

>Uj U: L`&!`5`>Uj U8 Yj Y`cdYffg`; i]XY



Hands on Lab Manual JavaFX

Stephen Chin
@steveonjava
<http://steveonjava.com/>

Introduction

JavaFX is the next step in the evolution of Java as a rich client platform. It is designed to provide a lightweight, hardware-accelerated Java UI platform for enterprise client applications. The latest release, JavaFX 2, represents a significant shift from previous releases. Developers can now create JavaFX applications completely in the Java programming language with a fresh new set of API libraries. There are a number of new features being introduced in JavaFX 2 such as Java Swing integration, web content integration, a hardware accelerated graphics pipeline and new UI controls library.

This HOL will provide a crash course to help you get up and running with JavaFX 2 quickly. No prior experience in JavaFX is required. Although a basic knowledge of the Java language will surely help. Join us and start creating stunning UI.

Prerequisites

This hands-on lab assumes you have some basic knowledge of or programming experience in, the following technologies:

- Java language programming

System Requirements

JavaFX 2 System Requirements

Use the following information to ensure that your operating system, browser, and JDK version meet the requirements for running the JavaFX technology. Also included are the hardware requirements for accelerated rendering of graphics.

Operating Systems and Browsers

You must run an operating system and browser that are compatible with JavaFX, as shown in the following table.

Operating System (32-Bit and 64-Bit)	Browsers (32-Bit and 64-Bit)
Windows XP Home and Professional with Service Pack 3	Internet Explorer 7 and 8 Firefox 3.5, 3.6, and 4 Chrome
Windows Vista Home Basic, Home Premium, Business, and Ultimate with Service Pack 2	Internet Explorer 7, 8, and 9 Firefox 3.5, 3.6, and 4 Chrome
Windows 7 Home Premium, Professional, and Ultimate	Internet Explorer 8 and 9 Firefox 3.5, 3.6, and 4 Chrome

Graphics Support

You will notice an accelerated rendering of graphics in your JavaFX applications if your system has support for the new Prism hardware pipeline. The following table lists the graphics cards that have been tested with Prism. If your system does not support Prism, then JavaFX uses the Java2D software pipeline under Prism.

Graphics Card	Supported Graphics Processing Units (GPUs)
NVIDIA	Mobile GPUs: GeForce 8M and 100M series or higher, NVS 2100M series or higher, and Mobility Quadro FX 300M series or higher Desktop GPUs: GeForce 8 and 100 series or higher Workstation GPUs: Quadro FX 300 series or higher
ATI	Mobile GPUs: Mobility Radeon HD 3000, 4000, and 5000 series Desktop GPUs: Radeon HD 2400, 3000, 4000, 5000, and 6000 series
Intel	Mobile GPUs: GMA 4500MHD and GMA HD Desktop GPUs: GMA 4500 and GMA HD

Software Needed For This Lab

The following software should be installed to participate in this lab exercises.

- [JDK 7 or higher](#)
- [NetBeans IDE 7.4](#)
- [JavaFX 2.2](#)

Lab Exercises

- Exercise 1: Getting started with JavaFX 2, running and understanding your first application. (10 mins)
- Exercise 2: A bit of everything (20 mins)
- Exercise 3: Understanding Layouts (10 mins)
- Exercise 4: Embedded Browser (10 mins)
- Exercise 5: Media (10 mins)

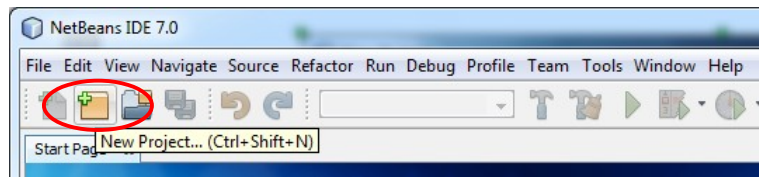
Additional Resources

<http://javafx.com>

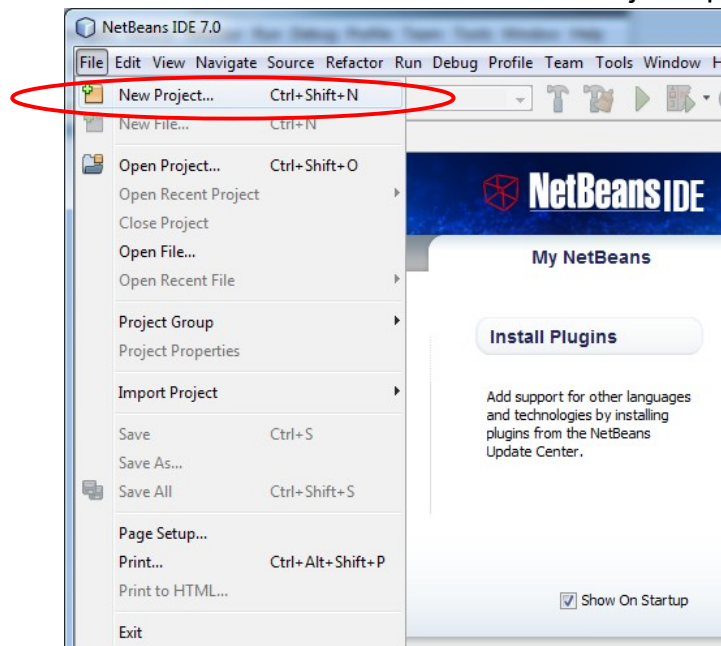
Exercise 1: Getting started with JavaFX 2, running and understanding your first applications. (10 mins)

This exercise will take you through some of the NetBeans' features to support JavaFX development. You will be able to create your first application without even typing any code.

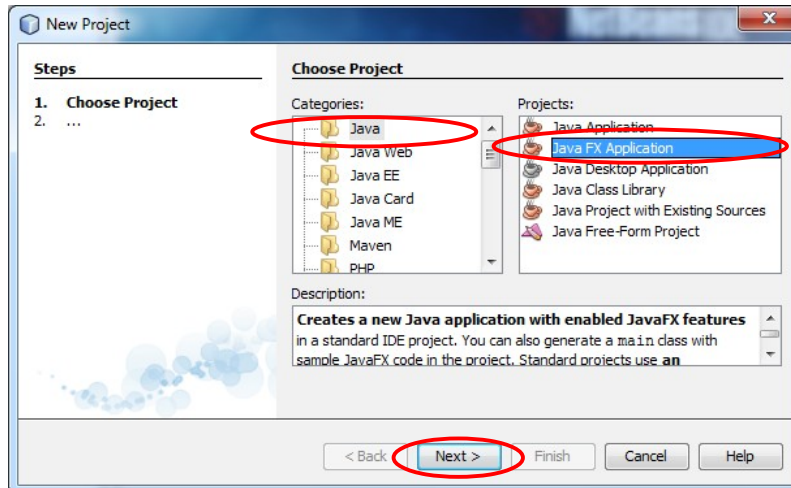
- Create a new project:
 - Option 1: Use the New Project... icon



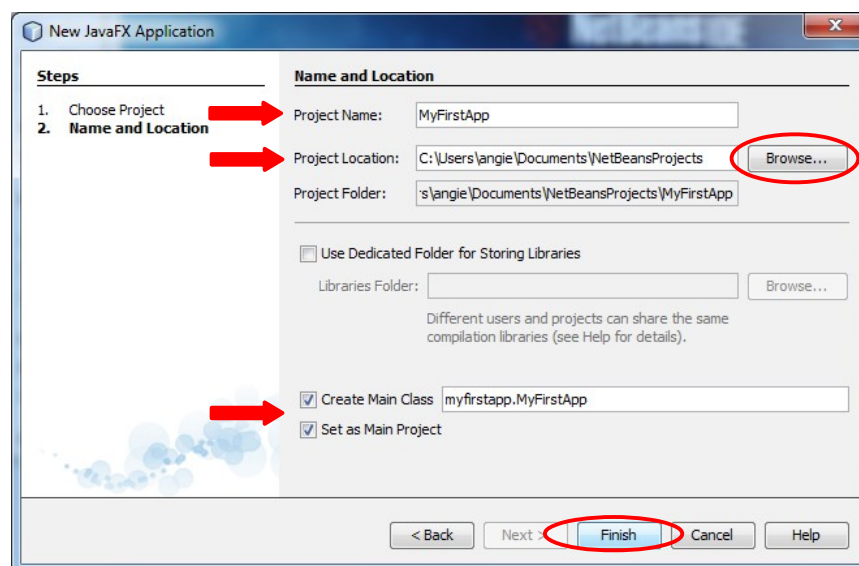
- Option 2: Go to the File menu and select New Project option from there.



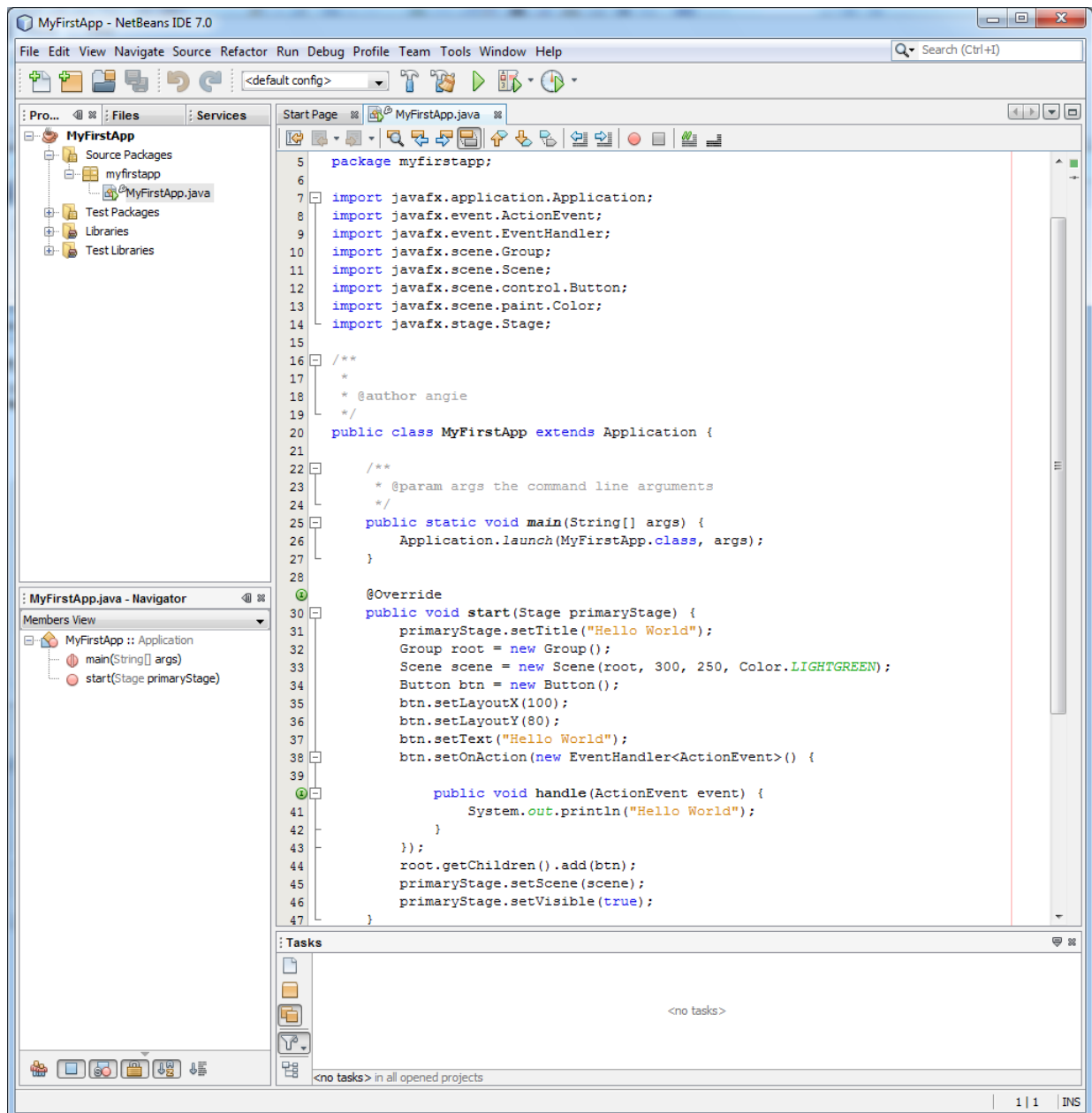
- The New Project window will appear. Select:
 - Categories: Java
 - Projects: JavaFX Application
 - Click on Next >



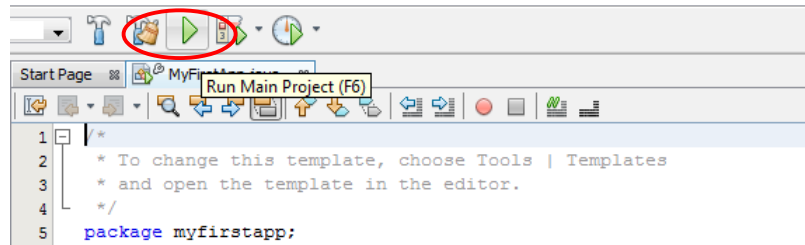
- The New JavaFX Application window will be displayed.
 - Use any name for your Project Name
 - You can also select any destination for your new project location
 - Make sure the Create Main Class option is selected
 - For your convenience, select also the Set as Main Project option
 - Click Finish



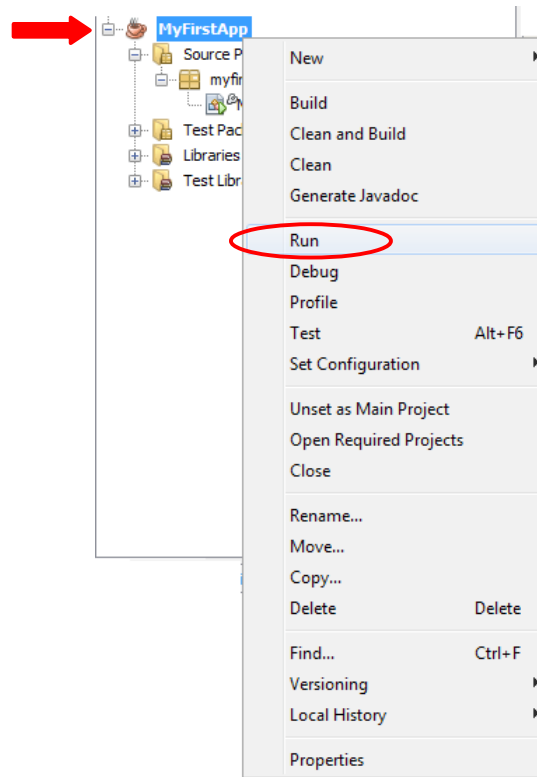
- Your new project will be created and the main class will be opened. Don't worry too much about the code, as we'll study it in the next exercise. Briefly you can notice a scene object, the container for your UI components, it has a size of 300x250 and filling color of light green. You can also see a Button, with a "Hello World" text on it. Finally there is an event handler for the mouse action events, where we simply do a print on the output window.



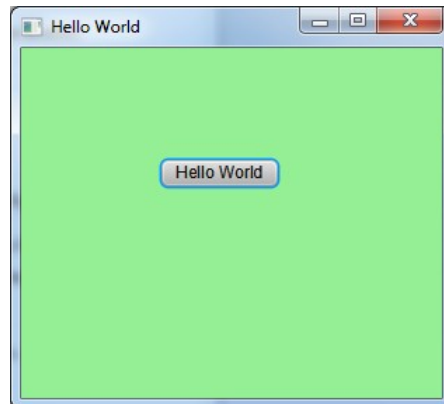
- Lets run the project:
 - Option 1: Use the Run Main Project icon



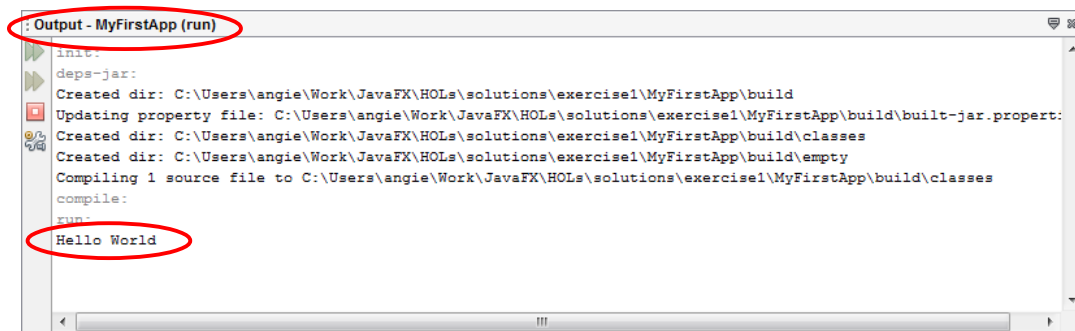
- Option 2: Right click on the project name and from the pop up menu select Run.



- Your application will look like this

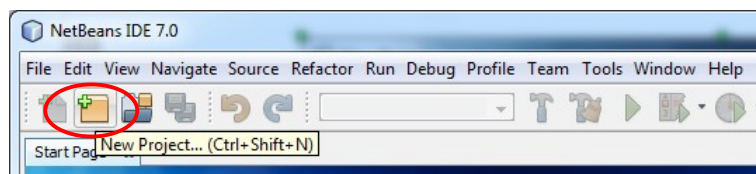


- Click on the “Hello World” button and notice the “Hello World” string on the output area.

