Project 2 – Shapes GUI JavaFX

Ryan A Burkhardt

UMGC

CMSC 335

Professor Ghosh

10Jun2024

JavaFX – Approach

For my approach to this project, I started by reading all required documentations in the Project 2 pdf. I then went into the Project 1 folder to refresh about the calculations which I would be needing for this project. I sat down and finished filling in the required documentation on the class diagram provided in the example which helped me when defining the abstract class for the shape. For this project I will provide some screenshots as rough drafts where I designed the shapes one line at a time. While this was a much more time-consuming approach with my knowledge of the graphics context was limited so I took my time with the program. I used a fill for the two-dimensional shapes while using a line drawing method for the three-dimensional shapes. I also noticed in Project 1 we had a shapes class which I used to add more modularity into this week’s project as well.

**JavaFX – Assumptions**

The assumptions I have made in this assignment is that the user has basic knowledge of programming and will have an IDE to launch the application. This week’s assumptions were far fewer because of the Gui application and that they will have to select from choices I have given them. It is also assumed that the user knows how to launch the application given the package format it is in.

**JavaFX – Not Implemented**

For the non-implemented methods I would have liked to add a text box that displayed to the user which shape is selected and the dimensions that followed each side of the shape. Another implementation I would like to have is the standalone application front. This is something I have never done in any of the classes but am very interested in learning how to do it so the user can just double click an icon and have the gui appear like a normal application. Lastly, I did struggle with adding color and shading to the objects. I would like to revisit this application and add a light gray or similar color that is transparent, so the shape is more defined.

**Polymorphism/Inheritance OOP – User Guide**

The user guide is quite simple for the application this week. All the users must do is load the file into their ide from the zipped package. Next the vmArgs are specific to my computer so the vmArgs will have to be updated to the users SDK so that it knows the java program has the referenced libraries of JavaFX. After the launch json file has been updated with the appropriate vmArgs it can be ran by hitting the launch from the ide. The user is to then select the appropriate shape they would like to have displayed. Then the user must enter an input for the dimension they would like to display. Lastly, the user selects draw and the shape will be displayed.

**JavaFX – Lessons Learned**

Lessons I learned in this project were that I simply had to little experience behind the drawing methods for JavaFX. I have used turtle multiple times and can use it fluently which helped when I saw similarities such as the fill, stroke, color, etc. This was a challenge because the difference in turtle is simply picking up the pen and moving it to the next location. In this program defining all the coordinates was a super hard challenge especially when it came to having appropriate scaling for the shapes. I noticed to many times where the dimensions would ruin my lines completely and the shape was not scalable.

**Polymorphism/Inheritance OOP – Possible improvements**

For possible improvements I would start with implementing more user-friendly interfaces. This would include an hbox and a larger canvas. I would also incorporate color into the equation where the user would have the ability to select the color from a color wheel and it would scale the translucent behavior and shading appropriately to have darker faces in the background or inner dimensions and lighter in the outer dimensions. Lastly, error handling text box to let the user know if they did do something wrong such as not inputting a dimension before hitting draw.

JavaFX – UML

A screenshot of a computer

Description automatically generated

JavaFX – Source Code

/\*\*

\* App Class

\* 18May2024

\* Ryan Burkhardt

\* This class is the main application class for a JavaFX-based shape drawing program.

\* It creates a GUI with a combo box to select the shape type, a combo box to select the shape size,

\* and a button to draw the shape. The shape is drawn on a canvas in the GUI.

\*/

package com.project2;

//import all necessary javafx packages

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.canvas.Canvas;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Label;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

public class Project2 extends Application {

// GUI components

private ComboBox<String> shapeComboBox; // combo box to select the shape type

private ComboBox<Double> sizeComboBox; // combo box to select the size

private Button drawButton; // button to draw the shape

private Canvas canvas; // canvas to draw on

@SuppressWarnings("exports")//Suppress our exports warnings for error handling. I had to do this because it would not run unless suppressed

