Chapter 6, Case Study: BallWorld

John M. Morrison

December 13, 2019

Contents

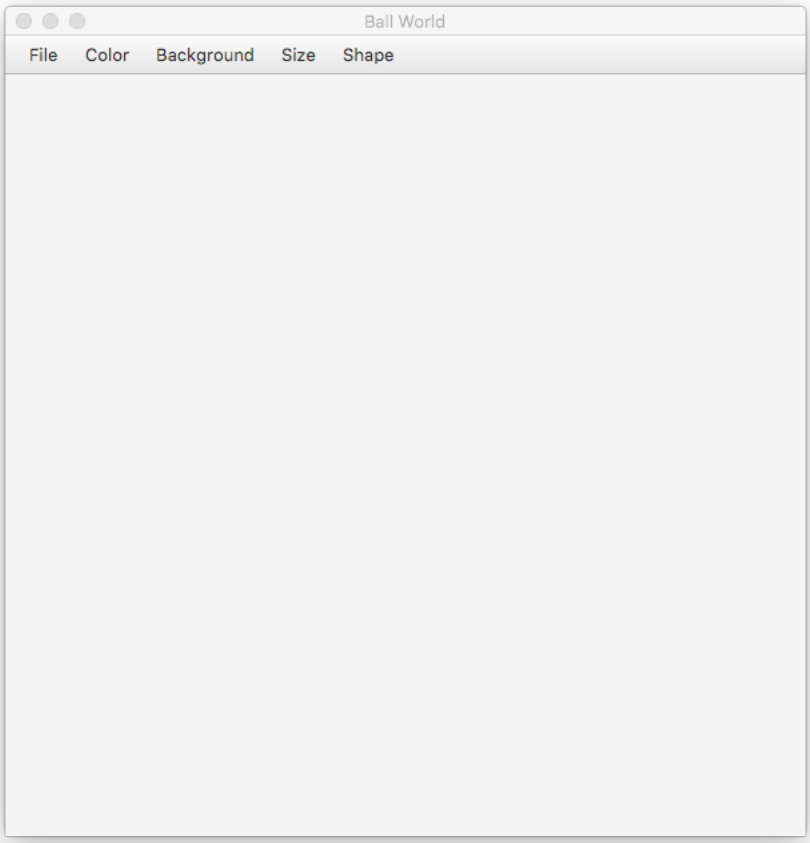
0 Introduction

In this chapter and the next, we will produce our first GUI application that does a nontrivial job and which can save object state. In this chapter, we will build the application make it fully responsive to user-generated events. In the next, we will explore exception handling and FileIO. We will use FileIO to create files

that save our work so we can reëdit it or so we can save it into various formats such as `.jpeg` or `.png`. First, we need to discuss the keyword `final` in a couple of contexts. With that out of the way, we shall begin to build `BallWorld` and see what we have been discussing pop to life.

1 Making our first big app: `BallWorld`

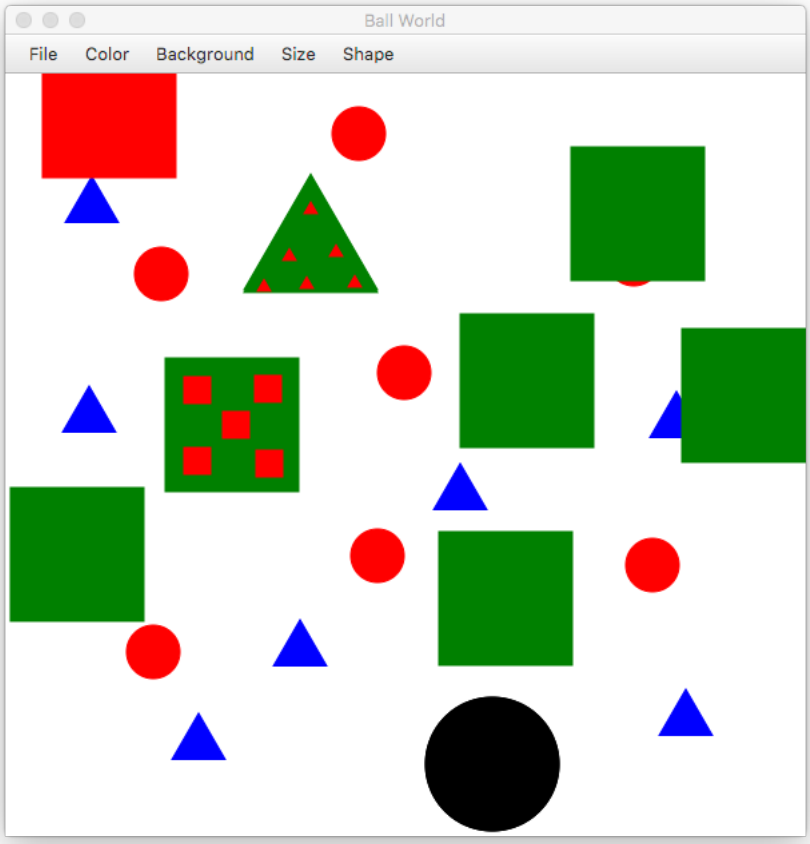
This application be built in this chapter and the next as we bring things online that will help us to do the job. We will go as far as we can with existing machinery, and add things in as we move along. Here is what you will see by the end of this chapter.