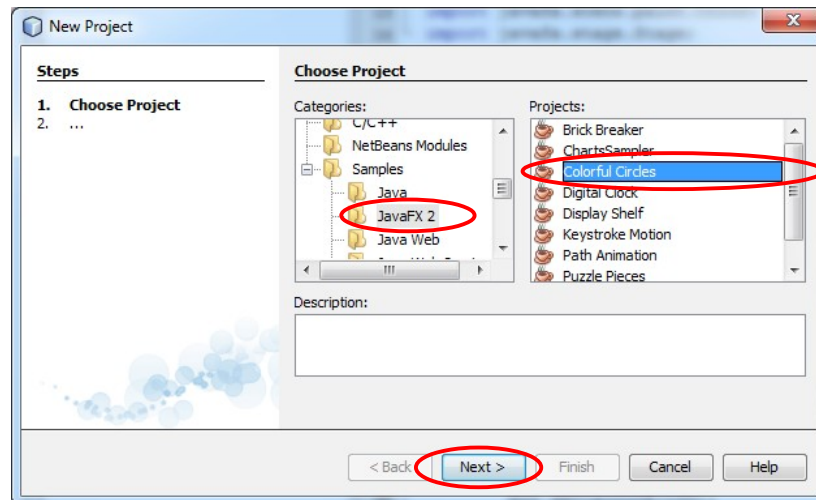


JavaFX 2 plugin for NetBeans also provides you with a bunch of pre-defined samples. They could be very useful as starting point for learning JavaFX or simply to start your projects, rather than creating an empty one. Lets see how to use them.

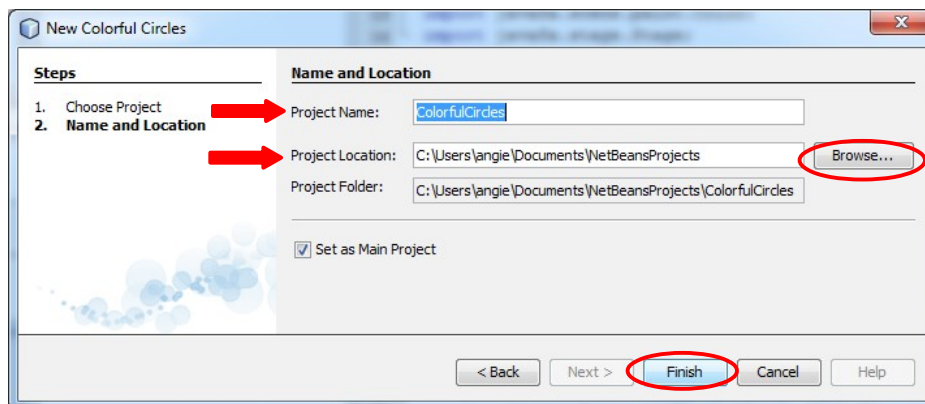
- Create a new Project



- The New Project window gets displayed:
 - In the Categories section: expand Samples and select JavaFX 2
 - For project: select ColorfulCircles
 - Click on Next >



- The New Colorful Circles window gets displayed:
 - Leave the default project name
 - Choose any destination directory for the project
 - Click on Finish



- One more time, we are not going to spend a lot of time at the code, we just want to highlight a few things before we run the project.
 - We have scene, in this case a bit bigger 800x600 with a black background
 - We create 3 different layers, and in each layer we create a group of circles.

- We add some blur effect to each layer
- We create a rectangle, and instead of filling it with a plain color, we use a linear gradient to have a nice effect.
- We put together the 3 layers, the previous rectangle and a newly created rectangle into a group, with a overlay blending mode.
- We create an animation to move all the created circles around the screen.
- We run the animation
- And finally we run the whole application.

```

1. package com.javafx.experiments.colorfulcircles;
2.
3. import javafx.animation.Animation;
4. import javafx.animation.KeyFrame;
5. import javafx.animation.KeyValue;
6. import javafx.animation.Timeline;
7. import javafx.util.Duration;
8. import javafx.scene.Group;
9. import javafx.scene.Node;
10. import javafx.scene.Scene;
11. import javafx.scene.effect.BlendMode;
12. import javafx.scene.effect.BoxBlur;
13. import javafx.scene.paint.Color;
14. import javafx.scene.paint.CycleMethod;
15. import javafx.scene.paint.LinearGradient;
16. import javafx.scene.paint.Stop;
17. import javafx.scene.shape.Circle;
18. import javafx.scene.shape.Rectangle;
19. import javafx.scene.shape.StrokeType;
20. import javafx.stage.Stage;
21.
22. import java.util.ArrayList;
23. import java.util.List;
24. import javafx.application.Application;
25. import static java.lang.Math.random;
26.
27. public class ColorfulCirclesDocs extends Application {
28.
29.     @Override public void start(Stage stage) {
30.         Scene scene = new Scene(new Group(), 800, 600, Color.BLACK);
31.         stage.setScene(scene);
32.         Group layer1 = new Group();
33.         for(int i=0; i<15;i++) {
34.             Circle circle = new Circle(200, Color.web("white", 0.05f));
35.             circle.setStrokeType(StrokeType.OUTSIDE);
36.             circle.setStroke(Color.web("white", 0.2f));
37.             circle.setStrokeWidth(4f);
38.             layer1.getChildren().add(circle);
39.         }
40.         Group layer2 = new Group();
41.         for(int i=0; i<20;i++) {
42.             Circle circle = new Circle(70, Color.web("white", 0.05f));
43.             circle.setStrokeType(StrokeType.OUTSIDE);
44.             circle.setStroke(Color.web("white", 0.1f));
45.             circle.setStrokeWidth(2f);
46.             layer2.getChildren().add(circle);
47.         }
48.         Group layer3 = new Group();
49.         for(int i=0; i<10;i++) {
50.             Circle circle = new Circle(150, Color.web("white", 0.05f));
51.             circle.setStrokeType(StrokeType.OUTSIDE);

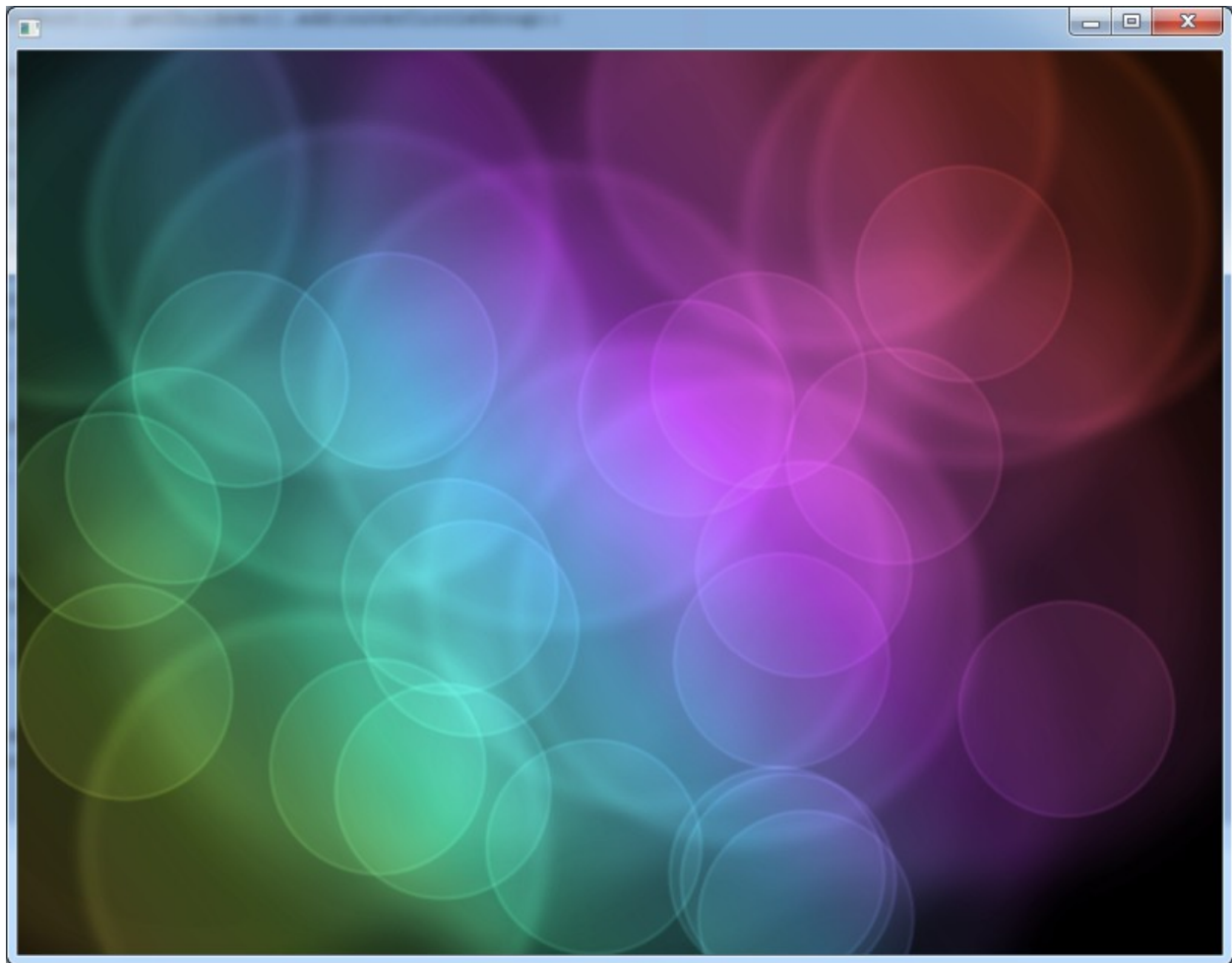
```

```

52.         circle.setStroke(Color.web("white",0.16f));
53.         circle.setStrokeWidth(4f);
54.         layer3.getChildren().add(circle);
55.     }
56.     layer1.setEffect(new BoxBlur(30,30,3));
57.     layer2.setEffect(new BoxBlur(2,2,2));
58.     layer3.setEffect(new BoxBlur(10,10,3));
59.     Rectangle colors = new Rectangle(scene.getWidth(), scene.getHeight(),
60.         new LinearGradient(0f,1f,1f,0f,true, CycleMethod.NO_CYCLE,new Stop[]{
61.             new Stop(0,Color.web("#f8bd55")),
62.             new Stop(0.14f,Color.web("#c0fe56")),
63.             new Stop(0.28f,Color.web("#5dfbc1")),
64.             new Stop(0.43f,Color.web("#64c2f8")),
65.             new Stop(0.57f,Color.web("#be4af7")),
66.             new Stop(0.71f,Color.web("#ed5fc2")),
67.             new Stop(0.85f,Color.web("#ef504c")),
68.             new Stop(1,Color.web("#f2660f")),
69.         });
70. };
71. Group outerCircleGroup = new Group(
72.     new Group(new Rectangle(scene.getWidth(), scene.getHeight(), Color.BLACK),
73.         layer1, layer2, layer3),
74.     colors );
75. colors.setBlendMode(BlendMode.OVERLAY);
76. ((Group)scene.getRoot()).getChildren().add(outerCircleGroup);
77. stage.show();
78. List<Node> allCircles = new ArrayList<Node>();
79. allCircles.addAll(layer1.getChildren());
80. allCircles.addAll(layer2.getChildren());
81. allCircles.addAll(layer3.getChildren());
82. Timeline timeline = new Timeline();
83. for(Node circle: allCircles) {
84.     timeline.getKeyFrames().addAll(
85.         new KeyFrame(Duration.ZERO,
86.             new KeyValue(circle.translateXProperty(),random()*800),
87.             new KeyValue(circle.translateYProperty(),random()*600)
88.         ),
89.         new KeyFrame(new Duration(40000),
90.             new KeyValue(circle.translateXProperty(),random()*800),
91.             new KeyValue(circle.translateYProperty(),random()*600)
92.         )
93.     );
94. }
95. timeline.setAutoReverse(true);
96. timeline.setCycleCount(Animation.INDEFINITE);
97. timeline.play();
98. }
99.
100. public static void main(String[] args) {
101.     Application.launch(args);
102. }
103. }

```

- You have a nice looking up running like this snap shot.



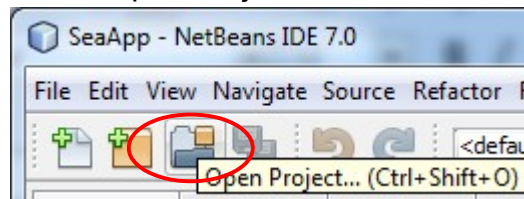
Exercise 2: A Bit of Everything

In this exercise we are going to create a small sample that tries to include some of the coolest features of JavaFX 2. We are going to use images, apply transformations, create animations and a lot of cool stuff.

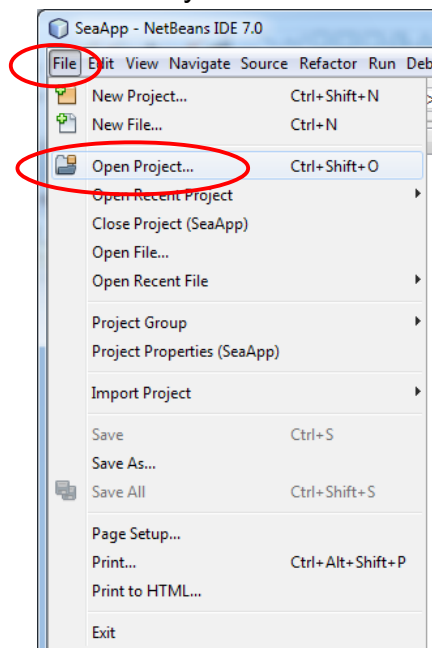
Running the solution

First, let's run the solution. It's important to have a clear idea of what you will be building.

- The HOL includes a **solutions** directory that contains a NetBeans project with the solution for each exercise. To open the solution project you have two options:
 - Option 1: Click on the Open Project icon

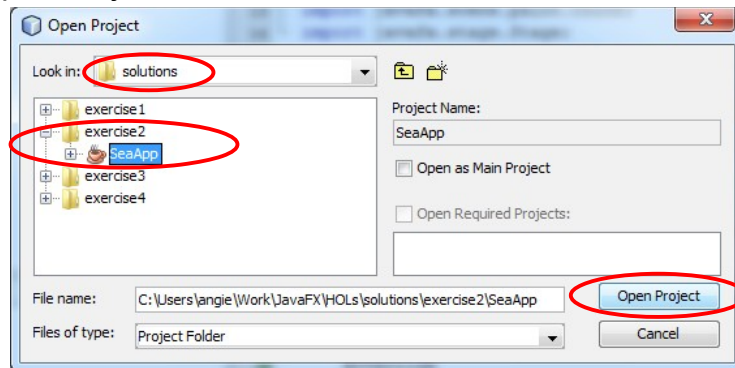


- Option 2: From the File menu you can select the Open Project option

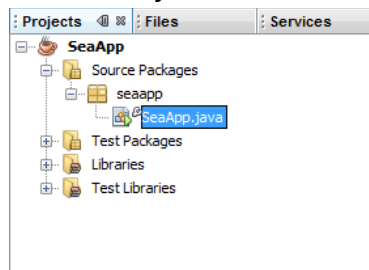


- In the Open Project screen:
 - Select the directory where you unzipped the HOL file.

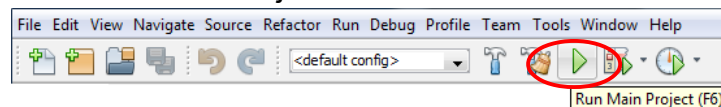
- Select the solutions directory
- Expands exercise2, and select the SeaApp Application
- Click Open Project



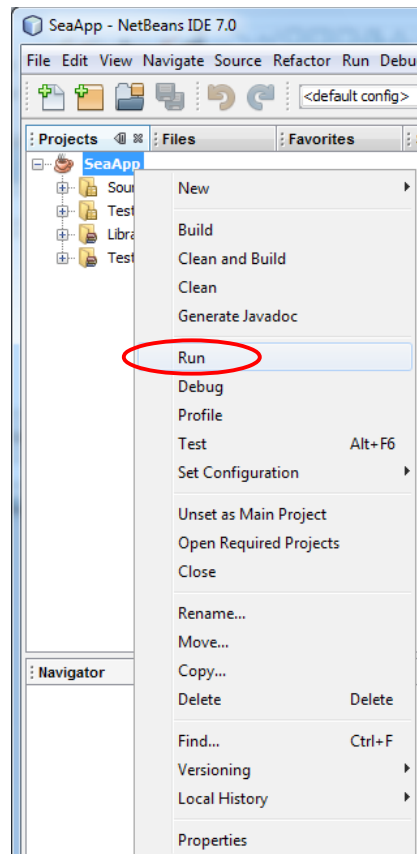
- You should see SeaApp in the Project list.



- Run the SeaApp project:
 - Option 1: Click the Run Project icon



- Option 2: Select SeaApp project and right click on it. From the popup menu select Run



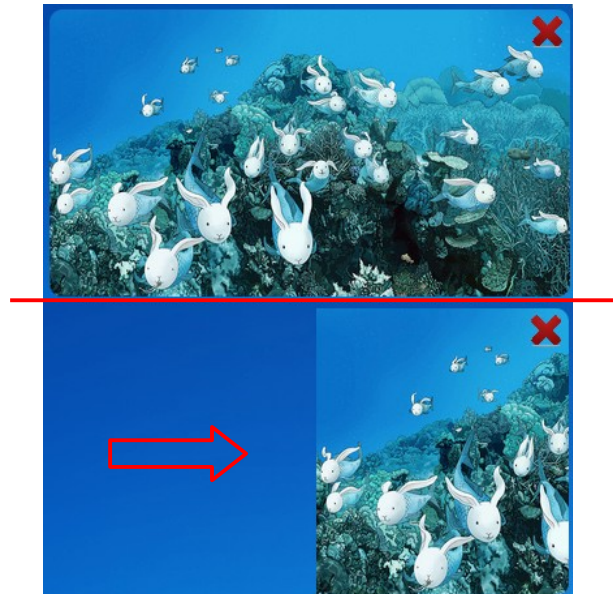
- You should see the app running.

