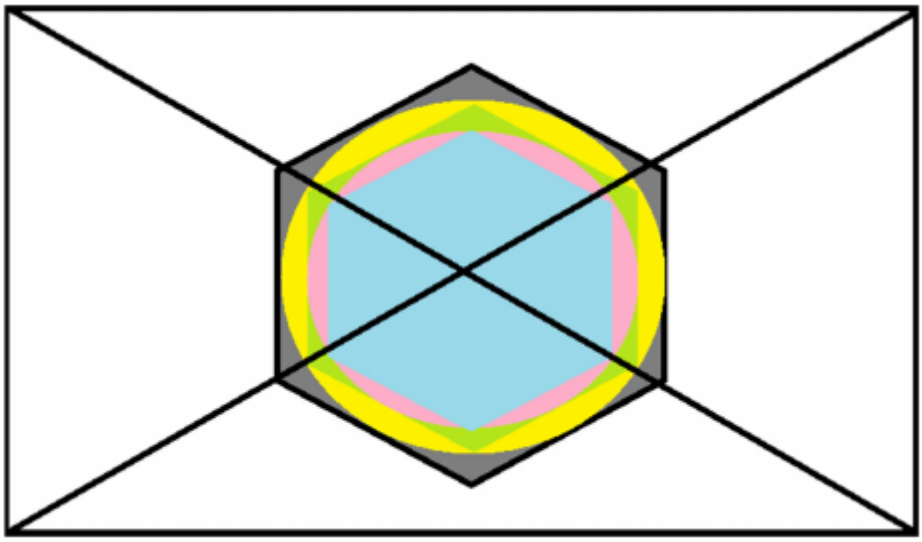
CSc 22100 Assignment 1

Leji Li

The Problem

The assignment is to build four classes, `MyShape`, `MyLine`, `MyPolygon` and `MyCircle`, and then use the `MyLine`, `MyPolygon` and `MyCircle` to draw a picture on canvas.



The pciture should be drawed

The `MyShape` is the super-class of other three classes. It should have the data fields representing a point, which has a x value and y value, and a data field representing the color of the giving shape. The `MyShape` class should has

- setters and getters addressing to the corresponding data fields;
- a `toString` method which will return the class's description in `String`;
- a `draw` method.

The `MyLine` class is a sub-class of the `MyShape` class. It is defined by two points. For the methods, this class should has

- a `getLenght` method that returns the length of the line;
- a `get_xAngle` method to return the angle between the line and the x-axis in degree;
- a `toString` method;
- a `draw` that draw a line between the given two points.

The `MyPolygon` class is a sub-class of the `MyShape` class. It takes a point (x,y) as the center, a *N* as the number of sides of the polygon, and a *r* indicates the radius of inscribed circle. This class has

- a `getArea` method returns the area of the current polygon;
- a `getPerimeter` method returns the perimeter of the current polygon;

- a `getSide` method returns the length of each side of the current polygon;
- a `toString` method;
- `draw` methods. Polygon can draw a filled polygon or a stroked polygon.

The `MyCircle` class is a sub-class of `MyShape` class. It takes a point (x,y) as the center and a radius. This class has

- a `getArea` method returns the area of current circle;
- a `getPeirmeter` method returns the perimeter of the current circle;
- a `getRadius` method returns the radius of the current circle;
- a `toString` method;
- a `draw` method that draw a filed circle.

An enum class `MyColor` class is needed. The classed above can only accept the `MyColor` obejcet as the color option, instead of using the color obejct from teh awt package.

After building up these classes, we should be able to use these class to draw the picture using JavaFX. The shapes should be drawed on a `Canvas` object and showed on a window. The dimensions have to be proportional to the smallset dimension of the canvas.

Solution Methods

`MyShape` class is the super-class of other three classes, and it takes only a point and a color object; it has nothing could draw. I set it an abstract class with an abstract method `draw`, because `MyLine`, `MyPolygon` and `Mycircle` classes inherit `MyShape` class and have their own drawing operation. Therefore, the `MyShape` class will finally has

- private `double` variables `x` and `y`.
- private `MyColor` object `color`.
    - constructors get some of all of the data fields above
    - The default x and y value are zero.
    - The default color is `MyColor.BLACK`.
- Constructs can have all three arguments, points only, color only and nothing. The missing part(s) will be set to the default value.
- a set of getters and setters to these three data fields.
- a `toString` method returns the description of this `MyShape` class.
- an abstract `draw` method that need to be override by sub-classes.