Here is what is in the menus. The `File` menu just contains a menu item

to `quit`. The `Color` and `Background` menus contain items that pick the color for the next shape to be drawn and for the color of the background. The `Size` menu controls the size of the next shape to be drawn. The `Shape` menu allows you to choose a ball, a square or a triangle.

As we proceed, we will use some of the stuff we already know, and introduce new classes and new ideas as needed. Let us now show some artwork generated by this application.



We now get to the business of building the application.

2 Putting Menus in the Window and Getting Started

Begin by creating this class. Compile and run. This will just create a blank window with a title in it.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
public class BallWorld extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Ball World");
        primaryStage.show();
    }
}
```

Now let us ask what our class needs to know to do its job. Here are some items we can easily think of.

1. We need to know the color to paint the background.
2. We need to know the color of the next shape
3. We need to know the size of the next shape
4. We need to know the type of the next shape: triangle/square/ball.

These would seem to be state variables. We shall insert them into our class and initialize them. Let us get things going, save for the current shape, which will require a little more machinery. Let us set up all of the menus. Here we will do this in a separate function, `makeMenus`. This will return a `MenuBar` that has all of the menus placed in it.

These imports are needed.

```
import javafx.scene.paint.Color;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
```

You should think of `start` as the conductor in the pit; it should call other methods to do most of its work. This keeps code for different parts of the app neatly organized into their own places, and it prevents `start` from becoming horridly bloated.

We add an `init()` method and a constructor. These gets called first, and can be used to initialize any state variables. Never put stages, scenes, or event handling code in `init()`.

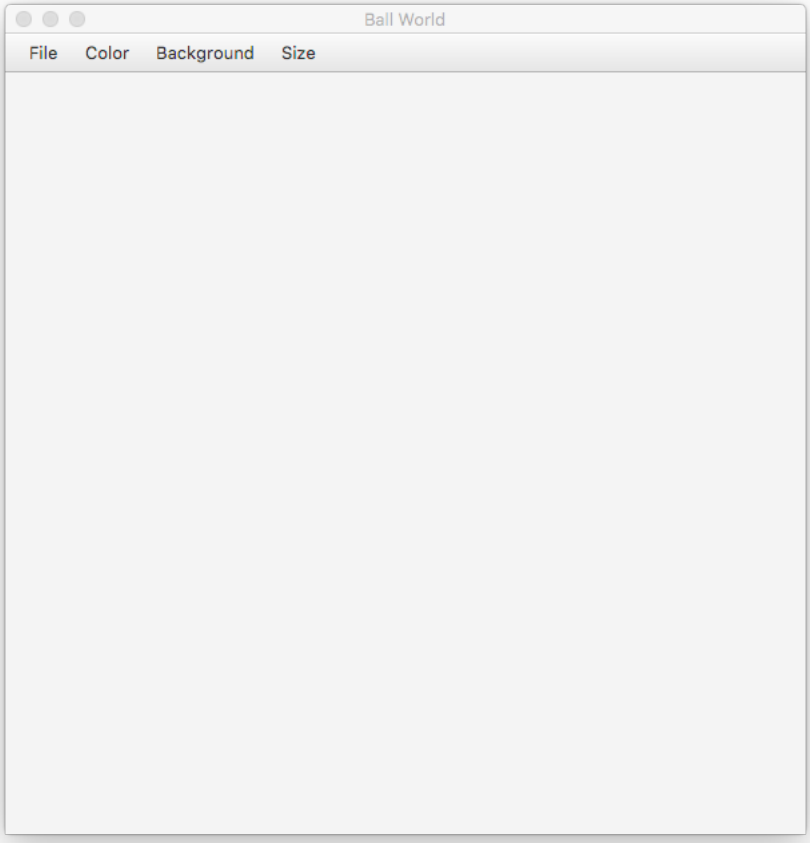
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;

public class BallWorld extends Application
{
    private Color backgroundColor;
    private Color currentColor;
    private int currentSize;
    public BallWorld()
    {
        backgroundColor = Color.WHITE;
        currentColor = Color.BLACK;
        currentSize = 50;
    }
    public void init()
    {
    }
    public static void main(String[] args)
    {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Ball World");
        BorderPane bp = new BorderPane();
        bp.setTop(makeMenus());
        primaryStage.setScene(new Scene(bp, 500,500));
        primaryStage.show();
    }
```

```
    private MenuBar makeMenus()
    {
        MenuBar mbar = new MenuBar();
        //create and insert menus
        Menu fileMenu = new Menu("File");
        Menu colorMenu = new Menu("Color");
        Menu backgroundMenu = new Menu("Background");
        Menu sizeMenu = new Menu("Size");
        mbar.getMenus().addAll(fileMenu, colorMenu, backgroundMenu, sizeMenu);
        return mbar;
    }
}
```