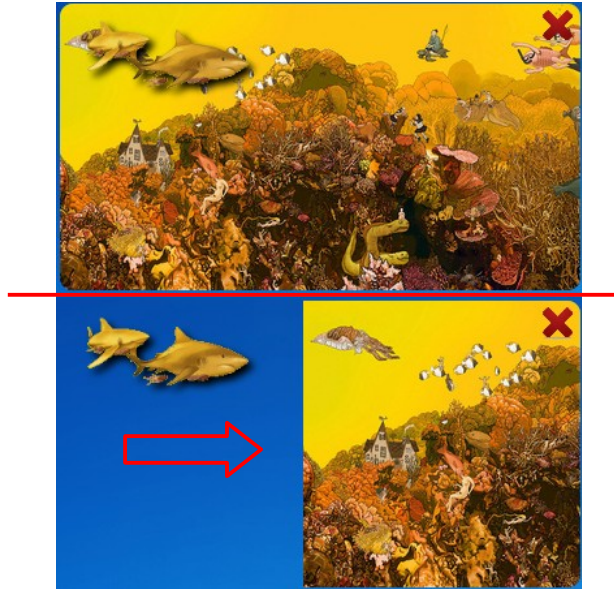


- Move the mouse over the application and notice that a new image gets displayed. In this exercise we are going to see how we can create animations to manipulate the transparency of the UI objects, and create nice transitions between screens.

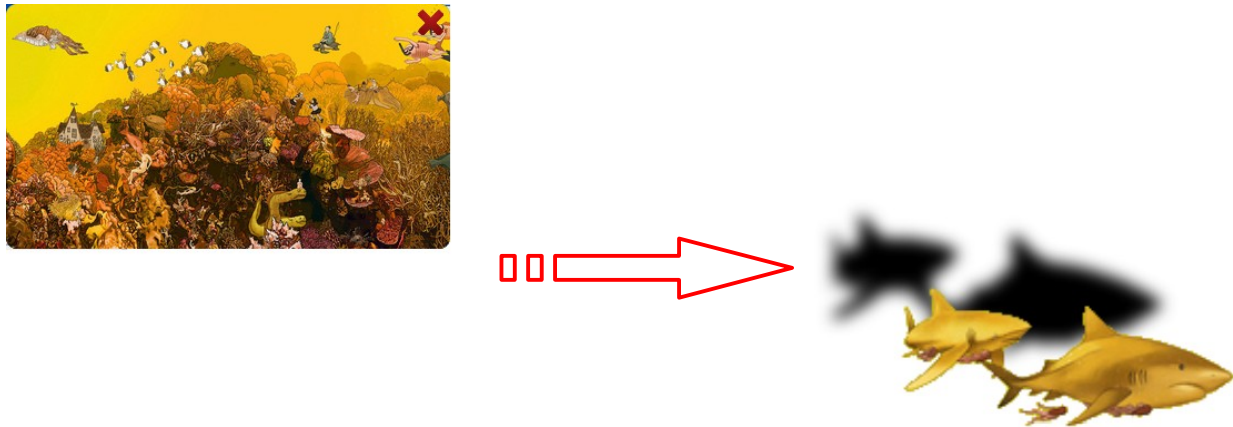


- The two background images (blue sea and yellow sea) are bound with the mouse enter and mouse exit events. Lets try now to move the mouse out of the application one more time, and see how the blue background appear again as you leave it. In this exercise, we will see how we can bind the opacity value of those images to some properties associated to the mouse enter and exit events.
- The x coordinate of the background images are also bound with mouse drag events. Try this feature by pressing the mouse, and dragging it along the x coordinates. Drag it all the way to the end of the background. Notice that the two sharks at the top are a separate image, and this one is not bind with the mouse events, they just stay there as you drag the mouse. Again you will learn how to program all this.





- Click on the two sharks, and see how they swim away. In this lab you will create this animation, and associate it to get triggered by the mouse click events.



- Close the application by clicking on the close icon. Now you are ready to start creating this application.



Getting Started With Your Application

- Create a new JavaFX project called SeaApp. You can place it any where you want it, but it has to be outside the `solutions` directory. (For detailed instructions how to create a JavaFX 2 projects, you can go back to exercise 1.)
- By default the generated code gets displayed, you should see something like this:

```
4. ...
5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.event.ActionEvent;
9. import javafx.event.EventHandler;
10. import javafx.scene.Group;
11. import javafx.scene.Scene;
12. import javafx.scene.control.Button;
13. import javafx.scene.paint.Color;
14. import javafx.stage.Stage;
15.
16. /**
17.  *
18.  * @author angie
19.  */
20. public class SeaApp extends Application {
21.
22.     /**
23.      * @param args the command line arguments
24.      */
25.     public static void main(String[] args) {
26.         Application.launch(SeaApp.class, args);
27.     }
28.
29.     @Override
30.     public void start(Stage primaryStage) {
31.         primaryStage.setTitle("Hello World");
32.         Group root = new Group();
33.         Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
34.         Button btn = new Button();
35.         btn.setLayoutX(100);
36.         btn.setLayoutY(80);
37.         btn.setText("Hello World");
38.         btn.setOnAction(new EventHandler<ActionEvent>() {
39.
40.             public void handle(ActionEvent event) {
41.                 System.out.println("Hello World");
42.             }
43.         });
44.         root.getChildren().add(btn);
45.         primaryStage.setScene(scene);
46.         primaryStage.show();
47.     }
48. }
```

Table 1.

Lets study the generated code.

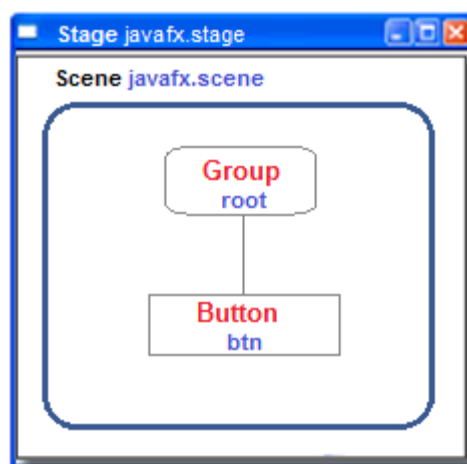
- Every time you want to create a JavaFX application you have to extend the `Class Application`. (See line 20th)

- To launch your standalone application we use the static method `launch`: `Application.launch(SeaApp.class, args);` (See line 26th). This method expects two parameters, the first one is the `Application`'s subclass, and the second any parameters you need to provide to your application.
- The main entry point for all JavaFX applications is the `start` method. This method is called after the `init` method has returned, and after the system is ready for the application to begin running.

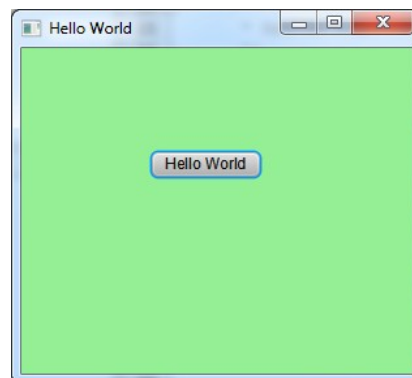
This method receives the primary stage for this application, onto which the application scene can be set. Before we go any further in this code, let's explain a bit the scene graph programming model used by JavaFX.

In JavaFX, all the UI content is structured as a scene graph. A scene graph is a collection of nodes in a tree structure. A node may have many children but often only a single parent, with the effect of a parent applied to all its child nodes; an operation performed on a group automatically propagates its effect to all of its members. A common feature, for instance, is the ability to group related shapes/objects into a compound object that can then be moved, transformed, selected, etc. as easily as a single object.

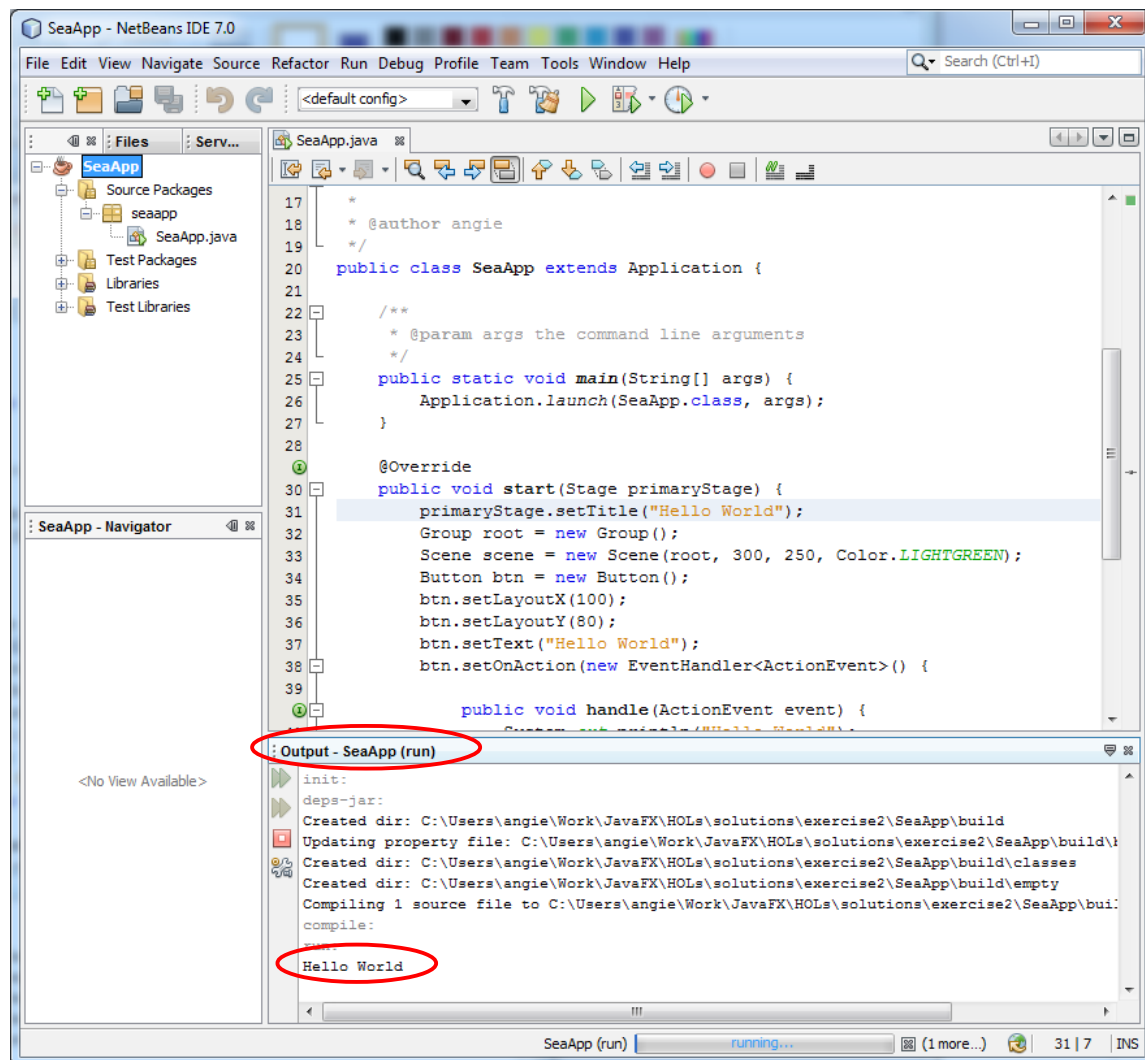
For JavaFX, the stage is the top-level container for the application, and the scene is the drawing surface for the application's content. The generated code has a very simple structure, just a `Group` for the root of the tree, and one node, in this case a `Button` `btn`.



- Line 31st sets the stage's title.
- Line 33rd creates the scene. Notice that in the constructor we pass the root of the tree, in this case the root is a Group created in line 32nd. You can also pass in the scene's constructor the size of the display area and the background color.
- Line 34th creates a Button called `btn`. Lines 35th - 37th sets some Button's attributes like the layouts X and Y and the text to be displayed on it.
- In lines 38th - 43rd we create an event handler associated to the button's action. In our case, we are just doing a simple printout "Hello World" each time the button gets clicked.
- Line 44th is very important, once we create a node we need to add it to the scene graph. In this case the button `btn` will go directly in the root, so we need to add it to it. See that we are adding this `btn` to the already existent root's children, in our case our root doesn't have any children yet, but this is the way we always add nodes in the scene graph tree.
- Finally we set the stage's scene(Line 45th), and we make the stage visible (Line 46th).
- Run the sample. (If you need details of how to run the sample, go back to exercise 1)

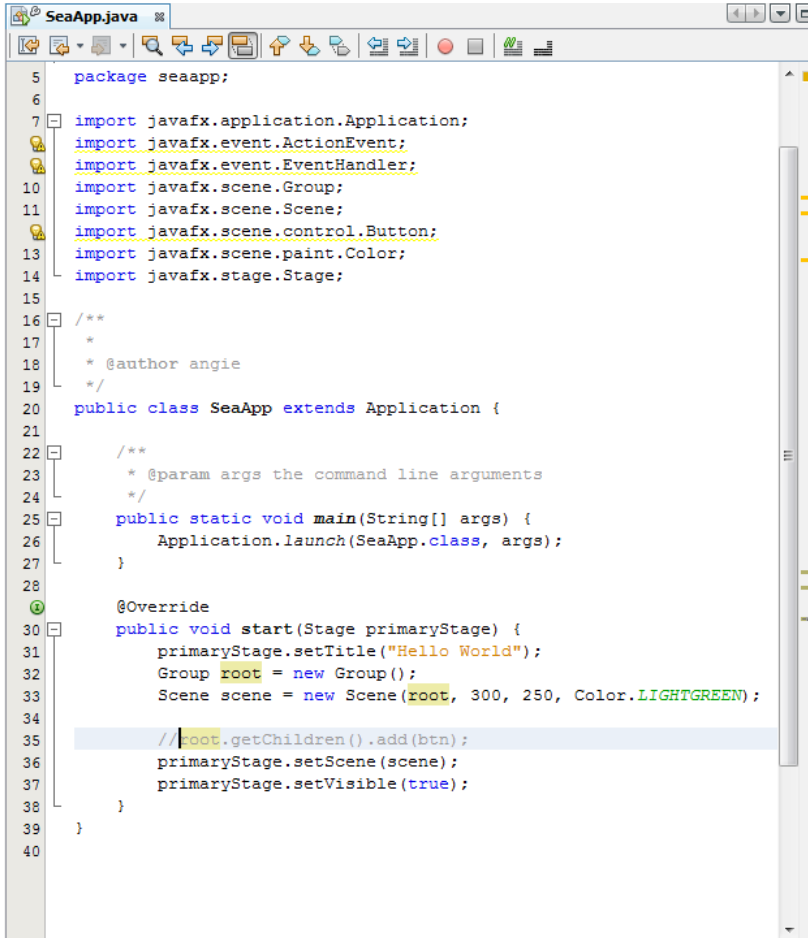


- Click on the Button and see the “Hello World” in the Output area in NetBeans.



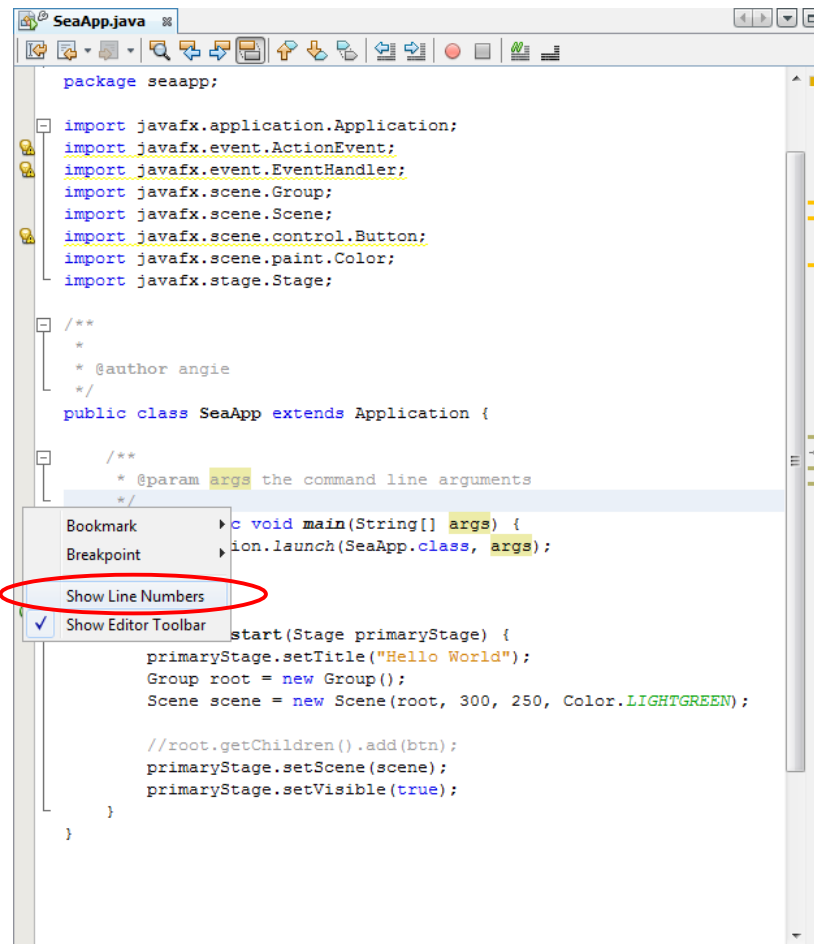
- Close the application.

