PART ONE – CREATING THE BOARD AND MAIN SPRITES

SETTING UP THE WINDOW

Create a new basic JavaFX application. Use a **Group** for the main layout. When setting the scene, ensure the basic window is **800 by 600**.

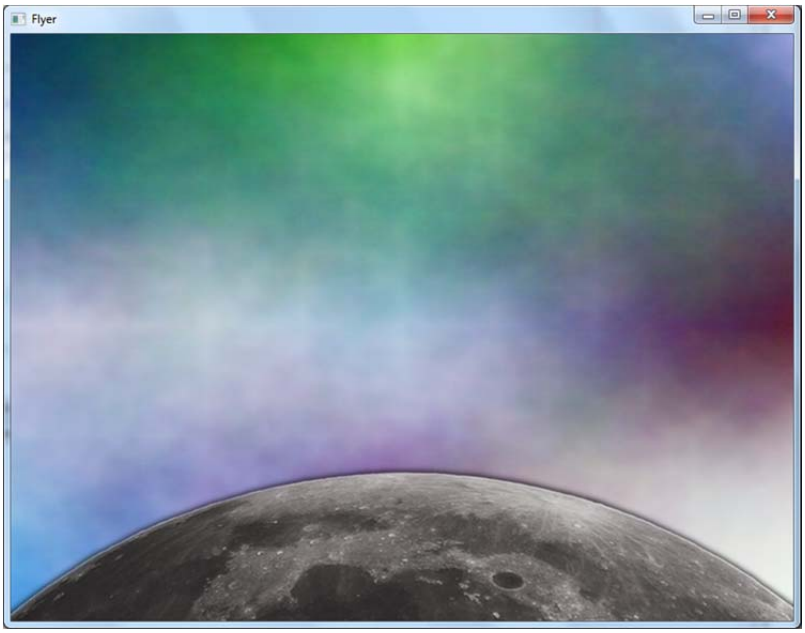
```
@Override
public void start(Stage primaryStage){
    Group g = new Group();
    Scene scene = new Scene(g, 800,600);

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Download and extract the files called **Sprites and Graphics**. Copy the files into the src directory for your application.

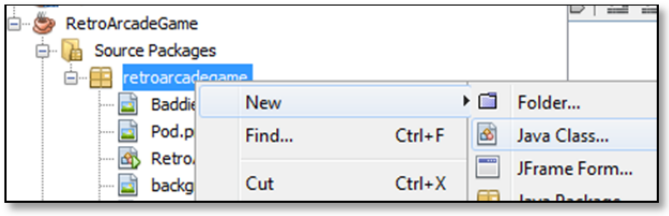
Create an Image that uses the background. Create an ImageView that uses this image and add it to the group.

```
Image imgBack = new Image(getClass().getResourceAsStream("background.png"));
ImageView ivBack = new ImageView(imgBack);
g.getChildren().add(ivBack);
```

Your application should look like this when run:



ADDING A PLAYER SPRITE



Right click the package for the source files and add a new Java class. Name this class **Rocket**.

Create the following global variables in your Rocket:

- Integer: x
- Integer: y
- Integer: dx
- Integer dy
- ImageView: ivRocket

A constructor method is one that is run whenever an object is instantiated from a class. It has the same name as the class. In the constructor for your rocket, initialise:

- x and y so that the rocket will appear at the bottom of the screen somewhere near the moon
- dx and dy to 10
- The ImageView ivRocket to the image of the rocket (ship.png)
- Set the x and y edges of the ImageView using setLayoutX and setLayoutY.

Next, we need a publicly accessible getImage method that returns the ImageView of the rocket to the calling member. Create a public method called getImage() that returns ivRocket.

This is the complete Rocket class so far:

```
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class Rocket {
    int x, y, dx, dy;
    ImageView ivRocket;