Now compile and run and you will see this. We can begin to see the outline of our app take shape.



3 Introducing Canvas

The JavaFX GUI framework provides the object `Canvas` as a surface for drawing. We will create a method called `refresh`, which will redraw our work when we ask it to. To do this, add the following imports.

```
import javafx.canvas.Canvas;
import javafx.canvas.GraphicsContext;
```

Then add this state variable

```
private Canvas canvas;
```

In the start method create the canvas

```
canvas = new Canvas(500,500);
```

In the start method put the canvas in the center and modify the setting of the scene as shown. Since we gave the canvas a size, there is no need to give the scene as size as well.

```
bp.setCenter(canvas)
primaryScene.setScene(new Scene(bp));
```

Finally, add this new method

```
private void refresh()
{
    GraphicsContext2D g = canvas.getGraphicsContext();
    g.setFill(color.white);
    g.fillRect(0, 0, getWidth(), getHeight());
}
```

right after setting the center in `bp` add this line

```
refresh(canvas);
```

you will have a nice white canvas to begin with.

`what is this GraphicsContext?` Your canvas comes with a pen/paintbrush that can render 16,777,216 colors. this is the graphics context. it knows how to draw lots of things; we will shortly begin to explore how it works.

4 Adding Shapes

Now we will create a second class named `Ball`. This class will define the ball shape and it will tell these balls how to *draw themselves*. We begin by asking: what does a ball need to know to draw itself?

1. A ball needs to know its location; when the canvas is clicked, the ball will draw with its center at the location of the mouse-click.
2. A ball needs to know its size.
3. A ball needs to know its color.

With an eye toward the future, we are going to store our ball's color as three doubles and its point by its coördinates. This will facilitate storing our Ball objects in a file. We will also provide methods for retrieving a ball's center and color.

So, let us make the file `Ball.java` and begin laying out our class as follows. Note the introduction of the `Point2D` class that represents points in the plane as ordered pairs of `double`s. let us make the state variables final since, once a ball is created, we will not change any of its properties.

The `GraphicsContext` pen has some useful methods we show here.

1. `setFill(Color c)` this sets the pen fill color to the color `c`.
2. `fillOval(double ulx, double uly, double width, double height)` this fills an oval whose upper left hand coördiantes are `(ulx, uly)` with height `height` and width `width`.

We also add in an import for the graphics context and we have the following.

```
import javafx.scene.paint.Color;
import javafx.geometry.Point2D;
public class Ball
{
    final private double x;
    final private double y;
    final private double red;
    final private double green;
    final private double blue;
    final private double radius;
        public Ball(Point2D center, Color color, double radius)
        {
```

```
        this.red = color.getRed();
        this.green = color.getGreen();
        this.blue = color.getBlue();
        this.x = center.getX();
        this.y = center.getY();
            this.radius = radius;
        }
    public Color getColor()
    {
        return Color.color(red, green, blue);
    }
    public Point2D getCenter()
    {
        return new Point2D(x,y);
    }
}
```

Take note of the fact that these private value of red, green, and blue are `doubles` between 0 and 1. The static method `Color.color(double double, double)` does exactly what we want it to. Now we get the balls into `BallWorld`. to do this, let us avail ourselves of the wonderfully expandable `ArrayList`. add this state variable

```
private ArrayList<Ball> shapes;
```

in the `init()` method initialize as follows.

```
shapes = new ArrayList<>();
```

make sure you do this import, too

```
import java.util.ArrayList;
```

next, we will tell all of our balls to draw themselves in the `refresh()` method as follows.

```
for(Ball b: shapes)
{
    b.draw(g);
}
```

our `refresh` method becomes

```
private void refresh(Canvas canvas)
{
```

```
    GraphicsContext g = canvas.getGraphicsContext2D();
    g.setFill(backgroundColor);
    g.fillRect(0,0, canvas.getWidth(),  canvas.getHeight());
    for(Ball b: shapes)
    {
        b.draw(g);
    }
}
```