- Delete lines 34th to 43rd, and comment out the line 44th. Your SeaApp.java class should look like this

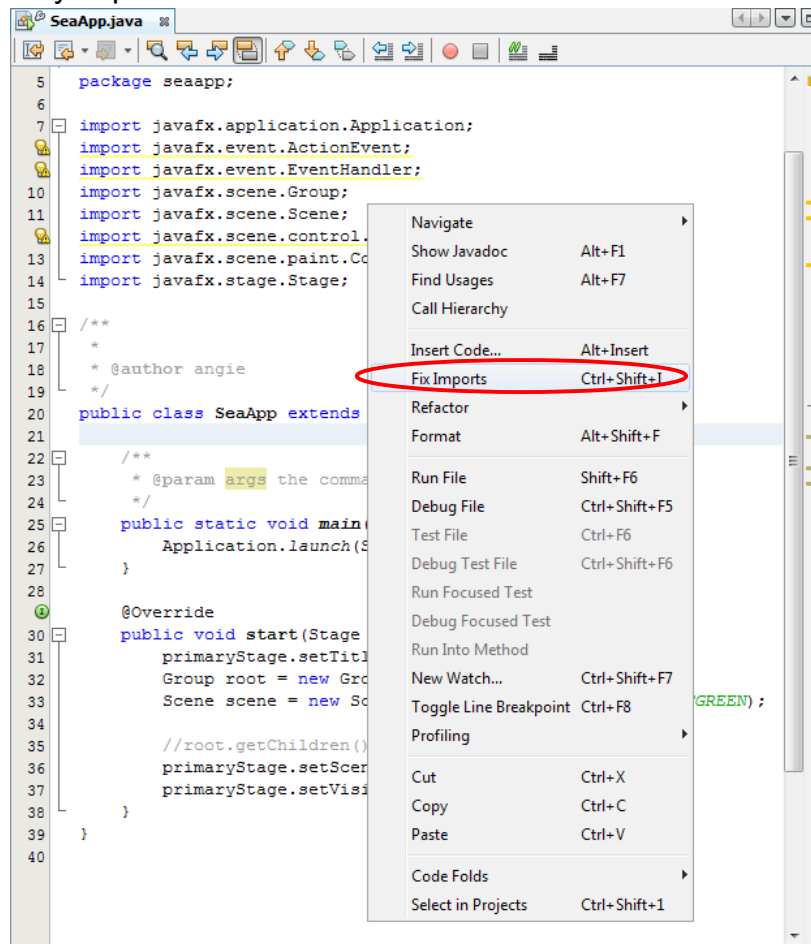


```
5 package seaapp;
6
7 import javafx.application.Application;
8 import javafx.event.ActionEvent;
9 import javafx.event.EventHandler;
10 import javafx.scene.Group;
11 import javafx.scene.Scene;
12 import javafx.scene.control.Button;
13 import javafx.scene.paint.Color;
14 import javafx.stage.Stage;
15
16 /**
17  *
18  * @author angie
19  */
20 public class SeaApp extends Application {
21
22     /**
23      * @param args the command line arguments
24      */
25     public static void main(String[] args) {
26         Application.launch(SeaApp.class, args);
27     }
28
29     @Override
30     public void start(Stage primaryStage) {
31         primaryStage.setTitle("Hello World");
32         Group root = new Group();
33         Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
34         //root.getChildren().add(btn);
35         primaryStage.setScene(scene);
36         primaryStage.setVisible(true);
37     }
38 }
39
40
```

- If you can't see the lines on the left hand side, right click on that space, and select "Show Line Numbers"

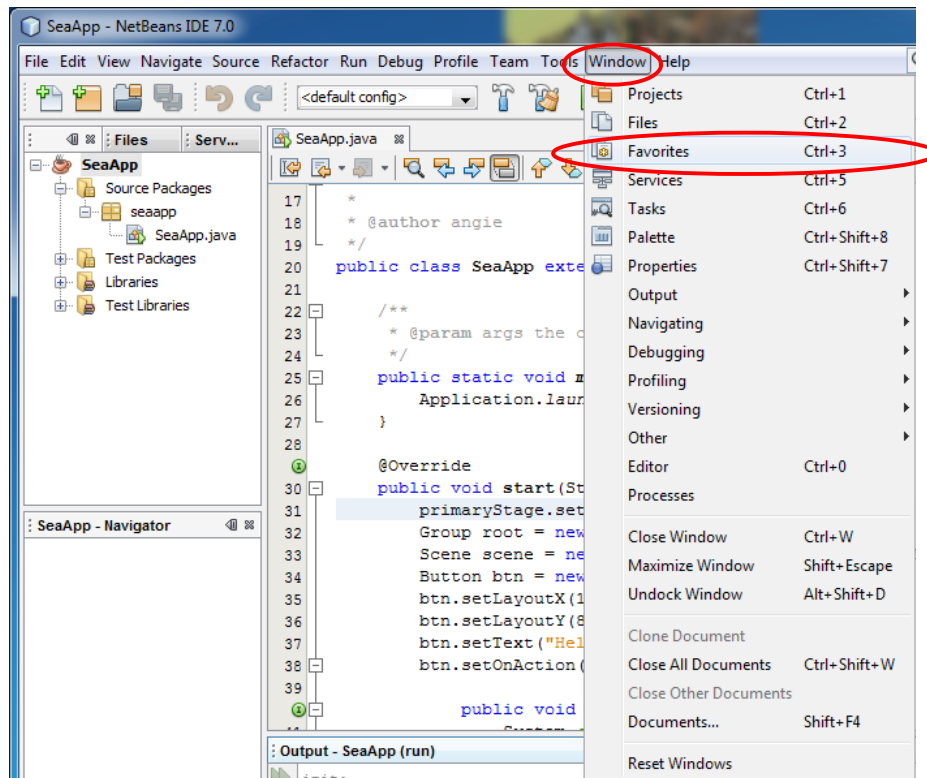


- Right click on the background, and select “Fix Imports”, this will get rid of any unnecessary imports we still have in the class.

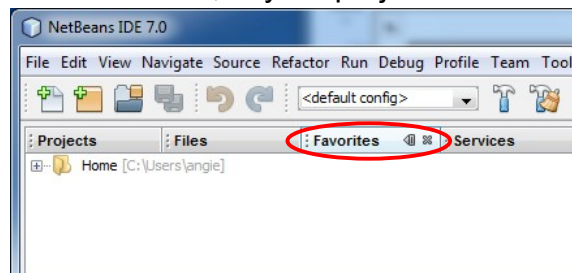


- Now we are ready to start creating our new application.

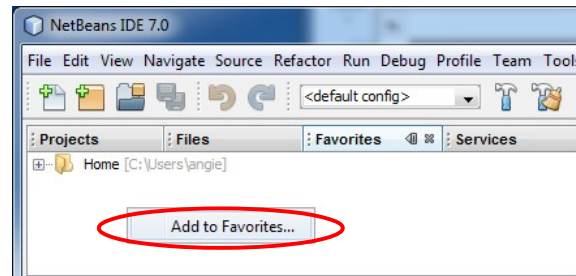
- First, we need some resources (images) for our project.
 - Click on the Window menu, and then click on favorites.



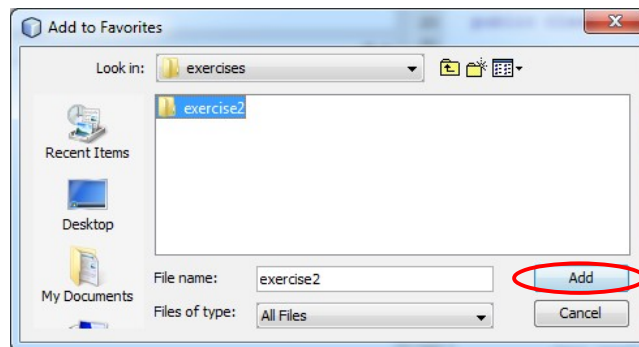
- This will add a Favorites tab, in your project area



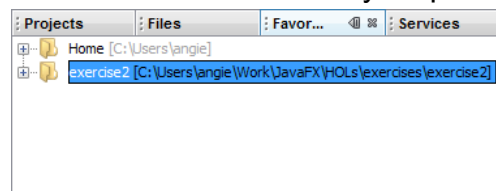
- Right click in the Favorites' white area, and select “Add to Favorites...”



- The “Add to Favorites” window gets displayed. Open the directory where you unzip your HOL zip file. Open the “exercises” directory, and select “exercise2”. Click on the Add button.

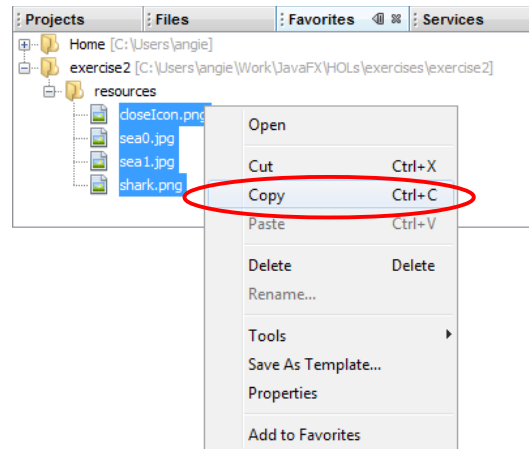


- Now you can see the “exercise2” directory as part of the favorites' list.

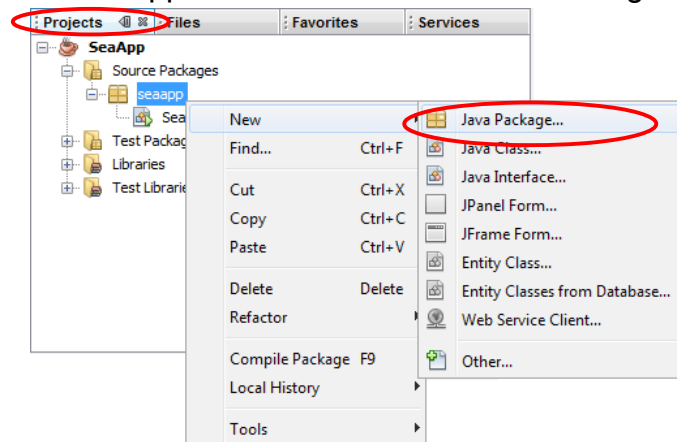


- Expands exercise2 → resources
- Select sea0.jpg, sea1.jpg, shark.png and closelcon.png files.

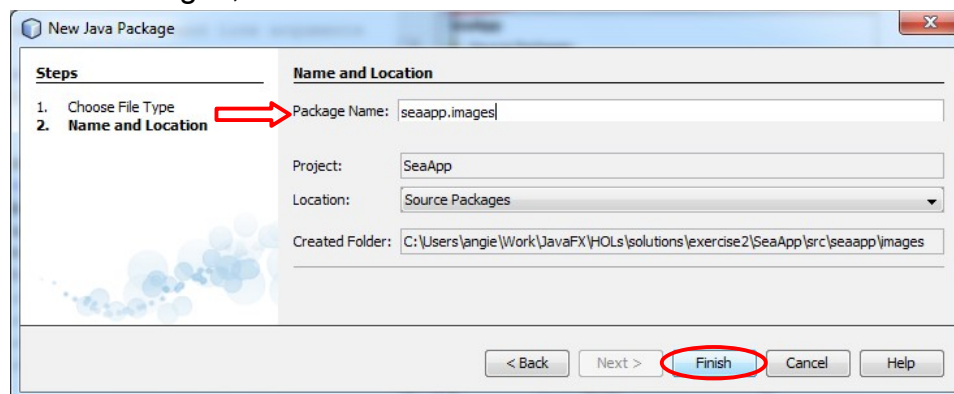
- Right click on them, and select copy.



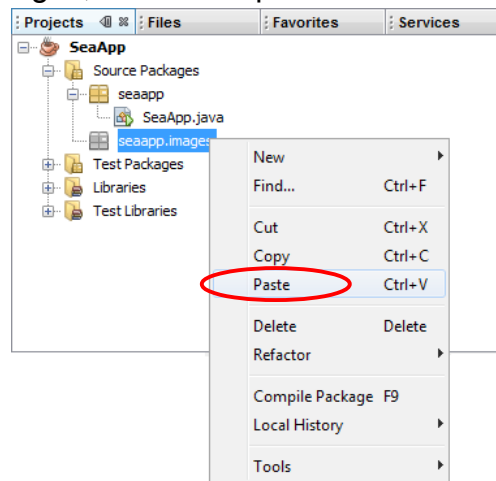
- Now, lets go back to Projects tab.
- Expands SeaApp → Source Packages → seaapp
- Right click on seaapp and select New → Java Package



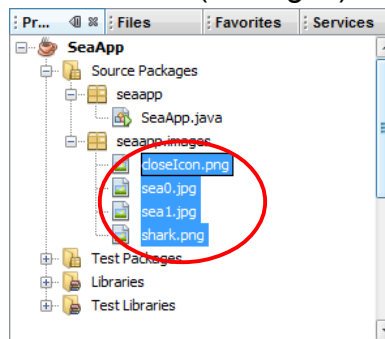
- Name it images, and click Finish



- Right click on images, and select paste.



- Now you can see your resources (4 images) as part of your project.



- Now lets add the code to show an image. There are two classes you should be aware of: `javafx.scene.image.Image` and `javafx.scene.image.ImageView`.

The `Image` class represents graphical images and is used for loading images from a specified URL. The `ImageView` is a `Node` used for painting images loaded with `Image` class, and this is the one you add to the scene graph.

- First we need to declare the `ImageView` variable

```
private ImageView sea0;
```

- In the `start` method we can create the image:
 - We need to create an `Image`, providing the location of the image to be displayed.

- Then create an `ImageView` with the `Image` just created. The code should look like this:

```
sea0 = new ImageView(  
    new Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")) );
```

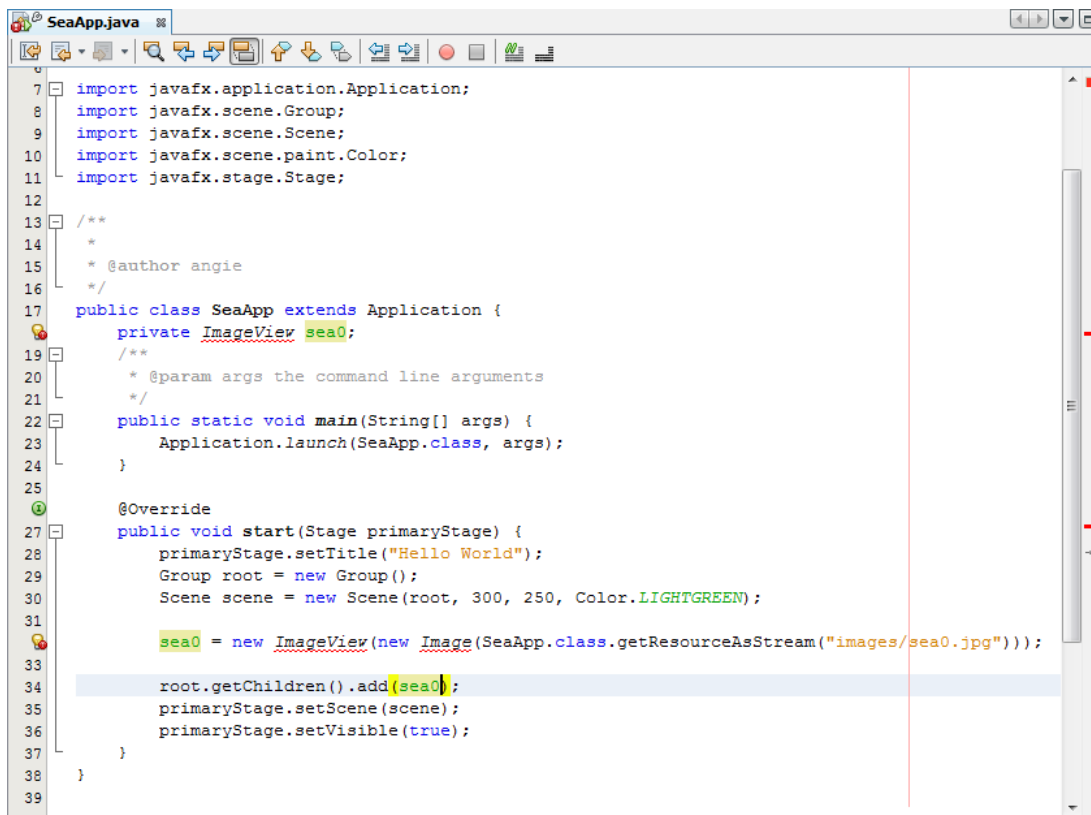
- Now we need to add the `ImageView` to the scene graph. The simplest way is to add it to the root.
Uncomment the following line from your code

```
//root.getChildren().add(btn);
```

And replace it with

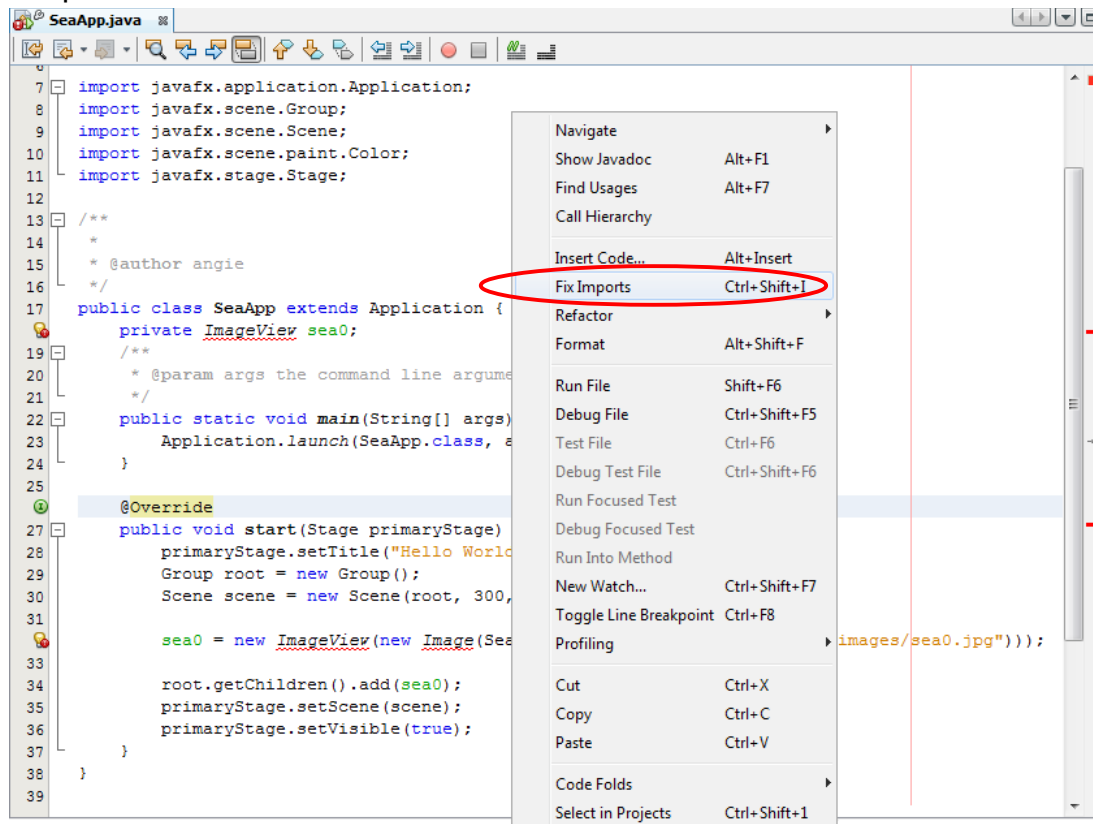
```
root.getChildren().add(sea0);
```

- Your code should look like this:

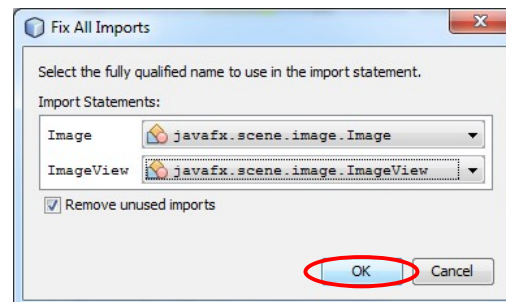


```
1  SeaApp.java
2
3  7 import javafx.application.Application;
4  8 import javafx.scene.Group;
5  9 import javafx.scene.Scene;
6 10 import javafx.scene.paint.Color;
7 11 import javafx.stage.Stage;
8 12
9 13 /**
10 14  *
11 15  * @author angie
12 16  */
13 17 public class SeaApp extends Application {
14 18     private ImageView sea0;
15 19
16 20     /**
17 21      * @param args the command line arguments
18 22      */
19 23     public static void main(String[] args) {
20 24         Application.launch(SeaApp.class, args);
21 25     }
22 26
23 27     @Override
24 28     public void start(Stage primaryStage) {
25 29         primaryStage.setTitle("Hello World");
26 30         Group root = new Group();
27 31         Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
28 32
29 33         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")));
30 34
31 35         root.getChildren().add(sea0);
32 36         primaryStage.setScene(scene);
33 37         primaryStage.setVisible(true);
34 38     }
35 39 }
```

- Notice that line 18th and line 32nd are highlighting some errors. This is because we are missing some imports. Right click on the background and select “Fix imports...”



- NetBeans might find more than one instance of `Image` and `ImageView` class, and it will prompt you to select the correct ones. Make sure you select the ones from `javafx.scene.image` package.



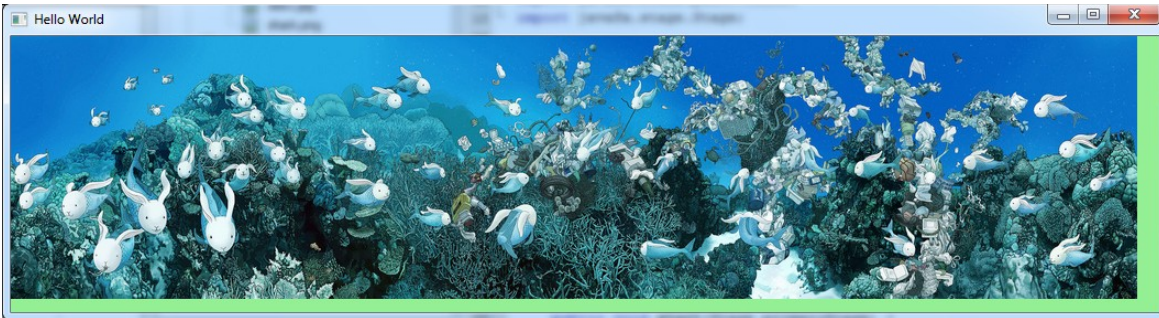
- Your code should look like this

```
SeaApp.java
5 package seaapp;
6
7 import javafx.application.Application;
8 import javafx.scene.Group;
9 import javafx.scene.Scene;
10 import javafx.scene.image.Image;
11 import javafx.scene.image.ImageView;
12 import javafx.scene.paint.Color;
13 import javafx.stage.Stage;
14
15 /**
16  *
17  * @author angie
18  */
19 public class SeaApp extends Application {
20     private ImageView sea0;
21
22     /**
23      * @param args the command line arguments
24      */
25     public static void main(String[] args) {
26         Application.launch(SeaApp.class, args);
27     }
28
29     @Override
30     public void start(Stage primaryStage) {
31         primaryStage.setTitle("Hello World");
32         Group root = new Group();
33         Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
34
35         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")));
36
37         root.getChildren().add(sea0);
38         primaryStage.setScene(scene);
39         primaryStage.setVisible(true);
40     }
41 }
```

- Run your application.



- Resize the window to your right, and notice that your image is really long.



- We can set a Clip area for this image. In our case we are going to use a Rectangle with rounded corners.
 - Define the clip. (Add the following code after line 20th)

```
private Rectangle seaClip0;
```
 - Create the rectangle and set the rounded corners. (Add the following code at line 35th)

```
seaClip0 = new Rectangle(400, 220);  
seaClip0.setArcHeight(20);  
seaClip0.setArcWidth(20);
```
 - Set the clip for the image. (Add the following code at line 40th)

```
sea0.setClip(seaClip0);
```
 - Right click on the background and select “Fix imports...”. When prompted make sure you select `Rectangle` from `javafx.scene.shape` package.

- Your code should now look like this:

```
5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.scene.Group;
9. import javafx.scene.Scene;
10. import javafx.scene.image.Image;
11. import javafx.scene.image.ImageView;
12. import javafx.scene.paint.Color;
13. import javafx.scene.shape.Rectangle;
14. import javafx.stage.Stage;
15.
16. /**
17.  *
18.  * @author angie
19.  */
20. public class SeaApp extends Application {
21.     private ImageView sea0;
22.     private Rectangle seaClip0;
23.     /**
24.      * @param args the command line arguments
25.      */
26.     public static void main(String[] args) {
27.         Application.launch(SeaApp.class, args);
28.     }
29.
30.     @Override
31.     public void start(Stage primaryStage) {
32.         primaryStage.setTitle("Hello World");
33.         Group root = new Group();
34.         Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
35.
36.         seaClip0 = new Rectangle(400, 220);
37.         seaClip0.setArcHeight(20);
38.         seaClip0.setArcWidth(20);
39.
40.         sea0 = new ImageView(new
41.             Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")));
42.         sea0.setClip(seaClip0);
43.
44.         root.getChildren().add(sea0);
45.         primaryStage.setScene(scene);
46.         primaryStage.show();
47.     }
```

- One cool trick you can do in your application is to make the stage transparent, this means that the application won't have a window around the image. The application will blend nicely with your desktop's background.

- Remove line 32nd where you set the title for you window.
- Add a new line with the following code

```
primaryStage.initStyle(StageStyle.TRANSPARENT);
```

- Right click on the background and fix the imports
- You can now remove the color green we use for the scene background.
Update the following line

```
Scene scene = new Scene(root, 300, 250, Color.LIGHTGREEN);
```

with

```
Scene scene = new Scene(root, 400, 250, null);
```

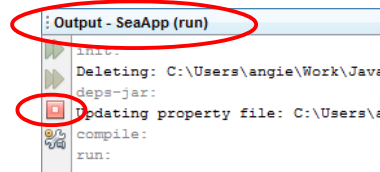
- Finally your code should look like this:

```
1. package seaapp;
2.
3. import javafx.application.Application;
4. import javafx.scene.Group;
5. import javafx.scene.Scene;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.paint.Color;
9. import javafx.scene.shape.Rectangle;
10. import javafx.stage.Stage;
11. import javafx.stage.StageStyle;
12.
13. /**
14.  *
15.  * @author angie
16.  */
17. public class SeaApp extends Application {
18.     private ImageView sea0;
19.     private Rectangle seaClip0;
20.     /**
21.      * @param args the command line arguments
22.      */
23.     public static void main(String[] args) {
24.         Application.launch(SeaApp.class, args);
25.     }
26.
27.     @Override
28.     public void start(Stage primaryStage) {
29.         primaryStage.initStyle(StageStyle.TRANSPARENT);
30.         Group root = new Group();
31.         Scene scene = new Scene(root, 400, 250, null);
32.
33.         seaClip0 = new Rectangle(400, 220);
34.         seaClip0.setArcHeight(20);
35.         seaClip0.setArcWidth(20);
36.
37.         sea0 = new ImageView(new
38.             Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")));
39.         sea0.setClip(seaClip0);
40.
41.         root.getChildren().add(sea0);
42.         primaryStage.setScene(scene);
43.         primaryStage.show();
44.     }
45. }
```

- Run the application and notice all the changes



- Now, the first issue we are facing is that we don't have a close icon. Fortunately you can use NetBeans for this.
 - In the NetBeans' output area, there is red square, click on this one to stop the application.



Getting Started With Events

- Now, let's add another image, the exit icon. (Detailed steps at the beginning of exercise 2.)
 - Declare an `ImageView` and call it `quit`
 - Inside the `start` method, create the `quit` `ImageView`.
 - Set the image's height and width to 25 (use methods `setFitHeight`, `setFitWidth`)
 - Set the image's x coordinate to 370 (use method `setLayoutX`)
 - Set the image's y coordinate to 5 (use method `setLayoutY`)
 - Make sure you add `quit` to the scene graph, directly to the root.
 - To add multiple children at the same time, use `addAll` method instead of `add`, and provide all the nodes as parameters.
- We need to add some code to handle the mouse click events on the `quit` Image. Basically, when the user clicks on the image, we want the application to quit.

```
quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
    public void handle(MouseEvent me) {
        System.exit(0);
    }
}));
```

We need to call the method `setOnMouseClicked` to set the event handler for the click events. As a parameter in our code we are providing an anonymous `EventHandler`, and at the same time we are providing the method `handle` that will be invoked every time the `quit` image gets a click.

- Adding this piece of code will show some errors. Make sure you add the required imports (also shown at the beginning of this exercise.)
- Your code should look like this:

```
5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.event.EventHandler;
9. import javafx.scene.Group;
10. import javafx.scene.Scene;
11. import javafx.scene.image.Image;
12. import javafx.scene.image.ImageView;
13. import javafx.scene.input.MouseEvent;
14. import javafx.scene.shape.Rectangle;
15. import javafx.stage.Stage;
16. import javafx.stage.StageStyle;
17.
18. /**
19.  *
20.  * @author angie
21.  */
22. public class SeaApp extends Application {
23.     private ImageView sea0;
24.     private Rectangle seaClip0;
25.     private ImageView quit;
26.
27.     /**
28.      * @param args the command line arguments
29.      */
30.     public static void main(String[] args) {
31.         Application.launch(SeaApp.class, args);
32.     }
33.
34.     @Override
35.     public void start(Stage primaryStage) {
36.         primaryStage.initStyle(StageStyle.TRANSPARENT);
37.         Group root = new Group();
38.         Scene scene = new Scene(root, 400, 250, null);
39.
40.         seaClip0 = new Rectangle(400, 220);
41.         seaClip0.setArcHeight(20);
42.         seaClip0.setArcWidth(20);
43.
44.         sea0 = new ImageView(new
Image(SeaApp.class.getResourceAsStream("images/sea0.jpg")));
45.         sea0.setClip(seaClip0);
46.
47.         quit = new ImageView(new
Image(SeaApp.class.getResourceAsStream("images/closeIcon.png")))
;
48.         quit.setFitHeight(25);
49.         quit.setFitWidth(25);
50.         quit.setLayoutX(370);
51.         quit.setLayoutY(5);
52.
53.         quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
```

```
54.
55.         public void handle(MouseEvent me) {
56.             System.exit(0);
57.         }
58.     }));
59.     root.getChildren().addAll(sea0, quit);
60.     primaryStage.setScene(scene);
61.     primaryStage.show();
62. }
63. }
```

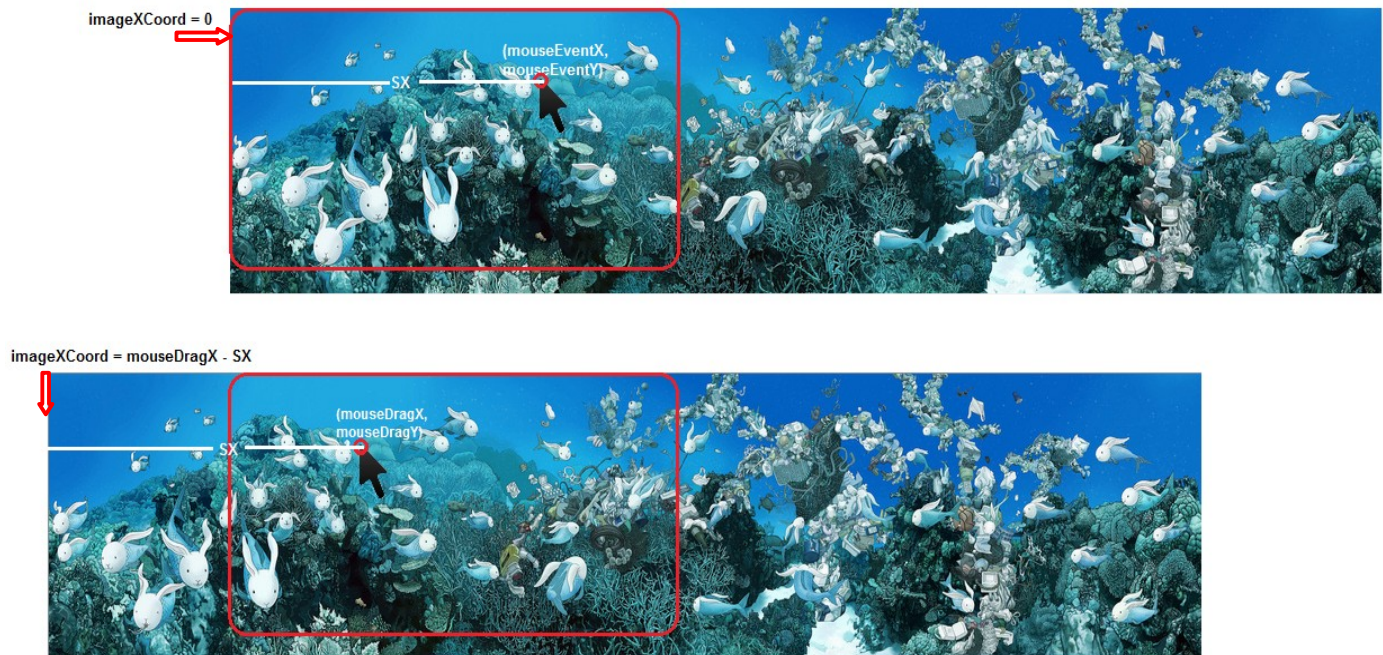
- Run your application.



- Click on the quit image, and make sure the application closes.

Now, let's play a bit more with mouse events and binding. We know image `sea0` is very wide, and we are just seeing a piece of it through the clip. What we want to do, is bind the x coordinate of the image with the x coordinate of the mouse drag event. The result will be that as you drag the mouse the image will be moving along with it.

Let's see our scenario:



There are few mouse events that get generated. First, there will be a mouse pressed event followed by a number of mouse drag events. Finally, once you release your mouse, a mouse release event will be generated. In our case we are interested in two of these three events, mouse pressed and mouse dragged.

On mouse pressed we need to find out the distance between the `mouseEventX` and the original x coordinate of the image. We can call it `SX`. As we drag the mouse around, we need to keep this distance, as we want the image to smoothly move along with the drag movement.

Let's see the code:

- First we need to declare two variables, one will hold the real value of the image's x coordinate and the second will hold the `sX` distance.


```
private double sX = 0;
private double coordXReal = 0;
```

- Now we need to implement `setOnMousePressed` and `setOnMouseDragged` for `sea0 ImageView`. (This code will go inside the `start` method)

```
sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent me) {
        sX = me.getSceneX() - coordXReal;
    }
});

sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent me) {
        coordXReal = me.getSceneX() - sX;
    }
});
```

Now, let's do some binding. Binding is a powerful mechanism for expressing direct relationships between variables. When objects participate in bindings, changes made to one object will automatically be reflected in another object.

Bindings are assembled from one or more sources, known as dependencies. A binding observes its dependencies for changes and updates its own value according to changes in the dependencies.

In JavaFX we normally use Properties as source of bindings. There are properties for each primitive data type, for example: `DoubleProperty`, `FloatProperty`, `IntegerProperty`, `BooleanProperty`, and also `ObjectProperty` and `StringProperty`.

Let's see the code:

- Our first thought is to bind `sea0`'s x coordinate to `coordXReal`, but we need a property for that, so let's convert `coordXReal` into a Property. Because we are using a double value, we need to use a `DoubleProperty`.
- Change the line:

```
private double coordXReal = 0;
```

for this line:

```
private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
```

- Now update the event handlers as following:

```

    sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
        public void handle(MouseEvent me) {
            sX = me.getSceneX() - imageXProperty.getValue();
        }
    });

    sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
        public void handle(MouseEvent me) {
            imageXProperty.set(me.getSceneX() - sX);
        }
    });

```

- Now we are ready to add the binding

```

sea0.xProperty().bind(imageXProperty);

```

Every time `imageXProperty` change, the x coordinate of `sea0` gets updated.

