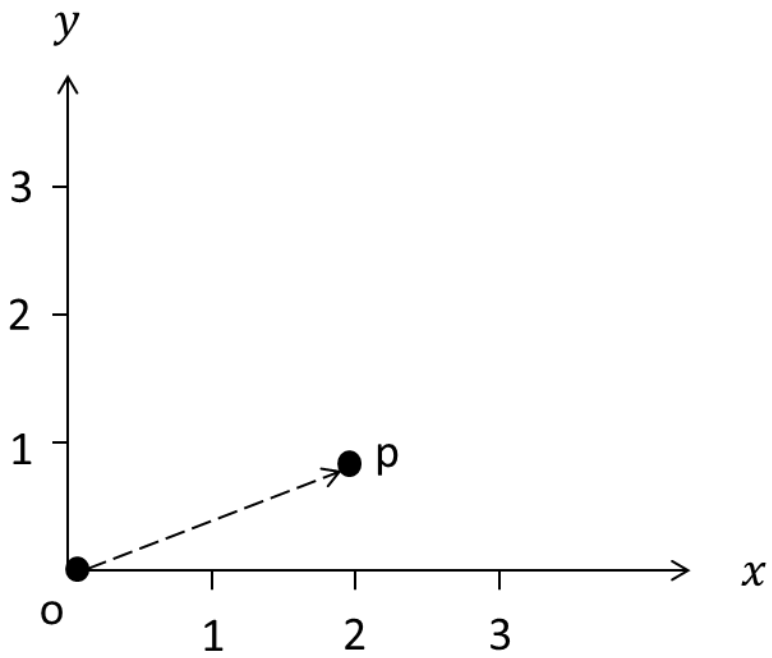
The draw() method draws koch curves using the triangle's sides.

* When constructing a new KochSnowflake using the empty constructor, and then calling the draw method, the resulting image should be as seen above. This is tested by running the given Java template files.

.**Good Luck!!!**

Appendix – 2D Geometry (translation and rotation)

In this section we describe basic manipulation of 2D points. Such points are defined by two coordinates, x and y, and may be marked by (x,y). For example, the point in the origin of the axes is point $O=(0,0)$, and if we move two steps to the right and one step up (direction $d=(2,1)$) we reach point $p' = (2,1)$, as seen in the figure below:



Translation

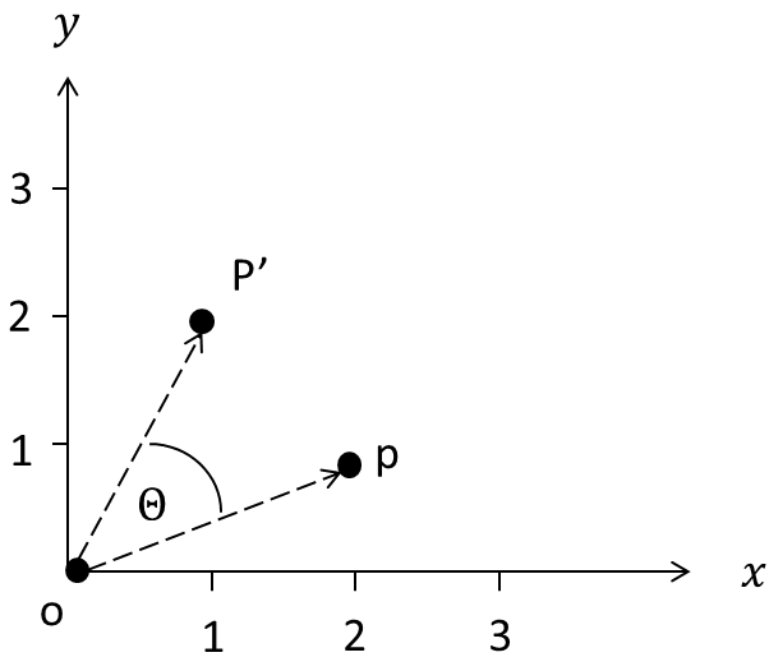
Moving a point from one place to another (as depicted in the figure above) is named Translation. Translating a point $p=(p_1,p_2)$ by direction $d=(d_1,d_2)$, we get a new point $p' = (p'_1,p'_2)$ by:

$$p' = p+d = (p_1,p_2)+(d_1,d_2) = (p_1+d_1,p_2+d_2)$$

(Notice that these values may be negative)

Rotation

A point may be rotated by an angle from the axes origin (the point O):



More formally, point $p = (p_1, p_2)$ is rotated by angle θ to form the point $p' = (p'_1, p'_2)$ by:

$$p' = (p_1 \cdot cos\,(\theta) - p_2 \cdot sin\,(\theta)\,, p_1 \cdot sin\,(\theta) + p_2 \cdot cos(\theta))$$

Notice that the rotation is counter clockwise, and relative to the axes origin (relative to the point O).