`MyLine` class inherits the `MyShape` class. It takes two points and/or a `MyColor` object. Because `MyLine` has two points, for convience purpose, I set up two Xs variables and two Ys varibales instead of using the x and y in the `MyShape` class. But I am using the color variable in the `MyShape` so that I don't have to create a new data field in `MyLine`. The `MyLine` class will finally have

- private `double` variables `x1`, `x2`, `y1`, and `y2`.
- constructors get two points and/or the color
    - the two points are required but the color is optional.
    - because this class inherits `MyShape` class, if the color is not passing to the constructor, it will use the default of `MyShape`.
- `getLength` method calculates and returns the length of line that starts at (x1, y1) and ends at (x2, y2). The formula is $\sqrt{(x1-x2)^2+(y1-y2)^2}$
- `get_xAngle` method calculates and returns the angle between the line and the x-axis
    - I use the inverse tangent function, passing in the (y1-y2)/(x1-x2).
    - Because the `Math.atan2` method will returns the angle in radios, a conversion to degree is needed.
- `toString` method return the length of the line and the xAngle.
- `draw` method draws the line represented by current `MyLine` object.
    - the `draw` method takes a `GraphicsContext` object as argument.
    - the method accepts a `GraphicsContext` object as argument.
    - use `GraphicsContext`'s `setStroke` method to set the drawing paint.
    - use `strokeLine` to draw the line, from (x1, y1) to (x2, y2).

`MyPolygon` class inherits the `MyShaped` class. This class uses the `MyShape`'s x, y and color variable, and has the *N* as the number of sides, the lower case *r* as the radius of the polygon's inscribed circle. The `MyPolygon` looks like

- private `double` variables r.
- private `int` variable N.
- constructors accept x, y, N, r and/or color
    - the color is optional, others are required.
    - because I use the x and y in the super-class, I have to call the super-class's constructor to set up the x and y value.
- `getArea` method calculates and returns the area of the polygon, giving the formula: $N * r^2 * tan(\pi/N)$
    - Because the `Math.tan` method takes the angle in radians, before passing the π/N, a conversion from degree to radians is needed.
- `getPrimeter` method calculates and returns the perimeter of the polygon. The formula is $\text{side length} * \text{number of sides}$.
- `getAngle` method calculates and returns the interior angle. The formula is 2π / N
- `getSide` method calculates and returns the length of a side. The formula is $2 * r * tan(\pi/N)$

- `toString` method return a `String` object that contains the information of side length, interior angle, perimeter and the area.
- `draw` methods. Because the `MyPolygon` class can draw a filled polygon or a stroked polygon, therefore there will be two drawing methods. One drawing method takes a `GraphicsContext` object and a boolean variable.

  ```
  public void draw(GraphicsContext gc, boolean isFill)
  ```

  The boolean variable is used to control drawing a filled polygon or a stroked polygon to the `GraphicsContext` object. The other drawing method takes a `GraphicsContext` object only.

  ```
  public void draw(GraphicsContext gc)
  ```

  This method overrides the `MyShape`'s `draw` method. This method will draw a stroked polygon. I can simply call

  ```
  draw(gc, false);
  ```

  to get the unfilled polygon drawn. The `GraphicsContext` takes the arrays of the polygon's points. I start at the center of the polygon and use the circumradius to get the points.
- `getInradiusByCircumradius` method takes the circumradius *R* and number of sides *N*, and return the inradius *r*. Because the given graphic needs to adapt to the smallest edge of the window, I let the circumradius of the polygon as the 90% of the smallest edge. But the constructor needs a inradius as the input, a converter from circumradius to inradius is needed.
- `getCircumradiusByInradius` method takes the inradius *r* and the number of sides *N*, and returns the circumradius *R*. Because the `MyPolygon` hosts the inradius *r*, and I use the circumradius *R* to draw the graph, a converter from inradius to circumradius is needed.

`MyCircle` class inherits the `MyShap` class. This class use the x, y and color variable on `MyShape`. It has a new data field radius, holds the radius of the current circle. The class has

- private `double` variable *r* for the radius.
- constructors accepts x, y, r and/or color
  - color is optional while x, y and r are required.
  - call the super-class's constructor to store the x and y.
- `getArea` method calculates and returns the area of the current circle. The formula is $\frac{4}{3} * \pi * r^2$.
- `getPerimeter` method calculates and returns the perimeter of the current circle. The formula is $2 * \pi * r$.
- `getRadius` method returns the radius *r*.
- `toString` method returns a `String` object contains the information of radius, perimeter and area.