5 Getting Balls to Appear When the Mouse is Clicked

look in the `canvas` API page. we need to set the canvas to react when the mouse is clicked. you will use a method inherited from ancestor class `node`, `setOnMouseClicked`. here is its method detail

```
public final void setOnMouseClicked(EventHandler<? super MouseEvent> value)
```

sets the value of the property onMouseClicked.

property description:

defines a function to be called when a mouse button has been clicked (pressed and released) on this node.

so, this function expects an event handler instance for a MouseEvent; note that this is a functional interface. as a result, we can handle this event using a lambda. so, we will do this.

```
canvas.setOnMouseClicked( e ->
    //add a ball to the array list at this location
    //the ball's color is the current color.
    //refresh the canvas.
)};
```

now we code it. look under the `MouseEvent` page. the event bears information about where it occurred in the canvas object via its methods `getX()` and `getY()`. so we proceed as follows.

```
canvas.setOnMouseClicked( e ->
    shapes.add(new Ball(new Point2D(e.getX(),e.getY()),
        currentSize, currentColor);
    refresh();
)};
```

Note that the `refresh` method takes care of the new ball being put to the canvas. Now, when you compile and run, you will see balls appearing on the canvas. As of yet, we have not given ourselves control over the size or the color of the balls. The background remains white and the balls are Henry ford black (you have no choice).

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.geometry.Point2D;
import java.util.ArrayList;

public class BallWorld extends Application
{
    private Color backgroundColor;
    private Color currentColor;
    private int currentSize;
    private ArrayList<Ball> shapes;
    public BallWorld()
    {
        backgroundColor = Color.WHITE;
        currentColor = Color.BLACK;
        currentSize = 50;
        shapes = new ArrayList<>();
    }
    public void init()
    {
    }
    public static void main(String[] args)
    {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Ball World");
        BorderPane bp = new BorderPane();
        Canvas canvas = new Canvas(500,500);
```

```
        canvas.setOnMouseClicked(e ->
        {
            Ball b = new Ball(new Point2D(e.getX(),e.getY()),
                currentColor, currentSize);
            shapes.add(b);
            refresh(canvas);
        });
        bp.setTop(makeMenus());
        bp.setCenter(canvas);
        primaryStage.setScene(new Scene(bp));
        primaryStage.show();
    }
    private MenuBar makeMenus()
    {
        MenuBar mbar = new MenuBar();
        //create and insert menus
        Menu fileMenu = new Menu("File");
        Menu colorMenu = new Menu("Color");
        Menu backgroundMenu = new Menu("Background");
        Menu sizeMenu = new Menu("Size");
        mbar.getMenus().addAll(fileMenu, colorMenu, backgroundMenu, sizeMenu);
        return mbar;
    }
        private void refresh(Canvas canvas)
        {
            GraphicsContext g = canvas.getGraphicsContext2D();
            g.setFill(backgroundColor);
            g.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
        for(Ball b: shapes)
        {
            b.draw(g);
        }
        }
}
```

6 Quel Est Le Menu?

Now let us gain control over the color of the next ball to be created. The user expects this. He will choose a color from the menu and nothing happens right away. then, when he makes the next click, a ball of the chosen color appears.

There are three important components of menus, the menu bar houses the menus; we have seen this. the third component is menu items. Menu items are really just buttons; they have as a button has, a `setOnAction` method. so the

process of enabling a menu item is the same as that of enabling a button.

So, let's see what goes into making a menu item for choosing red for our next ball. Clearly, this is in the cards.

```
MenuItem redItem = new MenuItem("red");
```

now we invoke `setOnAction`.

```
redItem.setOnAction( e -> currentColor = color.red;);
```

We could do this for several menu items; what you would notice is the appearance of lots of duplicate code.

This is a situation in which we can make smart use of inheritance. Note that we need access to the state variables of `BallWorld` so it is a smart idea to make this class be an inner class.

```
class ColorMenuItem extends MenuItem
{
}
```

we need to tell this object two things: the color we wish to use for the current color and the name of that color. now, we put in a constructor.

```
class ColorMenuItem extends MenuItem
{
    public ColorMenuItem(Color c, String name)
    {
    }
}
```

we can call the parent constructor to label the menu like so

```
super(name);
```

also, we can call the `setOnAction` method to change the current color. here is the full result.

```
class ColorMenuItem extends MenuItem
{
    public ColorMenuItem(color c, string name)
    {
        super(name);
        setOnAction( e -> currentColor = c);
    }
}
```

what is nice is we do not have to add a bunch of unneeded variables to our `start` method; we can attach the menu items anonymously to the color menu as follows.