Your code should look like this:

```

5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.beans.property.DoubleProperty;
9. import javafx.event.EventHandler;
10. import javafx.scene.Group;
11. import javafx.scene.Scene;
12. import javafx.scene.image.Image;
13. import javafx.scene.image.ImageView;
14. import javafx.scene.input.MouseEvent;
15. import javafx.scene.shape.Rectangle;
16. import javafx.stage.Stage;
17. import javafx.stage.StageStyle;
18.
19. /**
20.  *
21.  * @author angie
22.  */
23. public class SeaApp extends Application {
24.     private ImageView sea0;
25.     private Rectangle seaClip0;
26.     private ImageView quit;
27.
28.
29.     private double sX = 0;
30.     private DoubleProperty imageXProperty =
31.         new SimpleDoubleProperty(0);
32.
33.     /**
34.      * @param args the command line arguments
35.      */
36.     public static void main(String[] args) {
37.         Application.launch(SeaApp.class, args);
38.     }
39.
40.     @Override
41.     public void start(Stage primaryStage) {
42.         primaryStage.initStyle(StageStyle.TRANSPARENT);

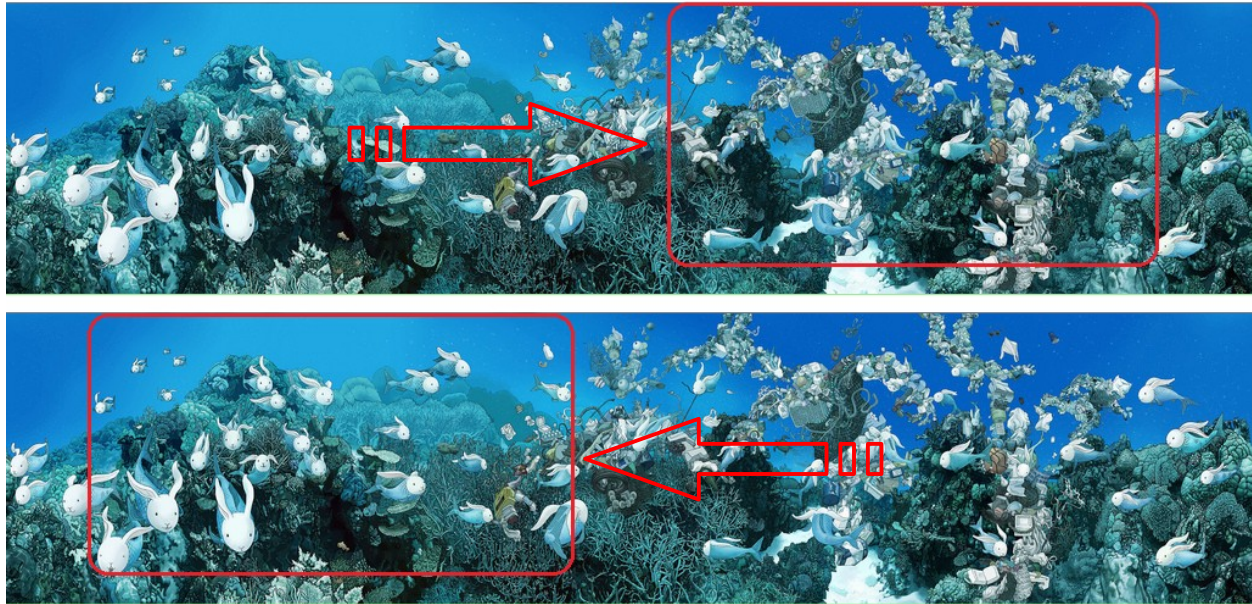
```

```

41.      Group root = new Group();
42.      Scene scene = new Scene(root, 400, 250, null);
43.
44.      seaClip0 = new Rectangle(400, 220);
45.      seaClip0.setArcHeight(20);
46.      seaClip0.setArcWidth(20);
47.
48.      sea0 = new ImageView(new Image(
49.          SeaApp.class.getResourceAsStream("images/sea0.jpg")));
50.      sea0.setClip(seaClip0);
51.
52.      sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
53.          public void handle(MouseEvent me) {
54.              sX = me.getSceneX() - imageXProperty.getValue();
55.          }
56.      });
57.
58.      sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
59.          public void handle(MouseEvent me) {
60.              imageXProperty.set(me.getSceneX() - sX);
61.          }
62.      });
63.      sea0.xProperty().bind(imageXProperty);
64.
65.      quit = new ImageView(new Image(SeaApp.class.
66.          getResourceAsStream("images/closeIcon.png")));
67.      quit.setFitHeight(25);
68.      quit.setFitWidth(25);
69.      quit.setLayoutX(370);
70.      quit.setLayoutY(5);
71.
72.      quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
73.          public void handle(MouseEvent me) {
74.              System.exit(0);
75.          }
76.      }));
77.      root.getChildren().addAll(sea0, quit);
78.      primaryStage.setScene(scene);
79.      primaryStage.show();
80.  }

```

Run the code and make sure you can drag the mouse to move the image around. Remember we only bind the x coordinates, so the image's movement is horizontally.



Before we continue with animations, let's organize the code a little bit.

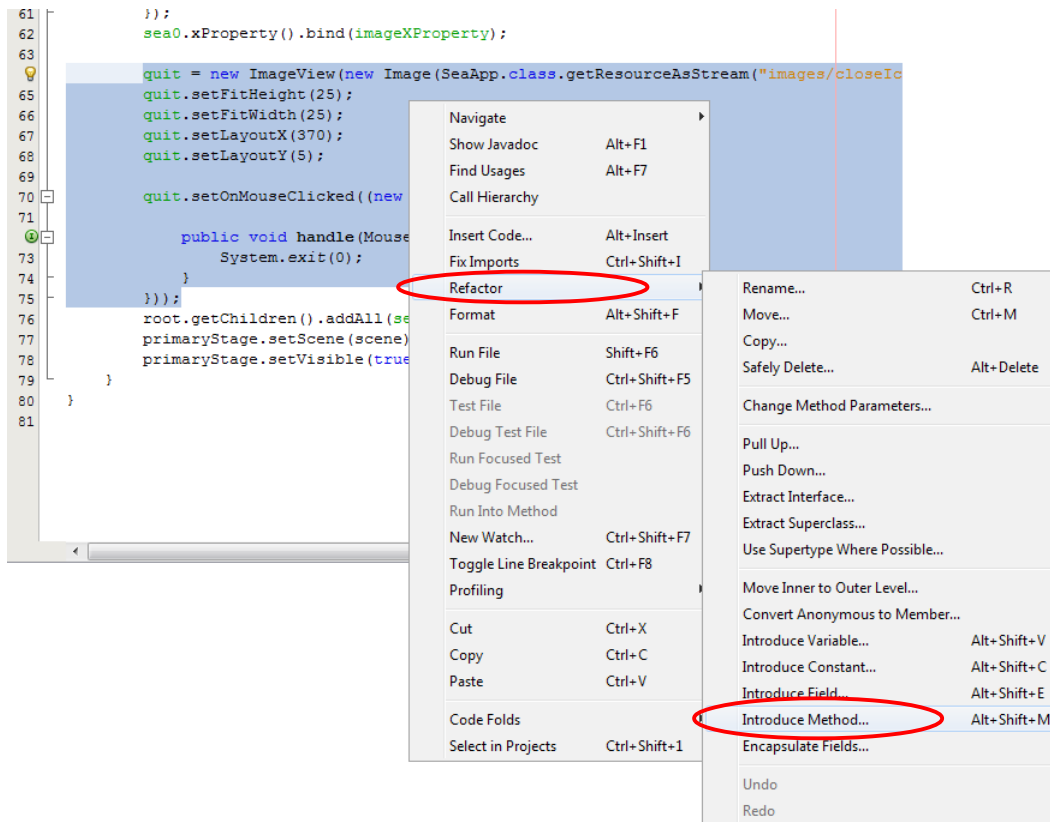
- Select lines 64th - 75th

```

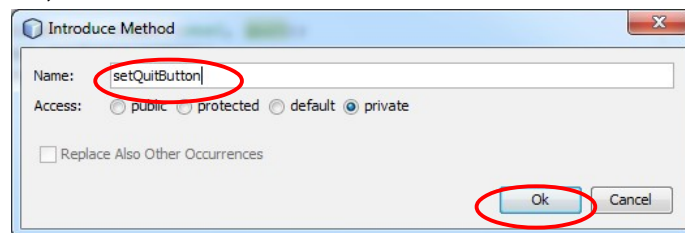
60 |         }
61 |     });
62 |     sea0.xProperty().bind(imageXProperty);
63 |
64 |     quit = new ImageView(new Image(SeaApp.class.getResourceAsStream("images/clo...
65 |     quit.setFitHeight(25);
66 |     quit.setFitWidth(25);
67 |     quit.setLayoutX(370);
68 |     quit.setLayoutY(5);
69 |
70 |     quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
71 |
72 |         public void handle(MouseEvent me) {
73 |             System.exit(0);
74 |         }
75 |     }));
76 |     root.getChildren().addAll(sea0, quit);
77 |     primaryStage.setScene(scene);
78 |     primaryStage.setVisible(true);
79 | }
80 |

```

- Right click on the highlighted code, and select Refactor → Introduce Method... What this will actually going to do is to take this code out of the `start` method, and put it into a separate method, and replace all this code with the appropriate method call.



- The “Introduce Method” window will appear. Enter `setQuitButton` for the method's name, and click ok.



- Review the new code.

```

61         });
62         sea0.xProperty().bind(imageXProperty);
63
64         setQuitButton();
65         root.getChildren().addAll(sea0, quit);
66         primaryStage.setScene(scene);
67         primaryStage.setVisible(true);
68     }
69
70     private void setQuitButton() {
71         quit = new ImageView(new Image(SeaApp.class.getResourceAsStream("images/
72         quit.setFitHeight(25);
73         quit.setFitWidth(25);
74         quit.setLayoutX(370);
75         quit.setLayoutY(5);
76
77         quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
78
79             public void handle(MouseEvent me) {
80                 System.exit(0);
81             }
82         }));
83     }
84 }
85

```

- Now, let's do the same refactoring process for setting up the mouse events in **sea0**.
 - Select lines 51st - 61st
 - Name the method **settingUpDragging**
- Your code should look like this:

```

1. package seaapp;
2.
3. import javafx.application.Application;
4. import javafx.beans.property.DoubleProperty;
5. import javafx.event.EventHandler;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.image.Image;
9. import javafx.scene.image.ImageView;
10. import javafx.scene.input.MouseEvent;
11. import javafx.scene.shape.Rectangle;
12. import javafx.stage.Stage;
13. import javafx.stage.StageStyle;
14.
15. /**
16.  *
17.  * @author angie
18.  */
19. public class SeaApp extends Application {
20.     private ImageView sea0;
21.     private Rectangle seaClip0;
22.     private ImageView quit;
23.
24.
25.     private double sX = 0;
26.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
27.     /**
28.      * @param args the command line arguments
29.      */
30.     public static void main(String[] args) {
31.         Application.launch(SeaApp.class, args);
32.     }
33.
34.     @Override
35.     public void start(Stage primaryStage) {

```

```

36.     primaryStage.initStyle(StageStyle.TRANSPARENT);
37.     Group root = new Group();
38.     Scene scene = new Scene(root, 400, 250, null);
39.
40.     seaClip0 = new Rectangle(400, 220);
41.     seaClip0.setArcHeight(20);
42.     seaClip0.setArcWidth(20);
43.
44.     sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
                                     "images/sea0.jpg")));
45.     sea0.setClip(seaClip0);
46.
47.     setUpDragging();
48.     sea0.xProperty().bind(imageXProperty);
49.
50.     setQuitButton();
51.     root.getChildren().addAll(sea0, quit);
52.     primaryStage.setScene(scene);
53.     primaryStage.show();
54. }
55.
56. private void setUpDragging() {
57.     sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
58.         public void handle(MouseEvent me) {
59.             sX = me.getSceneX() - imageXProperty.getValue();
60.         }
61.     });
62.
63.     sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
64.         public void handle(MouseEvent me) {
65.             imageXProperty.set(me.getSceneX() - sX);
66.         }
67.     });
68. }
69.
70. private void setQuitButton() {
71.     quit = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                     "images/closeIcon.png")));
72.     quit.setFitHeight(25);
73.     quit.setFitWidth(25);
74.     quit.setLayoutX(370);
75.     quit.setLayoutY(5);
76.
77.     quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
78.
79.         public void handle(MouseEvent me) {
80.             System.exit(0);
81.         }
82.     }));
83. }
84. }

```

Now, let's add more UI components so we can play with animations.

- Add a new `ImageView`, called `sea1`. `sea1` is going to be the same as `sea0`, but it will display `sea1.jpg`.
- Create a new `Rectangle`, `seaClip1`, with the same specifications as `seaClip0`.
- Set `seaClip1` as the clip node for `sea1`.
- Bind the `sea1`'s x coordinate to `imageXProperty`

- Create the `setOnMousePressed` and `setOnMouseDragged` methods for `sea1`. The will be the same as the ones for `sea0`.
- Finally, remember to add `sea1` to your scenegraph.
- Your code should look like this:

```

5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.beans.property.DoubleProperty;
9. import javafx.event.EventHandler;
10. import javafx.scene.Group;
11. import javafx.scene.Scene;
12. import javafx.scene.image.Image;
13. import javafx.scene.image.ImageView;
14. import javafx.scene.input.MouseEvent;
15. import javafx.scene.shape.Rectangle;
16. import javafx.stage.Stage;
17. import javafx.stage.StageStyle;
18.
19. /**
20.  *
21.  * @author angie
22.  */
23. public class SeaApp extends Application {
24.     private ImageView sea0;
25.     private Rectangle seaClip0;
26.     private ImageView sea1;
27.     private Rectangle seaClip1;
28.     private ImageView quit;
29.
30.
31.     private double sX = 0;
32.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
33.     /**
34.      * @param args the command line arguments
35.      */
36.     public static void main(String[] args) {
37.         Application.launch(SeaApp.class, args);
38.     }
39.
40.     @Override
41.     public void start(Stage primaryStage) {
42.         primaryStage.initStyle(StageStyle.TRANSPARENT);
43.         Group root = new Group();
44.         Scene scene = new Scene(root, 400, 250, null);
45.
46.         seaClip0 = new Rectangle(400, 220);
47.         seaClip0.setArcHeight(20);
48.         seaClip0.setArcWidth(20);
49.         seaClip1 = new Rectangle(400, 220);
50.         seaClip1.setArcHeight(20);
51.         seaClip1.setArcWidth(20);
52.
53.         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
54.             "images/sea0.jpg")));
55.         sea0.setClip(seaClip0);
56.         sea1 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
57.             "images/sea1.jpg")));
58.         sea1.setClip(seaClip1);
59.
60.         setUpDragging();
61.         sea0.xProperty().bind(imageXProperty);
62.         sea1.xProperty().bind(imageXProperty);
63.
64.         setQuitButton();

```



```

63.     root.getChildren().addAll(sea0, sea1, quit);
64.     primaryStage.setScene(scene);
65.     primaryStage.show();
66. }
67.
68. private void setUpDragging() {
69.     sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
70.         public void handle(MouseEvent me) {
71.             sX = me.getSceneX() - imageXProperty.getValue();
72.         }
73.     });
74.
75.     sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
76.         public void handle(MouseEvent me) {
77.             imageXProperty.set(me.getSceneX() - sX);
78.         }
79.     });
80.     sea1.setOnMousePressed(new EventHandler<MouseEvent>() {
81.         public void handle(MouseEvent me) {
82.             sX = me.getSceneX() - imageXProperty.getValue();
83.         }
84.     });
85.
86.     sea1.setOnMouseDragged(new EventHandler<MouseEvent>() {
87.         public void handle(MouseEvent me) {
88.             imageXProperty.set(me.getSceneX() - sX);
89.         }
90.     });
91. }
92.
93. private void setQuitButton() {
94.     quit = new ImageView(new Image(SeaApp.class.getResourceAsStream
95.                                     ("images/closeIcon.png")));
96.     quit.setFitHeight(25);
97.     quit.setFitWidth(25);
98.     quit.setLayoutX(370);
99.     quit.setLayoutY(5);
100.
101.     quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
102.         public void handle(MouseEvent me) {
103.             System.exit(0);
104.         }
105.     }));
106. }
107. }

```

- Run the application



When you run the application you should see the new added image `sea1`. In our case, what is happening is images are being added one on top of the other, so we only see

the one on top, but `sea0` still under it. This behavior will be different depending on the container you are using. (More about containers later on this lab).

We are going to use some animations to show and hide `sea1`, so we can see `sea0` that is sitting underneath. Initially we are going to show `sea0` by hiding `sea1`, and as the mouse move over it, we are going to show `sea1`, hiding `sea0`. For the visibility of the image, we can use the image's opacity, when it's 1.0 the image is visible, when it's set to 0.0 the image is transparent.