- `draw` method accepts a `GraphicsContext` object and use `fillOval` method to draw a circle. The `fillOval` fills an oval on a rectangle box.
    - The `fillOval` method takes four arguments, `x`, `y`, `w` and `h`. The `x` and `y` here is different from the center point.
    - (x, y) is the upper left point of the rectangle box. `x` should be center x - *r* and `y` is center y - *r*.
    - `w` is the width of the oval, which is 2*r* here.
    - `h` is the height of the oval, which is 2*r* here.

`MyColor` class is an `enum` class. It has a list of pre-defined color. Because the `MyColor` object cannot be recognized by the `GraphicsContext` object, `MyColor` should be converted to the `paint.Color` object. I can get the `paint.Color` object using RGB value, therefore, a constructor of the `MyColor` takes RGB values and a method converts `MyColor` to `paint.Color` are needed.

- private `int` variable r, g and b are used to store the RGB value
- private constructor accepts the pre-defined RGB value and store to local memory space.
- `toFXPaontColor` method returns the `paint.Color` object has the corresponding RGB value with the `MyColor` object.

`Main` class is the entrance of the program. It inherits the `Application` class.

- To let the application has more flexibility, I create a form to let the user type in the dimension and the number of the sides of the polygon. In case of user input a invalid string, `getValueFromInput` method is used to get the value from `TestField`.
- The drawn graph will be showed on a pop-up window. When the user resizes the size of the pop-up window, the graph will change along with the window. I finish this feature by adding listeners to the `WidthProperty` and `HeightProperty` of the `Scene`. Every time the listener catch a window resizing event, it will remove the current canvas from the pop-up scene and put a new canvas to it.
- `getShowCanvas` method finish the drawing process. This method has three parameters: `w` is the width of the canvas; `h` is the height of the canvas; `polygonNum` is the number of sides of the polygon. This method returns a canvas object that has the drawn picture on it and being add to the `scene` of the pop-up `stage`. This method will instantiate `MyLine` objects, `MyCircle` objects and `MyPolygon` objects and calls their `draw` method to let the drawing done. This method will be called whenever there needs a update of the pop-up window, which will be the initialization of the pop-up window and the resizing events of the pop-up window.

Codes developed

MyShape.java

```
package shapes;
import javafx.scene.canvas.GraphicsContext;
import java.awt.*;

public abstract class MyShape extends Object {
    private double x;
    private double y;
    private MyColor color;
```

```
    public MyShape(double x, double y, MyColor color){
        this.x = x;
        this.y = y;
        this.color = color;
    }
    public MyShape(double x, double y){this(x, y, MyColor.Black);}
    public MyShape(MyColor color){this(0, 0, color);}
    public MyShape(){this(0, 0);}
    public double getX() {return x;}
    public void setX(double x) {this.x = x}
    public double getY() {return y;}
    public void setY(double y) {this.y = y;}
    public MyColor getColor() {return color;}
    public void setColor(MyColor color) {this.color = color;}
    public abstract void draw(GraphicsContext gc);
    @Override
    public String toString(){
        return "Class MyShapeis the hierarchy's superclass and inherits the
Java class Object. An\n" +
                "implementation of the class defines a point (x, y) and the
color of the shape. ";
    }
}
```

MyLine.java

```
package shapes;
import javafx.scene.canvas.GraphicsContext;

public class MyLine extends MyShape {
    private double x1, y1;
    private double x2, y2;

    public MyLine(double x1, double y1, double x2, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
    }

    public MyLine(double x1, double y1, double x2, double y2, MyColor
color){
        this(x1, y1, x2, y2);
        setColor(color);
    }

    public double getLength(){
        return Math.sqrt(Math.pow(this.x2 - this.x1, 2) + Math.pow(this.y2
- this.y1,2));
    }

    public double get_xAngle(){
        return Math.toDegrees(Math.atan2((this.y2 - this.y1), (this.x2 -
this.x1)));
    }
```

```
    @Override
    public String toString(){
        return "MyLine{length=" + getLength()+ ", _xAngle=" + get_xAngle()
+ "}";
    }

    @Override
    public void draw(GraphicsContext gc) {
        gc.setStroke(getColor().toFXPaintColor());
        gc.strokeLine(this.x1, this.y1, this.x2, this.y2);
    }
}
```