```
colorMenu.getItems().addall(new ColorMenuItem(color.red, "red"));
colorMenu.getItems().addall(new ColorMenuItem(color.green, "green"));
colorMenu.getItems().addall(new ColorMenuItem(color.blue, "blue"));
colorMenu.getItems().addall(new ColorMenuItem(color.web("0x7BAFD4", "carolina blue"));
colorMenu.getItems().addall(new ColorMenuItem(color.web("0x001A57", "dook bloo"));
```

one line of code adds a new color.

handling the size menu is similarly simple.

```
class SizeMenuItem extends MenuItem
{
    public SizeMenuItem(double s)
    {
        super("" + s);
        setOnAction( e -> currentSize = s);
    }
}
```

we then add items to the size menu.

```
sizeMenu.getItems().add(new SizeMenuItem(1));
sizeMenu.getItems().add(new SizeMenuItem(2));
sizeMenu.getItems().add(new SizeMenuItem(5));
sizeMenu.getItems().add(new SizeMenuItem(10));
sizeMenu.getItems().add(new SizeMenuItem(20));
sizeMenu.getItems().add(new SizeMenuItem(50));
sizeMenu.getItems().add(new SizeMenuItem(100));
sizeMenu.getItems().add(new SizeMenuItem(200));
```

The background menu works a little differently. When a menu items is selected, you must both update the background color and refresh the canvas, since the user expects the background color to change right away in response to the selection of the menu item.

Again, we shall extends the `MenuItem` class to accomplish this. Notice how we format the event handling code when it has more than one line. You will see a "sad Santa" when its block closes.

```
class BackgroundMenuItem extends MenuItem
{
    public BackgroundMenuItem(color c, string name)
```

```
    {
        super(name);
        setOnAction( e ->
        {
            backgroundColor = c;
            refresh();
        });
    }
}
```

Adding the menu items is simple.

```
backgroundMenu.getItems().add(new BackgroundMenuItem(Color.RED, "red"));
backgroundMenu.getItems().add(new BackgroundMenuItem(Color.GREEN, "green"));
backgroundMenu.getItems().add(new BackgroundMenuItem(Color.BLUE, "blue"));
backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x7bafd4"), "carolina blue"))
backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x001a57"), "dook bloo"));
MenuItem randomBackgroundItem = new MenuItem("random");
randomBackgroundItem.setOnAction(e -> backgroundColor = randcolor());
backgroundMenu.getItems().addall(randomBackgroundItem);
```

Now let us compile. We are greeted by this surly horror.

```
unix> javafxc BallWorld.java
BallWorld.java:116: error: cannot find symbol
                refresh(canvas);
                        ^
  symbol:   variable canvas
  location: class BallWorld.BackgroundMenuItem
1 error
unix>
```

It's time for a little plastic surgery: we need to "lift" the canvas object up to being a state variable. Here is what we have now.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
```

```
import javafx.geometry.Point2D;
import java.util.ArrayList;

public class BallWorld extends Application
{
    private Color backgroundColor;
    private Color currentColor;
    private int currentSize;
    private ArrayList<Ball> shapes;
    public BallWorld()
    {
        backgroundColor = Color.WHITE;
        currentColor = Color.BLACK;
        currentSize = 50;
        shapes = new ArrayList<>();
    }
    public void init()
    {
    }
    public static void main(String[] args)
    {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Ball World");
        BorderPane bp = new BorderPane();
        Canvas canvas = new Canvas(500,500);
        refresh(canvas);
        canvas.setOnMouseClicked(e ->
        {
            Ball b = new Ball(new Point2D(e.getX(),e.getY()),
                currentColor, currentSize);
            shapes.add(b);
            refresh(canvas);
        });
        bp.setTop(makeMenus());
        bp.setCenter(canvas);
        primaryStage.setScene(new Scene(bp));
        primaryStage.show();
    }
    private MenuBar makeMenus()
    {
        MenuBar mbar = new MenuBar();
```

```
    //create and insert menus
    Menu fileMenu = new Menu("File");
    Menu colorMenu = new Menu("Color");
            colorMenu.getItems().add(new ColorMenuItem(Color.RED, "red"));
            colorMenu.getItems().add(new ColorMenuItem(Color.GREEN, "green"));
            colorMenu.getItems().add(new ColorMenuItem(Color.BLUE, "blue"));
            colorMenu.getItems().add(new ColorMenuItem(Color.web("0x7BAFD4"), "carolina
            colorMenu.getItems().add(new ColorMenuItem(Color.web("0x001A57"), "dook bloc
    Menu backgroundMenu = new Menu("Background");
    Menu sizeMenu = new Menu("Size");
            sizeMenu.getItems().add(new SizeMenuItem(1));
            sizeMenu.getItems().add(new SizeMenuItem(2));
            sizeMenu.getItems().add(new SizeMenuItem(5));
            sizeMenu.getItems().add(new SizeMenuItem(10));
            sizeMenu.getItems().add(new SizeMenuItem(20));
            sizeMenu.getItems().add(new SizeMenuItem(50));
    backgroundMenu.getItems().add(new BackgroundMenuItem(Color.RED, "red"));
    backgroundMenu.getItems().add(new BackgroundMenuItem(Color.GREEN, "green"));
    backgroundMenu.getItems().add(new BackgroundMenuItem(Color.BLUE, "blue"));
    backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x7bafd4"), "carolin
    backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x001a57"), "dook bl
    mbar.getMenus().addAll(fileMenu, colorMenu, backgroundMenu, sizeMenu);
    return mbar;
}
    private void refresh(Canvas canvas)
    {
        GraphicsContext g = canvas.getGraphicsContext2D();
        g.setFill(backgroundColor);
        g.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
    for(Ball b: shapes)
    {
        b.draw(g);
    }
    }
class ColorMenuItem extends MenuItem
{
    public ColorMenuItem(Color c, String name)
    {
        super(name);
        setOnAction( e -> {currentColor = c;System.out.println(currentColor);});
    }
}

    class SizeMenuItem extends MenuItem
    {
        public SizeMenuItem(double s)
```

```
            {
                super("" + s);
                setOnAction( e -> currentSize = (int)s);
            }
        }
    class BackgroundMenuItem extends MenuItem
    {
        public BackgroundMenuItem(Color c, String name)
        {
            super(name);
            setOnAction( e ->
            {
                backgroundColor = c;
                refresh(canvas);
            });
        }
    }
}
```