- First, lets set the opacity of `sea1` to 0.0, making this image transparent. Add the following code in line 57th.

```
57.      sea1.setOpacity(0.0);
```

- Implement `setOnMouseEntered` and `setOnMouseExited` for `sea1`. On mouse entered we set `sea1`'s opacity to 1.0, on mouse exit we set `sea1`'s opacity to 0.0.

```
private void setUpVisibility() {
    sea1.setOnMouseEntered(new EventHandler<MouseEvent>() {
        public void handle(MouseEvent me) {
            sea1.setOpacity(1.0);
        }
    });

    sea1.setOnMouseExited(new EventHandler<MouseEvent>() {
        public void handle(MouseEvent me) {
            sea1.setOpacity(0.0);
        }
    });
}
```

- Add `setUpVisibility` method call in line 60th.
- Your code should look like this:

```
5. package seaapp;
6.
7. import javafx.application.Application;
8. import javafx.beans.property.DoubleProperty;
9. import javafx.event.EventHandler;
10. import javafx.scene.Group;
11. import javafx.scene.Scene;
12. import javafx.scene.image.Image;
13. import javafx.scene.image.ImageView;
14. import javafx.scene.input.MouseEvent;
15. import javafx.scene.shape.Rectangle;
16. import javafx.stage.Stage;
17. import javafx.stage.StageStyle;
18.
19. /**
20.  *
21.  * @author angie
22.  */
```

```

23. public class SeaApp extends Application {
24.     private ImageView sea0;
25.     private Rectangle seaClip0;
26.     private ImageView seal;
27.     private Rectangle seaClip1;
28.     private ImageView quit;
29.
30.
31.     private double sX = 0;
32.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
33.     /**
34.      * @param args the command line arguments
35.      */
36.     public static void main(String[] args) {
37.         Application.launch(SeaApp.class, args);
38.     }
39.
40.     @Override
41.     public void start(Stage primaryStage) {
42.         primaryStage.initStyle(StageStyle.TRANSPARENT);
43.         Group root = new Group();
44.         Scene scene = new Scene(root, 400, 250, null);
45.
46.         seaClip0 = new Rectangle(400, 220);
47.         seaClip0.setArcHeight(20);
48.         seaClip0.setArcWidth(20);
49.         seaClip1 = new Rectangle(400, 220);
50.         seaClip1.setArcHeight(20);
51.         seaClip1.setArcWidth(20);
52.
53.         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
54.             "images/sea0.jpg")));
55.         sea0.setClip(seaClip0);
56.         seal = new ImageView(new Image(SeaApp.class.getResourceAsStream(
57.             "images/seal.jpg")));
58.         seal.setClip(seaClip1);
59.         seal.setOpacity(0.0);
60.
61.         setUpDragging();
62.         sea0.xProperty().bind(imageXProperty);
63.         seal.xProperty().bind(imageXProperty);
64.
65.         setQuitButton();
66.         root.getChildren().addAll(sea0, seal, quit);
67.         primaryStage.setScene(scene);
68.         primaryStage.show();
69.     }
70.
71.     private void setUpDragging() {
72.         sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
73.             public void handle(MouseEvent me) {
74.                 sX = me.getSceneX() - imageXProperty.getValue();
75.             }
76.         });
77.
78.         sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
79.             public void handle(MouseEvent me) {
80.                 imageXProperty.set(me.getSceneX() - sX);
81.             }
82.         });
83.
84.         seal.setOnMousePressed(new EventHandler<MouseEvent>() {
85.             public void handle(MouseEvent me) {
86.                 sX = me.getSceneX() - imageXProperty.getValue();
87.             }
88.         });

```

```

88.         seal.setOnMouseDragged(new EventHandler<MouseEvent>() {
89.             public void handle(MouseEvent me) {
90.                 imageXProperty.set(me.getSceneX() - sX);
91.             }
92.         });
93.     }
94.
95.     private void setUpVisibility() {
96.         seal.setOnMouseEntered(new EventHandler<MouseEvent>() {
97.             public void handle(MouseEvent me) {
98.                 seal.setOpacity(1.0);
99.             }
100.        });
101.
102.        seal.setOnMouseExited(new EventHandler<MouseEvent>() {
103.            public void handle(MouseEvent me) {
104.                seal.setOpacity(0.0);
105.            }
106.        });
107.    }
108.
109.    private void setQuitButton() {
110.        quit = new ImageView(new Image(SeaApp.class.getResourceAsStream(
111.                                     "images/closeIcon.png")));
112.        quit.setFitHeight(25);
113.        quit.setFitWidth(25);
114.        quit.setLayoutX(370);
115.        quit.setLayoutY(5);
116.        quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
117.            public void handle(MouseEvent me) {
118.                System.exit(0);
119.            }
120.        }));
121.    }
122. }
123. }

```

- Run the program.

You can notice that we obtained the desired result: as your mouse enter and exit the application's area, you alternate from one image to the other, however it happens abruptly. We want to have a nice and smooth transition between images, so lets use animations instead of direct value assignment.

In JavaFX as part of the animation package we have transitions, and one useful transition is **FadeTransition**. Fade transition creates a fade effect animation that spans its duration. This is done by updating the opacity variable of the node at regular interval. This is exactly what we want to do.

For a **FadeTransition** you normally set the **fromValue**, or node's opacity at the beginning of the transition, and the **toValue**, or node's opacity at the end of the interval once the transition finish.

- First define your `FadeTransition`

```
private FadeTransition fadeTransition;
```

- Fix the imports
- Inside the start method we need to create the `fadeTransition`:
 - The constructor needs two parameters:
 - `duration` → The duration of the `FadeTransition`, in our case 1000ms
 - `node` → The node which opacity will be animated, in our case `sea1`
- Set the `fromValue` and the `toValue` for the animation.
 - `fromValue` → 0.0 (= invisible)
 - `toValue` → 1.0 (= visible)

```
fadeTransition = new FadeTransition(Duration.seconds(1), sea1);  
fadeTransition.setFromValue(0.0f);  
fadeTransition.setToValue(1.0f);
```

- Fix imports as required
- Now, we need to update `mouseEntered` and `mouseExited` handlers.
 - For our application we would like to show and also hide image `sea1`, so we want to be able to run the transition forward and also backwards. The good news is we can use the animation's rate for this. A positive rate means we are playing the transition forward, a negative rate means we are playing the transition backwards.
 - Lets update the `setUpVisibility` method.

- For `setOnMouseEntered` we want to replace the line

```
sea1.setOpacity(1.0);
```

with

```
fadeTransition.setRate(1.0);  
fadeTransition.play();
```

First, we make sure we are running the animation forward, and then we play it.

- For `setOnMouseExited` we want to replace the line

```
sea1.setOpacity(0.0);
```

with

```
fadeTransition.setRate(-1.0);  
fadeTransition.play();
```

In here, we set the animation to play backward, and then we play it.

- Your code should look like this:

```

5. package seaapp;
6.
7. import ...

//code commented out
25. public class SeaApp extends Application {
26.     private ImageView sea0;
27.     private Rectangle seaClip0;
28.     private ImageView sea1;
29.     private Rectangle seaClip1;
30.     private ImageView quit;
31.
32.
33.     private double sX = 0;
34.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
35.
36.     private FadeTransition fadeTransition;
37.     /**
38.      * @param args the command line arguments
39.      */
40.     public static void main(String[] args) {
41.         Application.launch(SeaApp.class, args);
42.     }
43.
44.     @Override
45.     public void start(Stage primaryStage) {
46.         primaryStage.initStyle(StageStyle.TRANSPARENT);
47.         Group root = new Group();
48.         Scene scene = new Scene(root, 400, 250, null);
49.
50.         seaClip0 = new Rectangle(400, 220);
51.         seaClip0.setArcHeight(20);
52.         seaClip0.setArcWidth(20);
53.         seaClip1 = new Rectangle(400, 220);
54.         seaClip1.setArcHeight(20);
55.         seaClip1.setArcWidth(20);
56.
57.         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
58.             "images/sea0.jpg")));
59.         sea1 = new ImageView(new Image(SeaApp.class.getResourceAsStream(
60.             "images/sea1.jpg")));
61.         sea1.setClip(seaClip1);
62.         sea1.setOpacity(0.0);
63.
64.         setUpDragging();
65.         setUpVisibility();
66.
67.         fadeTransition = new FadeTransition(Duration.seconds(1), sea1);
68.         fadeTransition.setFromValue(0.0f);
69.         fadeTransition.setToValue(1.0f);
70.
71.         sea0.xProperty().bind(imageXProperty);
72.         sea1.xProperty().bind(imageXProperty);
73.
74.         setQuitButton();
75.         root.getChildren().addAll(sea0, sea1, quit);
76.         primaryStage.setScene(scene);
77.         primaryStage.show();
78.     }
79.     private void setUpDragging() {...} //code commented out
103.
104.     private void setUpVisibility() {
105.         sea1.setOnMouseEntered(new EventHandler<MouseEvent>() {
106.             public void handle(MouseEvent me) {
107.                 fadeTransition.setRate(1.0);

```

```

108.         fadeTransition.play();
109.     }
110. });
111.
112.     sea1.setOnMouseExited(new EventHandler<MouseEvent>() {
113.         public void handle(MouseEvent me) {
114.             fadeTransition.setRate(-1.0);
115.             fadeTransition.play();
116.         }
117.     });
118. }
119.
120. private void setQuitButton() {...} //code commented out
}

```

- Run your application. You should see a nice and smooth transition between images when the mouse enters and exit the application screen.

Lets add a bit of fun to our application by adding new components, animations and effects to our images.

- First lets add a shark image.

```

31. private ImageView shark;

```

- Create the `ImageView`

```

63. shark = new ImageView(new Image(SeaApp.class.getResourceAsStream(
    "images/shark.png")));

```

- The `shark` is a yellow shark, and it should be displayed whenever the yellow sea (or `sea1` image) get displayed. This is a good opportunity to use binding again. Lets bind the `shark`'s opacity with `sea1`'s opacity

```

64. shark.opacityProperty().bind(sea1.opacityProperty());

```

- Set the scale property of the shark to 0.8.

```

65. shark.setScaleX(0.8);
66. shark.setScaleY(0.8);

```

- Add your shark to your scenegraph. Update line 79.

```

79. root.getChildren().addAll(sea0, sea1, shark, quit);

```

- Your code will look like this

```

5. package seaapp;
6.
7. import ... //code commented out
20.
21. /**...*/ //code commented out
25. public class SeaApp extends Application {
26.     private ImageView sea0;
27.     private Rectangle seaClip0;
28.     private ImageView sea1;
29.     private Rectangle seaClip1;
30.     private ImageView quit;
31.     private ImageView shark;
32.
33.     private double sX = 0;

```

```

34.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
35.
36.     private FadeTransition fadeTransition;
37.     /**...*/      //code commented out
40.     public static void main(String[] args) {...} //code commented out
43.
44.     @Override
45.     public void start(Stage primaryStage) {
46.         primaryStage.initStyle(StageStyle.TRANSPARENT);
47.         Group root = new Group();
48.         Scene scene = new Scene(root, 400, 250, null);
49.
50.         seaClip0 = new Rectangle(400, 220);
51.         seaClip0.setArcHeight(20);
52.         seaClip0.setArcWidth(20);
53.         seaClip1 = new Rectangle(400, 220);
54.         seaClip1.setArcHeight(20);
55.         seaClip1.setArcWidth(20);
56.
57.         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                     ("images/sea0.jpg")));
58.         sea0.setClip(seaClip0);
59.         sea1 = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                     ("images/sea1.jpg")));
60.         sea1.setClip(seaClip1);
61.         sea1.setOpacity(0.0);
62.
63.         shark = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                     ("images/shark.png")));
64.         shark.opacityProperty().bind(sea1.opacityProperty());
65.         shark.setScaleX(0.8);
66.         shark.setScaleY(0.8);
67.
68.         setUpDragging();
69.         setUpVisibility();
70.
71.         fadeTransition = new FadeTransition(Duration.seconds(1), sea1);
72.         fadeTransition.setFromValue(0.0f);
73.         fadeTransition.setToValue(1.0f);
74.
75.         sea0.xProperty().bind(imageXProperty);
76.         sea1.xProperty().bind(imageXProperty);
77.
78.         setQuitButton();
79.         root.getChildren().addAll(sea0, sea1, shark, quit);
80.         primaryStage.setScene(scene);
81.         primaryStage.show();
82.     }
83.
84.     private void setUpDragging() {...} //code commented out
108.
109.     private void setUpVisibility() {...} //code commented out
124.
125.     private void setQuitButton() {...} //code commented out
139. }

```


- Run your application and notice the two sharks when you activate the yellow sea background.



Lets add some cool effects to the sharks.

- We can use the `DropShadow` class to render a shadow for the sharks with the specified color, radius, and offset.
 - First declare the shadow component, and fix imports

```
33.     private DropShadow sharkShadow;
```

- Now create a black shadow, set the x and y offsets to 5.0

```
71.     sharkShadow = new DropShadow();
72.     sharkShadow.setColor(Color.BLACK);
73.     sharkShadow.setOffsetX(5.0);
74.     sharkShadow.setOffsetY(5.0);
```

- Set the shark's effect

```
75.     shark.setEffect(sharkShadow);
```

- Run the app one more time, and see the difference.



Note: At the time this lab was created, there was a bug and the shadow doesn't seems to be coordinated with the shark's fade animation. The bug has already been filled and it's something we are working on.

- Before we continue, lets refactor again the code, so it looks more organized.
 - Select all the shark code within the start method, and select Refactor → Introduce Method...

- Name the new method **setSharks**
- Your code should look like this :

```
package seaapp;

import javafx.scene.effect.DropShadow;
import ... //code commented out

public class SeaApp extends Application {
    private ImageView sea0;
    private Rectangle seaClip0;
    private ImageView sea1;
    private Rectangle seaClip1;
    private ImageView quit;
    private ImageView shark;
    private DropShadow sharkShadow;

    private double sX = 0;
    private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);

    private FadeTransition fadeTransition;

    public static void main(String[] args) {...} //code commented out

    @Override
    public void start(Stage primaryStage) {...} //code commented out

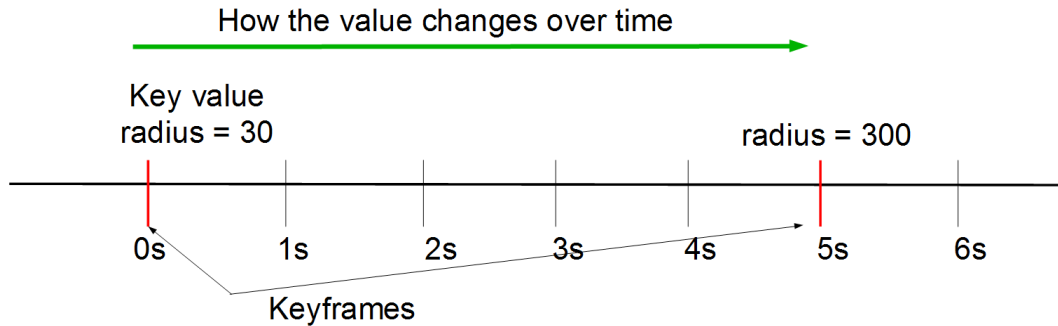
    private void setSharks() {
        shark = new ImageView(new Image(SeaApp.class.getResourceAsStream(
            "images/shark.png")));
        shark.opacityProperty().bind(sea1.opacityProperty());
        shark.setScaleX(0.8);
        shark.setScaleY(0.8);

        sharkShadow = new DropShadow();
        sharkShadow.setColor(Color.BLACK);
        sharkShadow.setOffsetX(5.0);
        sharkShadow.setOffsetY(5.0);

        shark.setEffect(sharkShadow);
    }

    private void setUpDragging() {...} //code commented out
    private void setUpVisibility() {...} //code commented out
    private void setQuitButton() {...} //code commented out
}
```

Finally, lets animate the sharks, so they can swim away. For this animation we are going to use Timelines. **Timeline** provides the capability to update the property values along the progression of time using a set of **KeyFrames**.



Timeline specifies how your application behaves over time, for example, we want to animate a circle, by modifying the radius property. A **KeyFrame** defines target values at a specified point in time, for example at time=0, the radius value (called **KeyValue**) is 30, and at time = 5s, **KeyValue** will be 300. By default a linear interpolation will be performed, unless otherwise it's specified.

```
final Circle circle = new Circle(100, 100, 30);

final Timeline timeline = new Timeline(
    new KeyFrame(Duration.ZERO,
        new KeyValue(circle.radiusProperty(), 30)),
    new KeyFrame(new Duration(5000),
        new KeyValue(circle.radiusProperty(), 300)));
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
timeline.play();
```

The **Timeline's** **cycleCount** will specify how many times the animation will be performed and the **autoReverse**, allows you to play the animation forward and backward. When you run the previous code, you will have a circle that grows till it's radius is 300 and they shrink till it's radius is 30. Each cycle will take 5 seconds (or as specified 5000 milliseconds)

Now that we have better understanding of **Timeline**, **KeyFrame** and **KeyValue**, we can add some animation for the sharks. We want the sharks to swim away: they need to move along the x and y coordinates, and they need to be zoomed in to give the impression of proximity. Another trick we can use, is to move the shadow a bit slower than the sharks, this will give us an impression that the sharks are getting closer to us, as their shadow is left behind.

- First we need to create some properties:

```

41. private DoubleProperty xOff = new SimpleDoubleProperty(0.0);
42. private DoubleProperty yOff = new SimpleDoubleProperty(0.0);
43. private DoubleProperty sOff = new SimpleDoubleProperty(5.0);
44. private DoubleProperty scale = new SimpleDoubleProperty(0.8);

```

`xOff` and `yOff` are two properties we are going to use for moving the sharks along x and y coordinates. They will be bound to shark's `xProperty` and `yProperty`.

`sOff` will be bound to the shadow's `xProperty` and `yProperty`.

`scale` will allow us to zoom in the sharks, and it will be bound to `scaleXProperty` and `scaleYProperty`.