MyPolygon.java

```
package shapes;
import javafx.scene.canvas.GraphicsContext;

public class MyPolygon extends MyShape {
    private int N;
    private double r;

    public MyPolygon(double x, double y, int N, double r){
        super(x, y);
        this.N = N;
        this.r = r;
    }

    public MyPolygon(double x, double y, int N, double r, MyColor color){
        this(x, y, N, r);
        setColor(color);
    }

    public double getArea(){
        return this.N * Math.pow(this.r, 2) *
Math.tan(Math.toRadians(Math.PI / this.N));
    }

    public double getPerimeter(){
        return this.getSide() * this.N;
    }

    public double getAngle(){
        return (180d * this.N - 360d) / this.N;
    }

    public double getSide(){
        return 2d * this.r * Math.tan(Math.toRadians(Math.PI / this.N));
    }

    public static double getInradiusByCircumradius(double R, int N){
        return R * Math.cos(Math.toRadians(180d / N));
    }

    public static double getCircumradiusByInradius(double r, int N){
```

```
        return r / Math.cos(Math.toRadians(180d / N));
    }

    @Override
    public String toString() {
        return "MyPolygon{" +
                "side=" + getSide() +
                ", angle=" + getAngle() +
                ", perimeter=" + getPerimeter() +
                ", area=" + getArea() +
                '}';
    }

    @Override
    public void draw(GraphicsContext gc){draw(gc, false);}

    public void draw(GraphicsContext gc, boolean isFill) {
        double R = getCircumradiusByInradius(this.r, this.N);
        double innerAngle = 360d / this.N;
        double Xs[] = new double[this.N];
        double Ys[] = new double[this.N];

        for (int i = 0; i < this.N; i++) {
            Ys[i] = getY() + R * Math.sin(Math.toRadians(innerAngle * i -
90));
            Xs[i] = getX() + R * Math.cos(Math.toRadians(innerAngle * i -
90));
        }

        if(isFill){
            gc.setFill(getColor().toFXPaintColor());
            gc.fillPolygon(Xs, Ys, this.N);
        } else {
            gc.setStroke(getColor().toFXPaintColor());
            gc.strokePolygon(Xs, Ys, this.N);
        }
    }
}
```

MyCircle.java

```
package shapes;
import javafx.scene.canvas.GraphicsContext;

public class MyCircle extends MyShape {
    private double r;

    public MyCircle(double x, double y, double r){
        super(x, y);
        this.r = r;
    }

    public MyCircle(double x, double y, double r, MyColor color){
        this(x, y, r);
        setColor(color);
    }
```

```
    public double getArea(){
        return 4 * Math.PI * Math.pow(this.r, 2) / 3;
    }

    public double getPerimeter(){
        return 2 * Math.PI * this.r;
    }

    public double getRadius(){return this.r;}

    @Override
    public void draw(GraphicsContext gc){draw(gc, false);}

    public void draw(GraphicsContext gc, boolean isFill) {
        if(isFill){
            gc.setFill(getColor().toFXPaintColor());
            gc.fillOval(getX() - r, getY()  - r, this.r * 2d, this.r * 2d);
        } else {
            gc.setStroke(getColor().toFXPaintColor());
            gc.strokeOval(getX() - r, getY() - r, this.r * 2d, this.r *
2d);
        }
    }

    @Override
    public String toString() {
        return "MyCircle{" +
                "radius=" + r +
                ", perimeter=" + getPerimeter() +
                ", area=" + getArea() +
                '}';
    }
}
```