Let us now begin the refactoring process. We need to do the following.

1. Make `canvas` a state variable and initialize it in the constructor.
2. Get rid of the local `canvas` variable in `start` and its initialization.
3. Since `canvas` is a state variable and is now visible inside of the class, let us refactor the `refresh(Canvas canvas)` method to `refresh()`. Find all calls to `refresh` and make them read `refresh()` instead of `refresh(canvas)`.

7 RRRRIIIINNNNGGGG!!! Marketing is Calling! Urgent!

"Yeah, ladies and gentlemen, BallWorld is cool but now our users want circle and triangles! Our website verily brims with requests. Put your best people on it."

Uh oh. a little more refactoring is in order. Happily, we have engineered our program so there is a separate class for making the balls appear. Let's review the interface we have for the balls. Our balls know their colors, sizes, and shapes. They know how to do one thing: draw themselves with the graphics pen.

Here is where a little interface magic comes in handy. Let us create an interface called `Shape`, which specifies a method for drawing.

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

```
public interface Shape
{
    public void draw(GraphicsContext g);
}
```

Next, we should make our `Ball` class implement this interface. Edit the header of your `Ball` class as follows.

```
public class Ball implements Shape
```

Now compile everything.

Now let us refactor in the main `BallWorld` class. Change your array list of balls as follows.

```
private ArrayList<Shape> shapes;
```

next visit the `refresh()` method and revise its loop as follows.

```
for(shape s: shapes)
{
    s.draw(g);
}
```

Recompile. It all now works. Our refactoring is successful.

What is new? Our array list of shapes can contain any shape we devise that implements the `Shape` interface. we have paved the way for meeting Marketing's request.

8 Adding New Shapes

How do we specify a shape? A smart way is to use an `enum`, or enumerated type. This is just a little "bag" that holds static constants. this can be nested in the `BallWorld` class. To add a new shape, we will add a new member to this nested `enum`.

```
private enum ShapeChoices{BALL, SQUARE};
```

Then add this to your state variables.

```
private ShapeChoices currentShape;
```

Add this line to the constructor.

```
currentShape = ShapeChoices.BALL;
```

Finally, add this conditional logic in the canvas's mouse clicked event handler.

```
canvas.setOnMouseClicked( e ->{
    if(currentShape.equals(ShapeChoices.BALL)
    {
        shapes.add(new Ball(new Point2D(e.getX(), e.getY()),
            currentSize, currentColor));
    }
    refresh();
});
```

Now we are ready to add a square. Begin by inserting the requisite method

```
import javafx.scene.canvas.GraphicsContext;
public class Square implements Shape
{
    public void draw(GraphicsContext g)
    {
    }
}
```