@Override

public void start(Stage primaryStage) {

// create the GUI components

shapeComboBox = new ComboBox<>();

shapeComboBox.getItems().addAll("Circle", "Square", "Triangle", "Rectangle", "Cone", "Cube", "Cylinder", "Sphere", "Torus"); // add shape types to the combo box

sizeComboBox = new ComboBox<>();

sizeComboBox.getItems().addAll(10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0); // add size options to the combo box

drawButton = new Button("Draw"); // create the draw button

drawButton.setOnAction(e -> drawShape()); // set the button's action to draw the shape

canvas = new Canvas(400, 400); // create the canvas

// create the GUI layout

VBox root = new VBox(10);

root.setPadding(new Insets(10));

root.getChildren().addAll(new Label("Select Shape:"), shapeComboBox, new Label("Select Size:"), sizeComboBox, drawButton, canvas);

// create the scene and stage

Scene scene = new Scene(root, 420, 500);

primaryStage.setTitle("Burkhardt Project 2");

primaryStage.setScene(scene);

primaryStage.show();

}

// method to draw the shape

private void drawShape() {

String shapeType = shapeComboBox.getValue(); // get the selected shape type

double size = sizeComboBox.getValue(); // get the selected size

Shape shape = null; // create a Shape object based off the users selection from the drop box using a switch case

switch (shapeType) {

case "Circle":

shape = new Circle(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Circle object

break;

case "Square":

shape = new Square(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Square object

break;

case "Triangle":

shape = new Triangle(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Triangle object

break;

case "Rectangle":

shape = new Rectangle(canvas.getWidth() / 2, canvas.getHeight() / 2, size, size); // create a Rectangle object

break;

case "Cone":

shape = new Cone(canvas.getWidth() / 2, canvas.getHeight() / 2, size, size); // create a Cone object

break;

case "Cube":

shape = new Cube(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Cube object

break;

case "Cylinder":

shape = new Cylinder(canvas.getWidth() / 2, canvas.getHeight() / 2, size, size); // create a Cylinder object

break;

case "Sphere":

shape = new Sphere(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Sphere object

break;

case "Torus":

shape = new Torus(canvas.getWidth() / 2, canvas.getHeight() / 2, size); // create a Torus object

break;

default:

break;

}

GraphicsContext gc = canvas.getGraphicsContext2D(); // get the GraphicsContext

gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight()); // clear the canvas that way the previous objects are not shown

shape.draw(gc); // draw the shape on the canvas

}

public static void main(String[] args) {

launch(args); // launch the JavaFX application

}

}

/\*\*Shape Class

\* 18May2024

\* Ryan Burkhardt

\* This class will be used to make a shape that passes in a simple 4 variable shape

\* It will be used in all of my shapes whether 2-d or 3-d

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

//Create an abstract class so it can be used for 3-d and 2-d shapes

//The x and y will be the dimensions as well as the location on the canvas

public abstract class Shape {

private double x;

private double y;

private double width;

private double height;

//Construct our shape with the setters

public Shape(double x, double y, double width, double height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

//Provide all getters to use in our shape code.

public double getX() {

return x;

}

public double getY() {

return y;

}

public double getWidth() {

return width;

}

public double getHeight() {

return height;

}

public abstract void draw(GraphicsContext gc);

}

/\*\*

\* Circle Class

\* 18May2024

\* Ryan Burkhardt

\* This class represents a 2D circle shape that can be drawn on a canvas.

\* It uses the shapes position on the canvas as well as the radius.

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

//Extend our shape class to add the circle

public class Circle extends Shape {

// constructor to initialize the circle's position and radius

public Circle(double x, double y, double radius) {

super(x, y, radius, radius); // call the Shape constructor with the diameter

}

// implementation of the draw method for a circle

@Override

public void draw(GraphicsContext gc) {

gc.fillOval(getX() - getWidth() / 2, getY() - getHeight() / 2, getWidth(), getHeight()); // draw a filled oval (circle) at the specified position and size

}

}

/\*\*Cone Class

\* 18May2024

\* Ryan Burkhardt

\* This class represents a 3D cone shape that can be drawn on a canvas.