    Rocket(){
        x=400;
        y=400;
        dx=10;
        dy=10;
        Image imgRocket = new
Image(getClass().getResourceAsStream("rocket.png"));
        ivRocket = new ImageView(imgRocket);
        ivRocket.setLayoutX(x);
        ivRocket.setLayoutY(y);
    }

    public ImageView getImageView(){
        return ivRocket;
    }
}
```

Now we need to display the rocket in the scene. Back in the main class for your project, define the rocket as a global object.

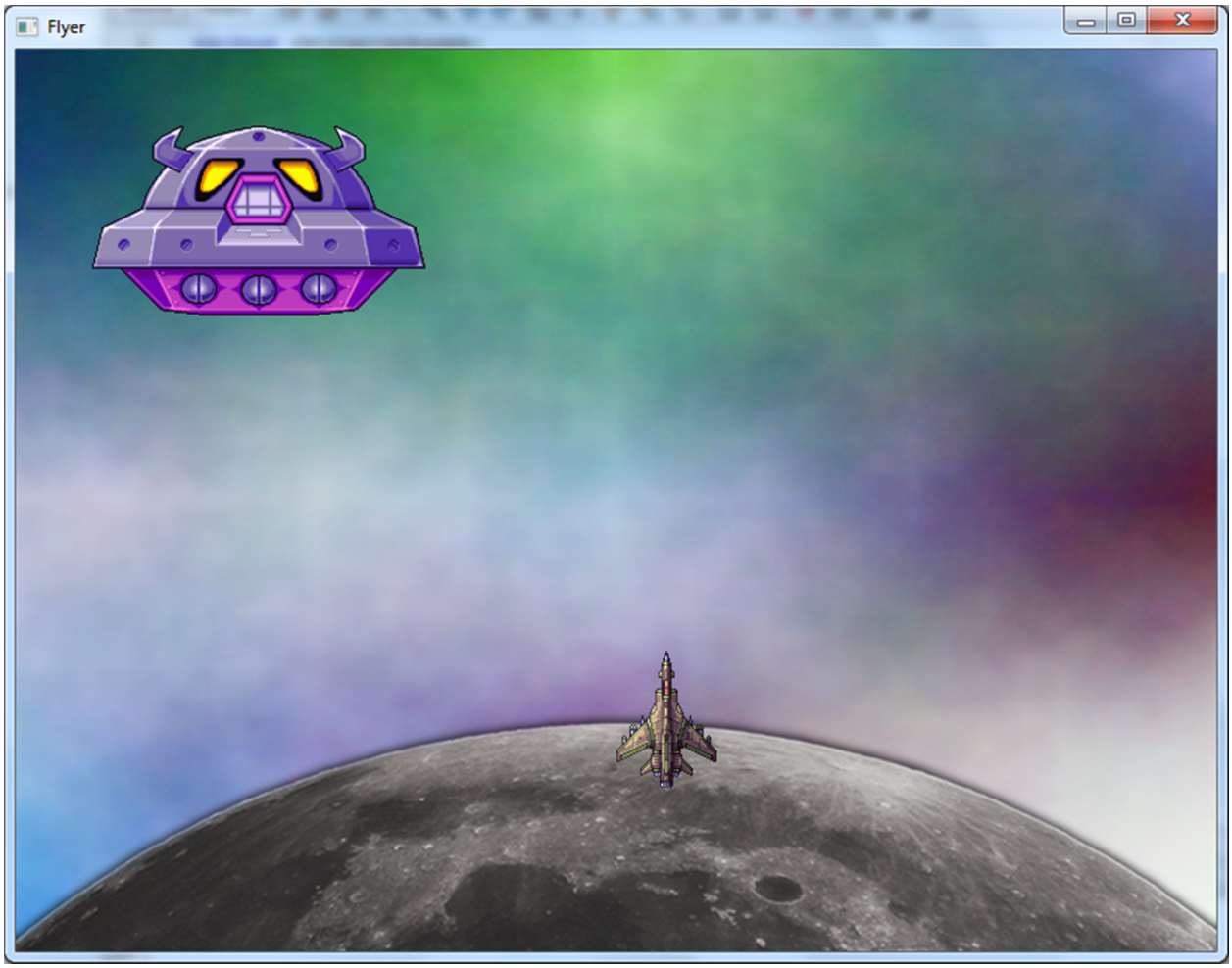
Declaring a global Rocket.