Now let us create the `Square` class. Our square will draw like a circle and be the size of a circle's bounding box.

```
import javafx.scene.paint.Color;
import javafx.geometry.Point2D;
import javafx.scene.canvas.GraphicsContext;
public class Square implements Shape
{
    final private double x;
    final private double y;
    final private double red;
    final private double green;
    final private double blue;
    final private double size;
        public Ball(Point2D center, Color color, double size)
        {
        this.red = color.getRed();
        this.green = color.getGreen();
        this.blue = color.getBlue();
        this.x = center.getX();
        this.y = center.getY();
            this.size = size;
```

```
        }
    public Color getColor()
    {
        return Color.color(red, green, blue);
    }
    public Point2D getCenter()
    {
        return new Point2D(x,y);
    }
    public void draw(GraphicsContext g)
    {
        g.setFill(getColor());
        g.fillRect(getCenter().getX() - size,
            getCenter().getY() - size, 2*size, 2*size);
    }
}
```

Now add this logic to the mouse clicked event handler.

```
canvas.setOnMouseClicked(e ->
{
    if(currentShape == ShapeChoices.BALL)
    {
        Shape s = new Ball(new Point2D(e.getX(),e.getY()),
            currentColor, currentSize);
        shapes.add(s);
    }
    if(currentShape.equals(ShapeChoices.SQUARE))
    {
        shapes.add(new Square(new Point2D(e.getX(),
           e.getY()), currentColor, currentSize));
    }
    refresh();
});
```

The next step is to add a menu to choose a shape. Append this code to `makeMenus()`.

```
Menu shapeMenu = new Menu("Shape");
mbar.getMenus().addAll(shapeMenu);
shapeMenu.getItems().addAll(new ShapeMenuItem(ShapeChoices.BALL));
shapeMenu.getItems().addAll(new ShapeMenuItem(ShapeChoices.SQUARE));
```

Now create a you guessed it ... a `ShapeMenuItem`.

```
class ShapeMenuItem extends MenuItem
{
```

```
    public ShapeMenuItem(ShapeChoices s)
    {
        super(s.toString());
        setOnAction( e -> currentShape = s);
    }
}
```

Now compile and run. Your program will now have a menu that offers a choice of shape.

Programming Exercises In these exercises, you will add triangles to your `BallWorld` program. Here is a to-do list to make it happen. Compile and run at each step so you can see what is happening.

1. Modify the nested enum to have a constant named `TRIANGLE`.
2. Add a new menu item for triangles.
3. Create a class named `Triangle.java`. Have it draw an upward-pointing equilateral triangle. Use the distance from the center to the vertices as its size. You will need to use a little trig for this.
4. Modify the mouse click handling code to add a branch for handling the menu choice of a triangle.
5. Try repeating the process with a new shape such as a hexagon or a pentagon.
6. **Mini-Research Project** Can you find a way to produce a dialog box that allows the user to enter a custom size?
7. **Research Project** Can you find a way to produce a color picker box and give custom colors to both the background and the current color?

9 All of the Source

`BallWorld.java`

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.scene.canvas.Canvas;
```

```
import javafx.scene.canvas.GraphicsContext;
import javafx.geometry.Point2D;
import java.util.ArrayList;

public class BallWorld extends Application
{
    private Color backgroundColor;
    private Color currentColor;
    private int currentSize;
    private ArrayList<Shape> shapes;
    private Canvas canvas;
    private ShapeChoices currentShape;

    public BallWorld()
    {
        backgroundColor = Color.WHITE;
        currentColor = Color.BLACK;
        currentSize = 50;
        shapes = new ArrayList<>();
        canvas = new Canvas(500,500);
        currentShape = ShapeChoices.BALL;
    }
    public void init()
    {
    }
    public static void main(String[] args)
    {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Ball World");
        BorderPane bp = new BorderPane();
        refresh();
        canvas.setOnMouseClicked(e ->
        {
            if(currentShape == ShapeChoices.BALL)
            {
                Shape s = new Ball(new Point2D(e.getX(),e.getY()),
                    currentColor, currentSize);
                shapes.add(s);
            }
            if(currentShape.equals(ShapeChoices.SQUARE))
            {
```

