SOFT2201/COMP9201 Week 4 Tutorial

JavaFX and Java GUI

We will be using JavaFX through out the semester as a GUI toolkit. JavaFX allows us to create rich multi-media applications, allowing programmers to create user-interfaces that integrate with the operating system's desktop environment.

Question 1: Setting up javaFX project

We want to construct a gradle project where we will be able to integrate javafx for the next set of exercises. Initialise a gradle project and modify the `build.gradle` file to include the following plugin.

```
plugins {
  id 'application'
  id 'org.openjfx.javafxplugin' version '0.0.8'
}
```

And include the javafx controls module within the `build.gradle` file.

```
javafx {
    version = "12.0.2"
    modules = [ 'javafx.controls' ]
}
```

Within your project, you should import the following classes to use within your JavaFX application. You will need to utilise them through out the tutorial.

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
```

Question 2: Creating a window

To get started, we will create a simple `Hello JavaFX` window application. We will be using this as the basis for our javafx applications. The first part of our application will contain a `start` method. The `start` method will be an overidden method from the `Application` class.

Our first javafx application will manage the initialisation by incorporating the `main` within its class.

```java
import javafx.application.Application;

public class HelloFX extends Application {

    @Override
    public void start(Stage stage) {
        //Application initialisation code here
    }

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

}
```

The `launch` method will create a standalone application. An `Application` instance will be constructed and the `start` method will be invoked.

Let's create a window, we will want to import `javafx.scene.Scene`, `Group` and `javafx.stage.Stage`.

Please refer to the Java FX 12 API documentation through out the semester.

We will need to set up a `Scene`, `Group` and `Stage` object. The stage object will contain our scene and our scene will contain all renderable objects.

```java
Group root = new Group();
Scene scene = new Scene(root);
stage.setScene(scene);
```

Afterwards we will set a new label for the topbar of our application.

```java
stage.setTitle("Hello JavaFX");
```

We now want a canvas which will allow us to draw images onto it. We will create a canvas with the dimensions of 600x400.

```java
Canvas canvas = new Canvas(600, 400);
```

However, we will need to attach the canvas to our root node. We can do this by retrieve the collection through the `.getChildren` method and simply add the canvas to it through the `.add` method.

We will then need to call `stage.show()` which will then start rendering our scene.

Once you have created this, launch your application through `gradle run` and view your result.

Question 3: Drawing elements on screen

We want to start drawing elements on the screen. JavaFX provides a simple interface for drawing 2D and 3D primitives. You will need to include the following types into your project. You may download a template `Template.zip` from Canvas that will have all the necessary classes and structure for you.

```
import javafx.scene.shape.Circle;
import javafx.scene.canvas.GraphicsCintext;
import javafx.scene.paint.Color;
```

We will need to use a `Group` type node to hold all the objects that will render the scene.

Our `Circle` object can be positioned and sized on instantiation using its constructor. We will need to extract the graphics context from the `Canvas` object that will provide us an interface to draw with.

Please refer to the the `GraphicsContext` documentation for the appropriate draw, fill and stroke methods.

```
Circle circle = new Circle(50, 50, 10);]
GraphicsContext gc = canvas.getGraphicsContext2D();
gc.setFill(Color.BLUE); //Any fill function will use this colour
gc.fillOval(circle.getCenterX() - circle.getRadius(),
    circle.getCenterY() - circle.getRadius()
    circle.getRadius() * 2
    circle.getRadius() * 2);
```

After you have simply rendered a circle on the screen, draw a `Ellipse` and `Rectangle` and `Line` object onto the screen. Refer to the documentation by checking the sub classes.

- Do you need a shape object to draw an element?
- What is the advantage of using a shape type within our application?
- What could we do to make the drawing of shapes easier?

Question 4: Bouncy ball

Create a program that will bounce a ball around the screen. Start with the ball travelling from the bottom of the screen to the top and back again.

```
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.util.Duration;
```

When we construct a `Timeline` object we will need to

```
Timeline animationLoop = new Timeline();
animationLoop.setCycleCount(Timeline.INDEFINITE);
```

We will need to utilise both `KeyFrame` as a single and continually updated rendered frame onto the canvas.

Define a duration, in this instance we will aim to update the frame every 60th of a second.

```
//Approximation
private static final double KEY_FRAME_DURATION = 0.017;
```

Afterwards, construct a key frame object, it will require a type which implements `EventHandler` interface (You can use a lambda function or method reference in this scenario).

```
KeyFrame frame = new KeyFrame(Duration.seconds(KEY_FRAME_DURATION),
    /* Your EventHandler type */);
```

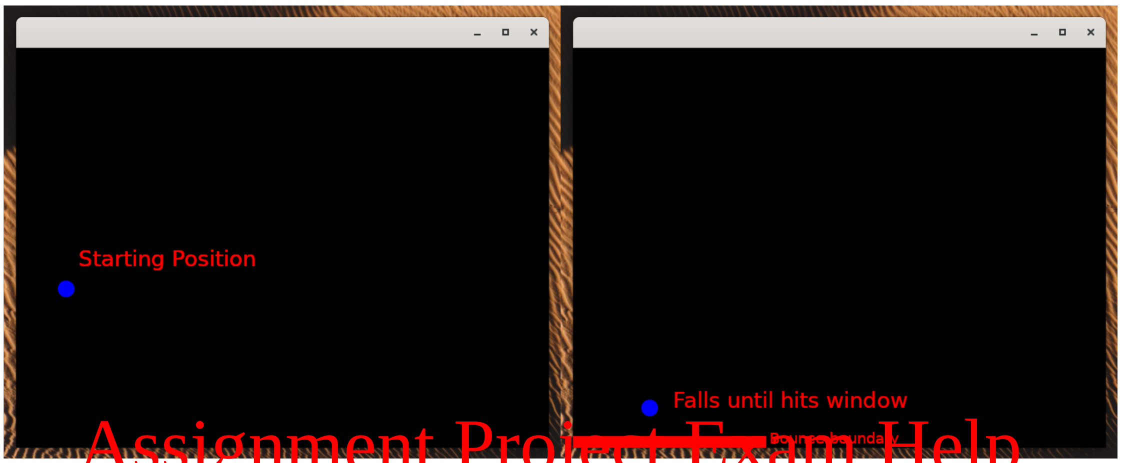
Afterwards, this is set to the `Timeline` object and rendered on each call.

```
KeyFrame frame = new KeyFrame(Duration.seconds(KEY_FRAME_DURATION),
    (actionEvent) -> {
        //Your event handler logic
    });
```

Question 5: Ball from end to end

You have been able to create a window, draw a circle and simply animate objects on screen. Now create a simple simulation where a ball starts from the left side and slowly bounces to the right, losing half its bounce height on each bounce until it rolls.

You will need to ensure it has a start horizontal speed so it gradually moves towards other end.



You will be required to create a gravity constant and a vector for the ball. The vector will be manipulated on each frame and will the Y component will invert when it hits the bottom of the window.

Question 6: Drawing images

Our ball simulation can use a background, consider the order of rendering when creating a background. The last rendered object will be rendered on top of everything else. Be conscious of rendering order and the number of elements to be draw in your scene.

To draw an image onto the screen, you can refer to the `Image` type within the javafx documentation.

We have provided you with a few assets above to use within your application.