```
20  public class RetroArcadeGame extends Application{
    Rocket rocket;
22
23      @Override
        public void start(Stage primaryStage){
```

Next, create an instance of the rocket in your start method. Finally, use the getImage() method of the rocket to add the ImageView to the group.

```
rocket = new Rocket();
g.getChildren().add(rocket.getImageView());
```

Repeat these steps for the baddie. If you run your application, it should look something like this:



This is my final main class **without the baddie added in**:

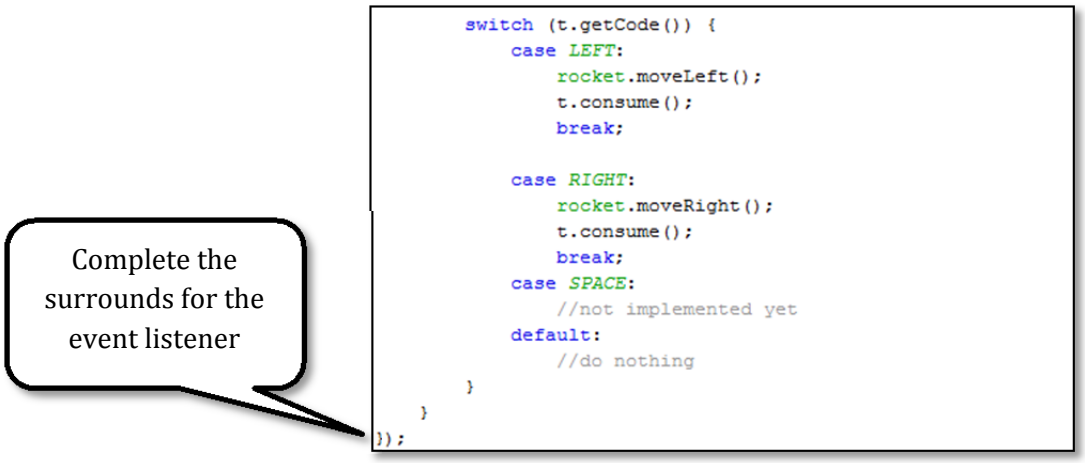
```
package retroarcadegame;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

/**
 *
 * @author Yatish
 */
public class RetroArcadeGame extends Application{
Rocket rocket;

    @Override
    public void start(Stage primaryStage){
        Group g = new Group();
        Scene scene = new Scene(g, 800,600);

        rocket = new Rocket();

        Image imgBack = new
Image(getClass().getResourceAsStream("background.png"));
        ImageView ivBack = new ImageView(imgBack);

        g.getChildren().add(ivBack);
        g.getChildren().add(rocket.getImageView());

        primaryStage.setTitle("Flyer");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }

}
```

ADDING ACTION LISTENERS TO THE ROCKET SPRITE

Add an action listener to the scene (in your main class) which listens for key events. This should implement an anonymous inner class with a handle method that overrides the interface.

```
scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
@Override
public void handle(KeyEvent t) {
```

Use selection to determine whether an action should be carried out in response to an event. I have used a switch. An if-statement would suffice.



This is my complete code for the action listener:

```
        scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
        @Override
        public void handle(KeyEvent t) {

            switch (t.getCode()) {
                case LEFT:
                    rocket.moveLeft();
                    t.consume();
                    break;

                case RIGHT:
                    rocket.moveRight();
                    t.consume();
                    break;
                case SPACE:
                    //not implemented yet
                default:
                    //do nothing
            }
        }
    });
```

PROGRAMMING THE ROCKET TO RESPOND

Return to your Rocket class. Create methods in the rocket for moveLeft and moveRight as called for by the action listener.

dx is going to be the variable that controls the speed of the rocket. The large dx, the large the number of pixels it will move each time the move method is called.

Subtract dx from x and assign the result to x.

```
x = x-dx;
```

We want the rocket to move left if it has not yet reached the edge of the screen. You can move the rocket by using the setLayoutX method of the ImageView that you used to position it initially.

```
void moveLeft() {
    if (x>50){
    x = x-dx;
    ivRocket.setLayoutX(x);
    }
}
```

The moveRight method works in a similar manner to this, but must test for the right edge of the screen.

When you run your game, you should be able to control the rocket sprite using the arrow keys on the keyboard.