\* It takes in the getWidth() and getHeight() from the shape class as well as the location on canvas

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.shape.ArcType;

// Cone.java

public class Cone extends Shape {

// constructor to initialize the cone's position, getWidth(), and getHeight()

public Cone(double x, double y, double radius, double height) {

super(x, y, radius, height); // call the Shape constructor with the specified width and getHeight()

}

// implementation of the draw method for a 3D cone

@Override

public void draw(GraphicsContext gc) {

// draw the bottom half circle

gc.setFill(javafx.scene.paint.Color.BLACK);

gc.setLineDashes(10); // set the dash length to 10 units

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth(), getWidth()/2, 0.0, 180.0, ArcType.OPEN);

gc.setLineDashes(0.0);

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth(), getWidth()/2, 180.0, 180.0, ArcType.OPEN);

gc.setStroke(javafx.scene.paint.Color.BLACK);

gc.setLineWidth(2);

// draw lines from the middle of the lower circle's edge to the centerpoint of the upper circle

gc.strokeLine(getX(), getY()-getWidth()/4, getX()-getWidth()/2, getY() - getHeight());

gc.strokeLine(getX() - getWidth(), getY()-getWidth()/4, getX()-getWidth()/2, getY() - getHeight());

}

}

/\*\*

\* Cube Class

\* 18May2024

\* Ryan Burkhardt

\* This class represents a 3D cube shape that can be drawn on a canvas.

\* It provides methods for drawing the cube, including the front face and the lines connecting it.

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;

public class Cube extends Shape {

// constructor to initialize the cube's position and size

public Cube(double x, double y, double size) {

super(x, y, size, size); // call the Shape constructor with the size

}

// implementation of the draw method for a cube

@Override

public void draw(GraphicsContext gc) {

// draw the front face of the cube

gc.setStroke(Color.BLACK);

gc.setLineWidth(2);

gc.strokeRect(getX(), getY(), getWidth(), getHeight());

// draw the lines to make the cube three dimensional

gc.strokeLine(getX(), getY(), getX()+getWidth()/2, getY()-getHeight()/2);

gc.strokeLine(getX()+getWidth(), getY() + getHeight(), getX() + getWidth()\*1.5, getY()+getHeight()/2);

gc.strokeLine(getX()+getWidth(), getY(), getX()+getWidth()\*1.5, getY()-getHeight()/2);

gc.strokeLine(getX()+getWidth()/2, getY()-getHeight()/2, getX()+getWidth()\*1.5, getY()-getHeight()/2);

gc.strokeLine(getX()+getWidth()\*1.5, getY()-getHeight()/2, getX()+getWidth()\*1.5, getY()+getHeight()/2);

}

}

/\*\*

\* Cylinder Class

\* 18May2024

\* Ryan Burkhardt

\* This class represents a 3D cylinder shape that can be drawn on a canvas.

\* It provides methods for drawing the cylinder, including the top and bottom elipses,

\* as well as the lines connecting them.

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.shape.ArcType;

// Cylinder.java

public class Cylinder extends Shape {

// constructor to initialize the cylinder's position, radius, and height

public Cylinder(double x, double y, double radius, double height) {

super(x, y, radius, height); // call the Shape constructor with the specified width and height

}

// implementation of the draw method for a 3D cylinder where the getWidth() method is equal to the radius and getHeight() is equal to the height

@Override

public void draw(GraphicsContext gc) {

// draw the bottom elipse by dividing the radius by 2 for the height of the arc and the radius for the width

gc.setLineWidth(2);//Set our line width to 2

gc.setFill(javafx.scene.paint.Color.BLACK);//set stroke to black

gc.setLineDashes(10); // set the dash length to 10 units

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth(), getWidth()/2, 0.0, 180.0, ArcType.OPEN);//draw our top dashed arc

gc.setLineDashes(0.0);//remove the dashes

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth(), getWidth()/2, 180.0, 180.0, ArcType.OPEN);//draw the bottom of the arc

// draw lines from the middle of the lower circle's edge to the edge of the upper circle

gc.strokeLine(getX() - getWidth(), getY() - getWidth()/4, getX() - getWidth(), getY() - getHeight());

gc.strokeLine(getX(), getY()-getWidth()/4, getX(), getY() - getHeight());

//draw the top elipse using the same method as the bottom but with no dashed lines

gc.strokeArc(getX() - getWidth(), getY()-getHeight()-getWidth()/4, getWidth(), getWidth()/2, 360.0, 360.0, ArcType.OPEN);

}

}

/\*\*Rectangle Class

\* 18May2024

\* Ryan Burkhardt

\* This class represents a 2D rectangle shape that can be drawn on a canvas.

\* It provides methods for drawing the rectangle, including the top and bottom edges,

\* as well as the left and right sides.