MyColor.java

```
package shapes;
import java.awt.*;

public enum MyColor {
    Red(255, 0, 0),
    FireBrick(178,34,34),
    IndianRed(205,92,92),
    LightCoral(240,128,128),
    LightPink(255,182,193),
    Pink(255,192,203),
    Green(0,255,0 ),
    LightGreen(144,238,144),
    PaleGreen(152,251,152),
    OliveDrab(85,107,47),
    MediumAquamarine(0,250,154),
    Turquoise(64,224,208),
    Blue(0,0,255),
    DarkBlue(0,0,139),
    RoyalBlue(65,105,225),
```

```
    SkyBlue(135,206,235),
    Azure(240,255,255),
    White(255,255,255),
    Black(0,0,0),
    Gray(128,128,128),
    LightGray(211,211,211),
    Yellow(255,255,0);

    private int r, g, b;
    private MyColor(){this(0, 0,0);}

    private MyColor(int r, int g, int b){setColor(r, g, b);}

    public Color toAWTColor(){
        return new Color(r, g, b);
    }

    public javafx.scene.paint.Color toFXPaintColor(){
        return javafx.scene.paint.Color.rgb(r, g, b);
    }

    public void setColor(int hex){
        this.r = (hex & 0xFF0000) >> 16;
        this.g = (hex & 0xFF00) >> 8;
        this.b = hex & 0xFF;
    }

    public void setColor(int r, int g, int b){
        this.r = r;
        this.g = g;
        this.b = b;
    }

    public int getHexColor(){
        return ((0xFF0000 & (r << 16)) | (0x00FF00 & (g << 8)) |b);
    }
}
```

Main.java

```
package com.demo;
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
```

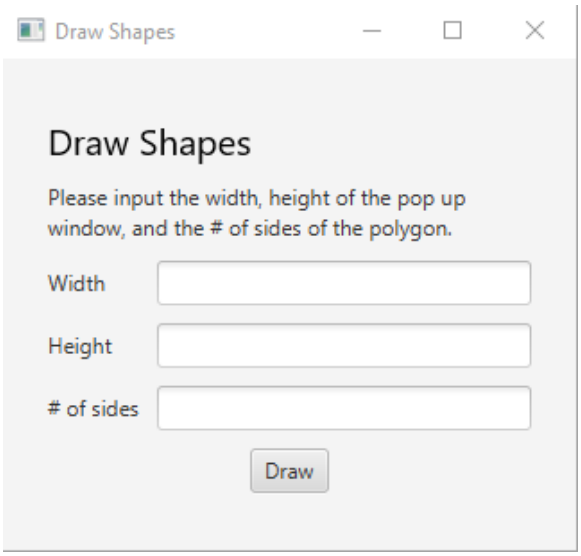
```
import javafx.scene.layout.Pane;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Modality;
import javafx.stage.Stage;
import shapes.MyCircle;
import shapes.MyColor;
import shapes.MyLine;
import shapes.MyPolygon;

public class Main extends Application implements EventHandler<ActionEvent>
{

    private static double MIN_WIDTH = 200d;
    private static double MIN_HEIGHT = 140d;
    private static int DEFAULT_SIDE_NUM = 6;
    private TextField wTextField, hTextField, nTextField;
    private double currentWidth, currentHeight;
    private Scene popScene;
    private Pane popHolder;
    private Canvas drawCanvas;

    public static void main(String[] args) {
        // write your code here
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        // Use grid pane to layout the main stage
        // Use a pop up window to show the shape graphics
        // the main stage should be able to accept the width, height
        //and the number of sides of the polygon from user
        GridPane root = new GridPane();
        root.setAlignment(Pos.CENTER);
        root.setHgap(10);
        root.setVgap(10);
        root.setPadding(new Insets(25,25,25,25));

        // use Text describe the view
        Text title = new Text("Draw Shapes");
        title.setFont(Font.font(20));
        root.add(title, 0, 0, 2, 1);
        // use Label for the description for better warping
        Label description = new Label("Please input the width, height of
the pop up window, and the # of sides of the polygon.");
        description.setWrapText(true);
        root.add(description, 0, 1, 2,1);

        // use TextField to accept the user inputs
        // use Label to give hints to user
        Label wLabel = new Label("Width");
        wTextField = new TextField();
        root.add(wLabel, 0, 2);
        root.add(wTextField, 1, 2);

        Label hLabel = new Label("Height");
        hTextField = new TextField();
```

```
        root.add(hLabel, 0, 3);
        root.add(hTextField,1,3);

        Label nLabel = new Label("# of sides");
        nTextField = new TextField();
        root.add(nLabel, 0, 4);
        root.add(nTextField, 1, 4);

        // A button
        Button btn = new Button("Draw");
        HBox btnBox = new HBox(10);
        btnBox.setAlignment(Pos.CENTER);
        btnBox.getChildren().add(btn);
        root.add(btnBox, 0, 5, 2, 1);
        btn.setOnAction(this);
        Scene scene = new Scene(root, 320, 275);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Draw Shapes");
        primaryStage.show();
    }

    /**
     * This function will return a canvas that have the graph drawn *
@param w the width of the canvas
     * @param h the height of the canvas
     * @param polygonNum the # of the sides of the polygon
     * @return */
  public Canvas getShowCanvas(double w, double h, int polygonNum){
        // keep the height and width properties
        // use for the resize
        currentHeight = h;
        currentWidth = w;

        Canvas canvas = new Canvas(w, h);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        MyLine line1 = new MyLine(0,0, w, 0, MyColor.Black);
        MyLine line2 = new MyLine(0, 0, 0, h, MyColor.Black);
        MyLine line3 = new MyLine(0, h, w, h,  MyColor.Black);
        MyLine line4 = new MyLine(w, 0, w, h, MyColor.Black);
        MyLine diagonal1 = new MyLine(0, 0, w, h, MyColor.Black);
        MyLine diagonal2 = new MyLine(w, 0, 0, h, MyColor.Black);

        double midX = w / 2.0;
        double midY = h / 2.0;
        double outerInradius =
MyPolygon.getInradiusByCircumradius(0.95*Math.min(w, h) / 2d, polygonNum);
        MyPolygon outerStrokeHexagon = new MyPolygon(midX, midY,
polygonNum, outerInradius, MyColor.Black);
        MyPolygon outerFillHexagon = new MyPolygon(midX, midY, polygonNum,
outerInradius, MyColor.Gray);
        MyCircle outerCircle = new MyCircle(midX, midY, outerInradius,
MyColor.Yellow);

        double midInradius =
MyPolygon.getInradiusByCircumradius(outerInradius, polygonNum);
        MyPolygon midHexagon = new MyPolygon(midX, midY, polygonNum,
midInradius, MyColor.LightGreen);
        MyCircle midCircle = new MyCircle(midX, midY, midInradius,
```