```
                shapes.add(new Square(new Point2D(e.getX(),
                   e.getY()), currentColor, currentSize));
            }
            refresh();
        });
        bp.setTop(makeMenus());
        bp.setCenter(canvas);
        primaryStage.setScene(new Scene(bp));
        primaryStage.show();
    }
    private MenuBar makeMenus()
    {
        MenuBar mbar = new MenuBar();
        //create and insert menus
        Menu fileMenu = new Menu("File");
        Menu colorMenu = new Menu("Color");
                colorMenu.getItems().add(new ColorMenuItem(Color.RED, "red"));
                colorMenu.getItems().add(new ColorMenuItem(Color.GREEN, "green"));
                colorMenu.getItems().add(new ColorMenuItem(Color.BLUE, "blue"));
                colorMenu.getItems().add(new ColorMenuItem(Color.web("0x7BAFD4"), "carolina
                colorMenu.getItems().add(new ColorMenuItem(Color.web("0x001A57"), "dook bloc
        Menu backgroundMenu = new Menu("Background");
        Menu sizeMenu = new Menu("Size");
                sizeMenu.getItems().add(new SizeMenuItem(1));
                sizeMenu.getItems().add(new SizeMenuItem(2));
                sizeMenu.getItems().add(new SizeMenuItem(5));
                sizeMenu.getItems().add(new SizeMenuItem(10));
                sizeMenu.getItems().add(new SizeMenuItem(20));
                sizeMenu.getItems().add(new SizeMenuItem(50));
        backgroundMenu.getItems().add(new BackgroundMenuItem(Color.RED, "red"));
        backgroundMenu.getItems().add(new BackgroundMenuItem(Color.GREEN, "green"));
        backgroundMenu.getItems().add(new BackgroundMenuItem(Color.BLUE, "blue"));
        backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x7bafd4"), "carolir
        backgroundMenu.getItems().add(new BackgroundMenuItem(Color.web("0x001a57"), "dook bl
        mbar.getMenus().addAll(fileMenu, colorMenu, backgroundMenu, sizeMenu);
        Menu shapeMenu = new Menu("Shape");
        mbar.getMenus().addAll(shapeMenu);
        shapeMenu.getItems().add(new
            ShapeMenuItem(ShapeChoices.BALL));
        shapeMenu.getItems().add(new
            ShapeMenuItem(ShapeChoices.SQUARE));
        return mbar;
    }
        private void refresh()
        {
            GraphicsContext g = canvas.getGraphicsContext2D();
```

```
            g.setFill(backgroundColor);
            g.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
        for(Shape s: shapes)
        {
            s.draw(g);
        }
        }
    class ColorMenuItem extends MenuItem
    {
        public ColorMenuItem(Color c, String name)
        {
            super(name);
            setOnAction( e -> {currentColor = c;System.out.println(currentColor);});
        }
    }

        class SizeMenuItem extends MenuItem
        {
            public SizeMenuItem(double s)
            {
                super("" + s);
                setOnAction( e -> currentSize = (int)s);
            }
        }
    class BackgroundMenuItem extends MenuItem
    {
        public BackgroundMenuItem(Color c, String name)
        {
            super(name);
            setOnAction( e ->
            {
                backgroundColor = c;
                refresh();
            });
        }
    }
    private enum ShapeChoices{BALL, SQUARE};
    class ShapeMenuItem extends MenuItem
    {
        public ShapeMenuItem(ShapeChoices s)
        {
            super(s.toString());
            setOnAction( e -> currentShape = s);
        }
    }
}
```

Ball.java

```
import javafx.scene.paint.Color;
import javafx.geometry.Point2D;
import javafx.scene.canvas.GraphicsContext;
public class Ball implements Shape
{
    final private double x;
    final private double y;
    final private double red;
    final private double green;
    final private double blue;
    final private double radius;
        public Ball(Point2D center, Color color, double radius)
        {
        this.red = color.getRed();
        this.green = color.getGreen();
        this.blue = color.getBlue();
        this.x = center.getX();
        this.y = center.getY();
            this.radius = radius;
        }
    public Color getColor()
    {
        return Color.color(red, green, blue);
    }
    public Point2D getCenter()
    {
        return new Point2D(x,y);
    }
    public void draw(GraphicsContext g)
    {
        g.setFill(getColor());
        g.fillOval(getCenter().getX() - radius,
            getCenter().getY() - radius, 2*radius, 2*radius);
    }
}
```

Square.java

```
import javafx.scene.paint.Color;
import javafx.geometry.Point2D;
import javafx.scene.canvas.GraphicsContext;
public class Square implements Shape
{
```

```
    final private double x;
    final private double y;
    final private double red;
    final private double green;
    final private double blue;
    final private double size;
        public Square(Point2D center, Color color, double size)
        {
        this.red = color.getRed();
        this.green = color.getGreen();
        this.blue = color.getBlue();
        this.x = center.getX();
        this.y = center.getY();
            this.size = size;
        }
    public Color getColor()
    {
        return Color.color(red, green, blue);
    }
    public Point2D getCenter()
    {
        return new Point2D(x,y);
    }
    public void draw(GraphicsContext g)
    {
        g.setFill(getColor());
        g.fillRect(getCenter().getX() - size,
            getCenter().getY() - size, 2*size, 2*size);
    }
}
```

Shape.java

```
import javafx.scene.canvas.GraphicsContext;
public interface Shape
{
    public void draw(GraphicsContext g);
}
```