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

//Extend our shape class and use the super method to bring in the dimensions

public class Rectangle extends Shape {

// constructor to initialize the rectangle's position and size

public Rectangle(double x, double y, double width, double height) {

super(x, y, width, height); // call the Shape constructor with the specified width and height

}

// implementation of the draw method for a rectangle using shapes getter methods

@Override

public void draw(GraphicsContext gc) {

// draw a filled rectangle using the fillrect from graphics context

gc.fillRect(getX()-getWidth()/2, getY()-getHeight()/2, getWidth(), getHeight()/2);

}

}

/\*\*Sphere Class

\* 18May2024

\* Ryan Burkhardt

\* This class will be used to make a 3D sphere shape that passes in a simple int of number of dimensions

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.paint.Color;

import javafx.scene.shape.ArcType;

// Sphere.java

public class Sphere extends Shape {

// constructor to initialize the sphere's position and radius

public Sphere(double x, double y, double radius) {

super(x, y, radius \* 2, radius \* 2); // call the Shape constructor with the specified width and height

this.radius = radius;

}

private double radius;

// implementation of the draw method for a 3D sphere

@Override

public void draw(GraphicsContext gc) {

// Draw the sphere

gc.setStroke(Color.BLACK);

gc.setLineWidth(2);

gc.setLineDashes(0);

gc.strokeOval(getX() - radius, getY() - radius, radius \* 2, radius \* 2);

// Draw horizontal arc for 3D effect

gc.strokeArc(getX() - radius, getY()-radius/2, radius \* 2, radius, 180, 180,ArcType.OPEN);

// Draw vertical arc for 3D effect

gc.strokeArc(getX()-radius/2, getY() - radius, radius, radius\*2, 90, 180,ArcType.OPEN);

// Draw horizontal arc for 3D effect

gc.setLineDashes(10);

gc.strokeArc(getX() - radius, getY()-radius/2, radius \* 2, radius, 0, 180,ArcType.OPEN);

// Draw vertical arc for 3D effect

gc.strokeArc(getX()-radius/2, getY() - radius, radius, radius\*2, 270, 180,ArcType.OPEN);

gc.setLineDashes(0);//needed to add this because it was throwing off the apps and making everything dashed

}

}

/\*\*Square Class

\* 18May2024

\* Ryan Burkhardt

\* This class will be used to make a shape that passes in a simple int of number of dimensions

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

// Square.java

public class Square extends Shape {

// constructor to initialize the square's position and size

public Square(double x, double y, double size) {

super(x, y, size, size); // call the Shape constructor with the same width and height

}

// implementation of the draw method for a square

@Override

public void draw(GraphicsContext gc) {

gc.fillRect(getX()-getWidth()/2, getY()-getHeight(), getHeight(), getWidth()); // draw a filled rectangle ( square ) at the specified position and size

}

}

/\*\*Sphere Class

\* 18May2024

\* Ryan Burkhardt

\* This class will be used to make a 3D sphere shape that passes in a simple int of number of dimensions

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

import javafx.scene.shape.ArcType;

// Sphere.java

public class Torus extends Shape {

// constructor to initialize the sphere's position and radius

public Torus(double x, double y, double minorRadius) {

super(x, y, minorRadius, minorRadius); // call the Shape constructor with the specified width and height

}

// implementation of the draw method for a 3D sphere

@Override

public void draw(GraphicsContext gc) {

// draw the bottom elipse by dividing the radius by 2 for the height of the arc and the radius for the width

gc.setLineWidth(2);//Set our line width to 2

gc.setFill(javafx.scene.paint.Color.BLACK);//set stroke to black

gc.strokeArc(getX() - getWidth()/2, getY()-getWidth()/5, getWidth(), getWidth()/4, 0.0, 180.0, ArcType.OPEN);//draw our top dashed arc

gc.strokeArc(getX() - getWidth()/2, getY()-getWidth()/5, getWidth(), getWidth()/4, 180.0, 180.0, ArcType.OPEN);//draw the bottom of the arc

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth()\*2, getWidth(), 0.0, 180.0, ArcType.OPEN);//draw our top dashed arc

gc.strokeArc(getX() - getWidth(), getY() - getWidth()/2, getWidth()\*2, getWidth(), 180.0, 180.0, ArcType.OPEN);//draw the bottom of the arc

gc.strokeArc(getX() - getWidth()/1.34, getY()-getWidth()/8, getWidth()/4, getWidth()/8, 0.0, 90.0, ArcType.OPEN);//draw our top dashed arc

gc.strokeArc(getX() + getWidth()/1.34 - getWidth()/4, getY()-getWidth()/8, getWidth()/4, getWidth()/8, 180, -90.0, ArcType.OPEN);//draw our mirrored top dashed arc

}

}

/\*\*Triangle Class

\* 18May2024

\* Ryan Burkhardt

\* This class will be used to make a triangle by extending the shape class

\* it takes in the graphic context object gc to draw a filled in polygon shape as the triangle.