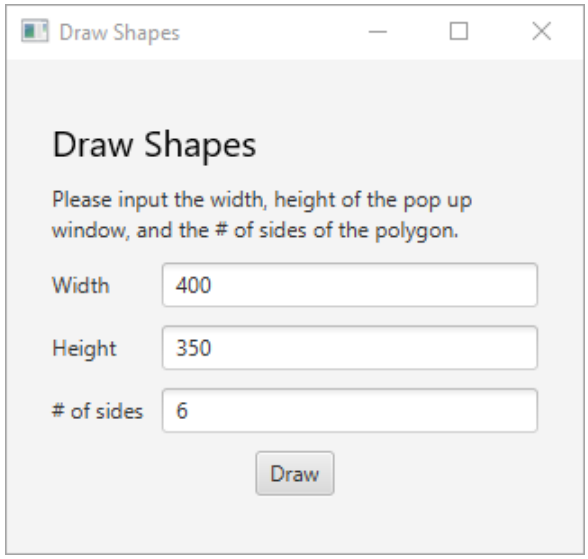
```
MyColor.LightPink);

        double innerInradius =
MyPolygon.getInradiusByCircumradius(midInradius, polygonNum);
        MyPolygon innerHexagon = new MyPolygon(midX, midY, polygonNum,
innerInradius, MyColor.SkyBlue);

        outerFillHexagon.draw(gc, true);
        outerStrokeHexagon.draw(gc);
        outerCircle.draw(gc, true);

        midHexagon.draw(gc, true);
        midCircle.draw(gc, true);

        innerHexagon.draw(gc, true);

        line1.draw(gc);
        line2.draw(gc);
        line3.draw(gc);
        line4.draw(gc);

        diagonal1.draw(gc);
        diagonal2.draw(gc);

        return canvas;
    }

    @Override
    public void handle(ActionEvent event) {
        double w, h;
        int n;

        w = getValueFromInput(wTextField, MIN_WIDTH, MIN_WIDTH);
        h = getValueFromInput(hTextField, MIN_HEIGHT, MIN_HEIGHT);
        n = (int)getValueFromInput(nTextField, 3, DEFAULT_SIDE_NUM);

        final Stage dialog = new Stage();
        dialog.initModality(Modality.NONE);
        popHolder = new Pane();
        drawCanvas = getShowCanvas(w, h, n);
        popHolder.getChildren().add(drawCanvas);
        popScene = new Scene(popHolder, w, h);

        // when the window size changes
        // resize the draw shapes
        popScene.widthProperty().addListener(new ChangeListener<Number>() {
            @Override
            public void changed(ObservableValue<? extends Number>
observable, Number oldValue, Number newValue) {
                popHolder.getChildren().clear();

popHolder.getChildren().add(getShowCanvas(newValue.doubleValue(),
currentHeight, n));
                wTextField.setText(String.valueOf(newValue.doubleValue()));
            }
        });

        popScene.heightProperty().addListener(new ChangeListener<Number>()
```

```
{
            @Override
            public void changed(ObservableValue<? extends Number>
observable, Number oldValue, Number newValue) {
                popHolder.getChildren().clear();
                popHolder.getChildren().add(getShowCanvas(currentWidth,
newValue.doubleValue(), n));
                hTextField.setText(String.valueOf(newValue.doubleValue()));
            }
        });

        dialog.setScene(popScene);
        dialog.setTitle("Finish!");
        dialog.show();
    }

    /**
     * this function * @param input the input TextField object
     * @param condition the smallest value of a correct input should be
     * @param defaultValue the default value.
     * @return the value of text in the TextField object if user has the
correct input
     * default value if the input cannot be recognized the input * or the
input value is less than the condition value */
     private double getValueFromInput(TextField input, double condition,
double defaultValue){
        double result;
        try {
            result = Double.parseDouble(input.getText());
            if( result < condition){
                result = defaultValue;
            }
            input.setText(String.valueOf(result));
        } catch (Exception e){
            result = defaultValue;
            input.setText(String.valueOf(result));
        }
        return result;
    }
}
```