\*/

package com.project2;

import javafx.scene.canvas.GraphicsContext;

// Extend our Shape class to add the Triangle using the super methods

public class Triangle extends Shape {

// constructor to initialize the triangle's position and size

public Triangle(double x, double y, double size) {

super(x, y, size, size); // call the Shape constructor with the same width and height

}

// implementation of the draw method for a triangle

@Override

public void draw(GraphicsContext gc) {

// draw a filled triangle using the polygon methods

gc.fillPolygon(new double[]{getX()-getWidth()/2, (getX() + getWidth())-getWidth()/2, (getX() + getWidth() / 2)-getWidth()/2}, new double[]{getY(), getY(), getY() - getHeight()}, 3);

}

}

Polymorphism/Inheritance OOP – Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Purpose of Test | Positive or Negative Test | Input Values | Expected Result | Pass/Fail |
| 1 | Test that the GUI does not crash if only the shape is inputted from drop down but not dimensions when hitting draw | Negative | Rectangle  Draw | Should do nothing but not crash | Pass |
| 2 | Test that the GUI does not crash if only the dimension is inputted from drop down but not shape when hitting draw | Negative | 100.0  draw | Should do nothing but not crash | Pass |
| 3 | Test that resizing does not affect drawing canvas as it is not resiazable. | Negative | Rectangle  100.0  Draw  resize | Should resize window but gui should remain the same size | Pass |
| 4 | Construct a Circle to test class and switch | Positive | Shape = Circle  Dimension = 10  Draw  Shape = Circle  Dimension = 100  Draw | Circle should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 5 | Construct a Cone to test class and switch | Positive | Shape = Cone  Dimension = 10  Draw  Shape = Cone  Dimension = 100  Draw | Cone should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 6 | Construct a Cube to test class and switch | Positive | Shape = Cube  Dimension = 10  Draw  Shape = Cube  Dimension = 100  Draw | Cube should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 7 | Construct a Cylinder to test class and switch | Positive | Shape = Cylinder  Dimension = 10  Draw  Shape = Cylinder  Dimension = 100  Draw | Cylinder should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 8 | Construct a Rectangle test class and switch | Positive | Shape = Rectangle  Dimension = 10  Draw  Shape = Rectangle  Dimension = 100  Draw | Rectangle should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 9 | Construct a Sphere to test class and switch | Positive | Shape = Sphere  Dimension = 10  Draw  Shape = Sphere  Dimension = 100  Draw | Sphere should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 10 | Construct a Square to test class and switch | Positive | Shape = Square  Dimension = 10  Draw  Shape = Square  Dimension = 100  Draw | Square should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. | Pass |
| 11 | Construct a Torus to test class and switch | Positive | Shape = Torus  Dimension = 10  Draw  Shape = Torus  Dimension = 100  Draw | Torus should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. |  |
| 12 | Construct a Triangle to test class and switch | Positive | Shape = Triangle  Dimension = 10  Draw  Shape = Triangle  Dimension = 100  Draw | Triangle should be created, and displayed in the gui for both small and large dimension as well as canvas should be cleared each time the draw button is hit. |  |

JavaFX – Test Cases Screenshots

Test # 1

A screenshot of a computer

Description automatically generated

Test # 2

A screenshot of a computer

Description automatically generated

Test # 3

A screenshot of a computer

Description automatically generated

Test # 4

A screenshot of a computer

Description automatically generated

Test # 5

A screenshot of a computer

Description automatically generated

Test # 6

A screenshot of a computer

Description automatically generated

Test # 7

A screenshot of a computer

Description automatically generated

Test # 8

A screenshot of a computer

Description automatically generated

Test # 9

A screenshot of a computer

Description automatically generated

Test # 10

A screenshot of a computer

Description automatically generated

Test # 11

A screenshot of a computer

Description automatically generated

Test # 12

A screenshot of a computer

Description automatically generated