Outputs

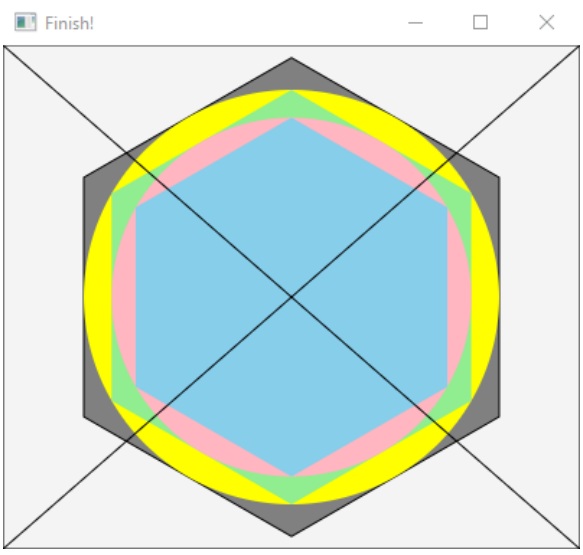


Input window

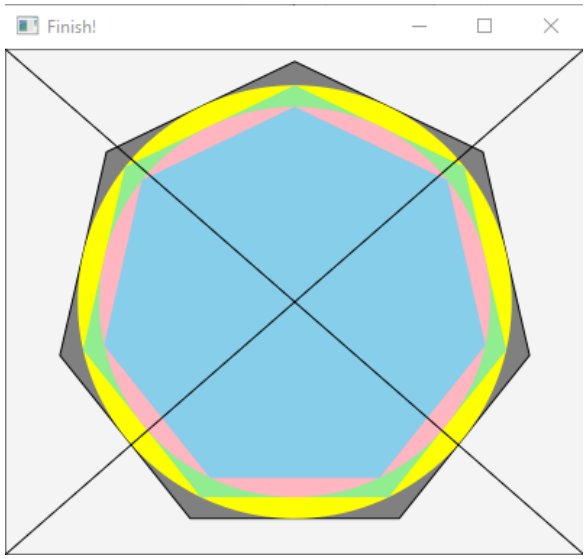


Input window filled

The pictures above are the window that let the user type in the width, height and the number of sides of the polygon. The minimum width is 200 pixels and the minimum height is 140 pixels. If a user enter a number less than the minimum height/width or a invalid input such as words, the application will set the height/width to the default value. Otherwise, the input value will be set. The minimum number of sides is 3 but the default number of sides is 6. If a user input a value less than the minimum number of sides or an invalid number, default number of sides will be set. Otherwise, the input value will be accepted.



polygon with 6 sides



polygon with 7 sides

The application draws the graph can fit into the window. It has spacing around the largest polygon so that it will not touch the edge of the window. Also, the application can draw polygons with other number of sides, not just hexagon.