- Lets define the binding for the shark and shadow's properties. Make sure you update the `setSharks` methods correctly to match the following code.

```

88.     private void setSharks() {
89.         shark = new ImageView(new Image(SeaApp.class.
                                   getResourceAsStream("images/shark.png")));
90.         shark.opacityProperty().bind(seal.opacityProperty());
91.         shark.scaleXProperty().bind(scale);
92.         shark.scaleYProperty().bind(scale);
93.         shark.xProperty().bind(xOff);
94.         shark.yProperty().bind(yOff);
95.
96.         sharkShadow = new DropShadow();
97.         sharkShadow.setColor(Color.BLACK);
98.         sharkShadow.offsetXProperty().bind(sOff);
99.         sharkShadow.offsetYProperty().bind(sOff);
100.
101.         shark.setEffect(sharkShadow);
102.     }

```

- Declare the **Timeline**

```

47.     private Timeline sharkSwimAway;

```

- Create **sharkSwimAway** Timeline

```

109. sharkSwimAway = new Timeline(
110.     new KeyFrame(Duration.ZERO,
111.         new KeyValue(sOff, 0),
112.         new KeyValue(scale, 0.8),
113.         new KeyValue(xOff, 0),
114.         new KeyValue(yOff, 0)),
115.     new KeyFrame(new Duration(5000),
116.         new KeyValue(sOff, -100),
117.         new KeyValue(scale, 4.0),
118.         new KeyValue(xOff, 2500),
119.         new KeyValue(yOff, 800))
120. );

```

- We are going to trigger the animation when the user click on the sharks

```
122. shark.setOnMouseClicked(new EventHandler<MouseEvent>() {
123.     public void handle(MouseEvent me) {
124.         sharkSwimAway.play();
125.     }
126. });
```

- Update the scene size, so the sharks will have enough space to swim

```
62. Scene scene = new Scene(root, 1800, 1100);
```

- Before we can run the code, there is one final thing we need to fix. As the sharks are placed on top of `sea0` and `sea1`, when the mouse goes over them, it will trigger the `mouseExited` on `sea1`, changing the background, but we haven't really gone out of the application area. This can be easily fixed by adding a line to check if we really got out of the screen. Make sure you update the `setUpVisibility` method as follows.

```
155. private void setUpVisibility() {
156.     sea1.setOnMouseEntered(new EventHandler<MouseEvent>() {
157.         public void handle(MouseEvent me) {
158.             fadeTransition.setRate(1.0);
159.             fadeTransition.play();
160.         }
161.     });
162.
163.     sea1.setOnMouseExited(new EventHandler<MouseEvent>() {
164.         public void handle(MouseEvent me) {
165.             if (!seaClip1.contains(new Point2D(me.getSceneX(),
166.                                                     me.getSceneY()))){
167.                 fadeTransition.setRate(-1.0);
168.                 fadeTransition.play();
169.             }
170.         }
171.     });
172. }
```

Your final code should look like this:

```
5. package seaapp;
6.
7. import javafx.animation.FadeTransition;
8. import javafx.animation.KeyFrame;
9. import javafx.animation.KeyValue;
10. import javafx.animation.Timeline;
11. import javafx.application.Application;
12. import javafx.beans.property.DoubleProperty;
13. import javafx.event.EventHandler;
14. import javafx.geometry.Point2D;
```

```

15. import javafx.scene.Group;
16. import javafx.scene.Scene;
17. import javafx.scene.effect.DropShadow;
18. import javafx.scene.image.Image;
19. import javafx.scene.image.ImageView;
20. import javafx.scene.input.MouseEvent;
21. import javafx.scene.paint.Color;
22. import javafx.scene.shape.Rectangle;
23. import javafx.stage.Stage;
24. import javafx.stage.StageStyle;
25. import javafx.util.Duration;
26.
27. /**
28.  *
29.  * @author angie
30.  */
31. public class SeaApp extends Application {
32.     private ImageView sea0;
33.     private Rectangle seaClip0;
34.     private ImageView sea1;
35.     private Rectangle seaClip1;
36.     private ImageView quit;
37.     private ImageView shark;
38.     private DropShadow sharkShadow;
39.
40.     private double sX = 0;
41.     private DoubleProperty imageXProperty = new SimpleDoubleProperty(0);
42.
43.     private FadeTransition fadeTransition;
44.
45.     private DoubleProperty xOff = new SimpleDoubleProperty(0.0);
46.     private DoubleProperty yOff = new SimpleDoubleProperty(0.0);
47.     private DoubleProperty sOff = new SimpleDoubleProperty(5.0);
48.     private DoubleProperty scale = new SimpleDoubleProperty(0.8);
49.
50.     private Timeline sharkSwimAway;
51.
52.     /**
53.      * @param args the command line arguments
54.      */
55.     public static void main(String[] args) {
56.         Application.launch(SeaApp.class, args);
57.     }
58.
59.     @Override
60.     public void start(Stage primaryStage) {
61.         primaryStage.initStyle(StageStyle.TRANSPARENT);
62.         Group root = new Group();
63.         Scene scene = new Scene(root, 1800, 1100, null);
64.
65.         seaClip0 = new Rectangle(400, 220);
66.         seaClip0.setArcHeight(20);
67.         seaClip0.setArcWidth(20);
68.         seaClip1 = new Rectangle(400, 220);
69.         seaClip1.setArcHeight(20);
70.         seaClip1.setArcWidth(20);
71.
72.         sea0 = new ImageView(new Image(SeaApp.class.getResourceAsStream
            ("images/sea0.jpg")));

```

```

73.         sea0.setClip(seaClip0);
74.         seal = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                   ("images/seal.jpg")));
75.         seal.setClip(seaClip1);
76.         seal.setOpacity(0.0);
77.         setSharks();
78.
79.         setUpDragging();
80.         setUpVisibility();
81.
82.         fadeTransition = new FadeTransition(Duration.seconds(1), seal);
83.         fadeTransition.setFromValue(0.0f);
84.         fadeTransition.setToValue(1.0f);
85.
86.         sea0.xProperty().bind(imageXProperty);
87.         seal.xProperty().bind(imageXProperty);
88.
89.         setQuitButton();
90.         root.getChildren().addAll(sea0, seal, shark, quit);
91.         primaryStage.setScene(scene);
92.         primaryStage.show();
93.     }
94.
95.     private void setSharks() {
96.         shark = new ImageView(new Image(SeaApp.class.getResourceAsStream
                                           ("images/shark.png")));
97.         shark.opacityProperty().bind(seal.opacityProperty());
98.         shark.scaleXProperty().bind(scale);
99.         shark.scaleYProperty().bind(scale);
100.        shark.xProperty().bind(xOff);
101.        shark.yProperty().bind(yOff);
102.
103.        sharkShadow = new DropShadow();
104.        sharkShadow.setColor(Color.BLACK);
105.        sharkShadow.offsetXProperty().bind(sOff);
106.        sharkShadow.offsetYProperty().bind(sOff);
107.
108.        shark.setEffect(sharkShadow);
109.
110.        sharkSwimAway = new Timeline(
111.            new KeyFrame(Duration.ZERO,
112.                new KeyValue(sOff, 0),
113.                new KeyValue(scale, 0.8),
114.                new KeyValue(xOff, 0),
115.                new KeyValue(yOff, 0)),
116.            new KeyFrame(new Duration(5000),
117.                new KeyValue(sOff, -100),
118.                new KeyValue(scale, 4.0),
119.                new KeyValue(xOff, 2500),
120.                new KeyValue(yOff, 800))
121.        );
122.
123.        shark.setOnMouseClicked(new EventHandler<MouseEvent>() {
124.            public void handle(MouseEvent me) {
125.                sharkSwimAway.play();
126.            }
127.        });
128.    }
129.

```

```

130.     private void setUpDragging() {
131.         sea0.setOnMousePressed(new EventHandler<MouseEvent>() {
132.             public void handle(MouseEvent me) {
133.                 sX = me.getSceneX() - imageXProperty.getValue();
134.             }
135.         });
136.
137.         sea0.setOnMouseDragged(new EventHandler<MouseEvent>() {
138.             public void handle(MouseEvent me) {
139.                 imageXProperty.set(me.getSceneX() - sX);
140.             }
141.         });
142.         sea1.setOnMousePressed(new EventHandler<MouseEvent>() {
143.             public void handle(MouseEvent me) {
144.                 sX = me.getSceneX() - imageXProperty.getValue();
145.             }
146.         });
147.
148.         sea1.setOnMouseDragged(new EventHandler<MouseEvent>() {
149.             public void handle(MouseEvent me) {
150.                 imageXProperty.set(me.getSceneX() - sX);
151.             }
152.         });
153.     }
154.
155.     private void setUpVisibility() {
156.         sea1.setOnMouseEntered(new EventHandler<MouseEvent>() {
157.             public void handle(MouseEvent me) {
158.                 fadeTransition.setRate(1.0);
159.                 fadeTransition.play();
160.             }
161.         });
162.
163.         sea1.setOnMouseExited(new EventHandler<MouseEvent>() {
164.             public void handle(MouseEvent me) {
165.                 if (!seaClip1.contains(new Point2D(me.getSceneX(), me.getSceneY()))) {
166.                     fadeTransition.setRate(-1.0);
167.                     fadeTransition.play();
168.                 }
169.             }
170.         });
171.     }
172.
173.     private void setQuitButton() {
174.         quit = new ImageView(new Image(SeaApp.class.getResourceAsStream(
175.             "images/closeIcon.png")));
176.         quit.setFitHeight(25);
177.         quit.setFitWidth(25);
178.         quit.setLayoutX(370);
179.         quit.setLayoutY(5);
180.
181.         quit.setOnMouseClicked((new EventHandler<MouseEvent>() {
182.             public void handle(MouseEvent me) {
183.                 System.exit(0);
184.             }
185.         }));
186.     }
187. }

```


- Run the application and verify that everything is working as expected.

Exercise 3: Understanding Layouts (15 mins)

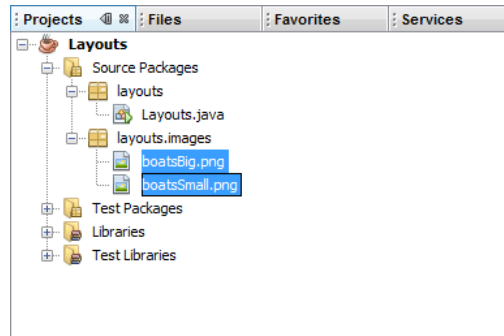
The JavaFX SDK provides several layout container classes, called panes, for the easy setup and management of classic layouts such as rows, columns, stacks, tiles, and others. As a window is resized, the layout pane automatically repositions and resizes the nodes that it contains according to the properties for the nodes. In this exercise we are going to show you how this layouts works.

Before we start playing with the different layouts, lets bring some resources.

- In the previous exercise you learned how to add a directory to your favorites window, lets add exercise3 directory from our exercises directory. (For more detailed instructions look into the previous exercise)
 - Click on Favorite tab
 - Right click on the background and select Add to Favorites
 - Select the directory where you unzip the lab file, and go to exercises
 - Select exercise3 directory and click on Add button

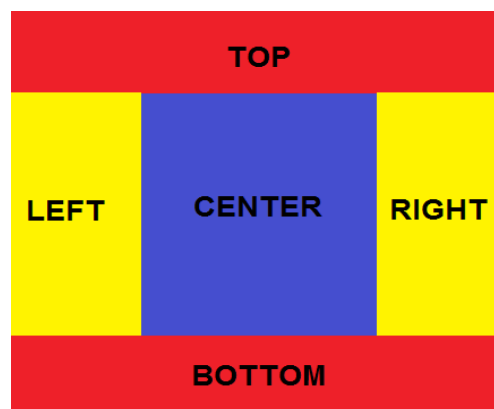
Border Layout

- Create a new project, called it Layouts and place it anywhere you want.
- Click on the Favorites tab
- Expand exercise3 directory
- Select the two files: boatsBig.png and boatsSmall.png
- Right click and select copy from the pop up menu
- Click on the Project Tab
- Expand Layouts project:
 - Expand Source Packages → Layouts
- Right click on layouts package and select New → Java Package... from the pop up menu
- The New Java Package window will be displayed
 - Enter layouts.images for the package name
 - Click on Finish
- Right click on the newly created package and select paste
- Your project should look like the following image



- Resize the scene to be 540x540
- Remove all the lines referring to `Button btn` (from line 34 – 43)
- Comment out the line `root.getChildren().add(btn);`

Now let's start playing with `BorderPane` Layout. The `BorderPane` layout pane provides five regions in which to place nodes: top, bottom, left, right, and center.



- Add the following code in line 32

```
BorderPane border = new BorderPane();
```
- Fix imports
- Uncomment the line with `root.getChildren().add(btn);`, and replace it with

```
root.getChildren().add(border);
```
- Now that you create your container we can start adding components to it
- Add the following code, right after the new border Pane

```
border.setTop(new Rectangle(500,100, Color.RED));
border.setBottom(new Rectangle(500,100, Color.RED));
border.setCenter(new ImageView(new Image(Layouts.class.
    getResourceAsStream("images/boatsSmall.png"))));
border.setLeft(new Rectangle(100,300, Color.BLUE));
border.setRight(new Rectangle(100,300, Color.BLUE));
```

Using `BorderPane` is extremely easy, you just add the components to the area you want, using `setTop`, `setBottom`, `setCenter`, `setLeft` and `setRight` methods.

- Right click on the background and select `Fix Imports`
 - Make sure the `Image` and `Rectangle` that you select is from the `javafx` libraries
- Your code should look like this

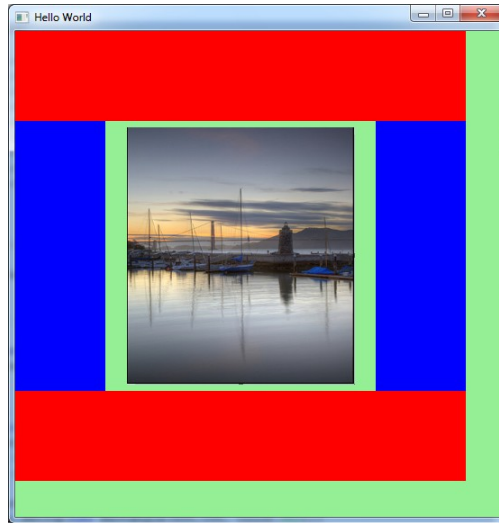
```
5. package layouts;
6.
7. import javafx.application.Application;
8. import javafx.scene.Group;
9. import javafx.scene.Scene;
10. import javafx.scene.image.Image;
11. import javafx.scene.image.ImageView;
12. import javafx.scene.layout.BorderPane;
13. import javafx.scene.paint.Color;
14. import javafx.scene.shape.Rectangle;
15. import javafx.stage.Stage;
16.
17. /**
18.  *
19.  * @author angie
20.  */
21. public class Layouts extends Application {
22.
23.     /**
24.      * @param args the command line arguments
25.      */
26.     public static void main(String[] args) {
27.         Application.launch(Layouts.class, args);
28.     }
29.
30.     @Override
31.     public void start(Stage primaryStage) {
32.         primaryStage.setTitle("Border Pane");
33.         Group root = new Group();
34.         Scene scene = new Scene(root, 540, 540, Color.LIGHTGREEN);
35.
36.         BorderPane border = new BorderPane();
37.         border.setTop(new Rectangle(500,100, Color.RED));
38.         border.setBottom(new Rectangle(500,100, Color.RED));
39.         border.setCenter(new ImageView(new Image(Layouts.class.
```

```

40.         getResourceAsStream("images/boatsSmall.png"))));
41.         border.setLeft(new Rectangle(100,300, Color.BLUE));
42.         border.setRight(new Rectangle(100,300, Color.BLUE));
43.         root.getChildren().add(border);
44.         primaryStage.setScene(scene);
45.         primaryStage.show();
46.     }
47. }

```

- Run your project

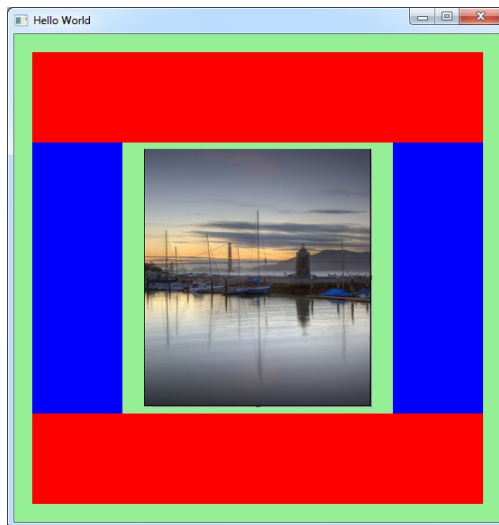


- You can also add some padding to your panel. Add the following line and run your project again to see the difference

```

42. border.setPadding(new Insets(20,20,20,20));

```

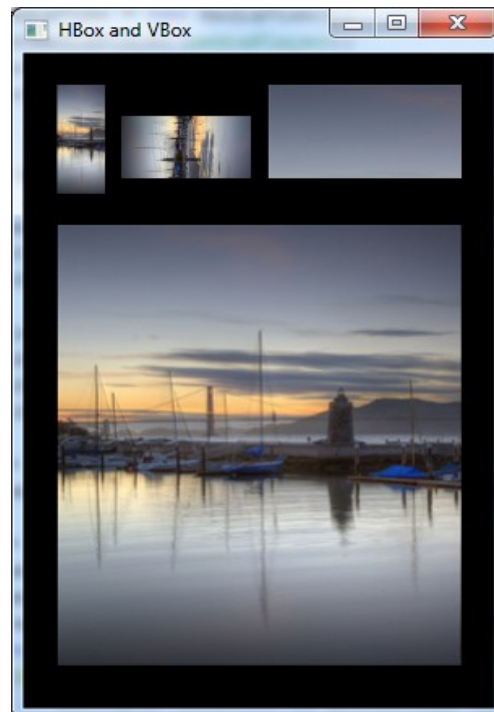


VBox and HBox Pane

The **HBox** layout pane provides an easy way for arranging a series of nodes in a single row. The **VBox** layout pane is similar to the **HBox** layout pane, except that the nodes are arranged in a single column.

Padding attributes can be set to manage the distance between the nodes and the edges of these panes. Spacing attributes can be set to manage the distance between the nodes. The style can be set to change the background color.

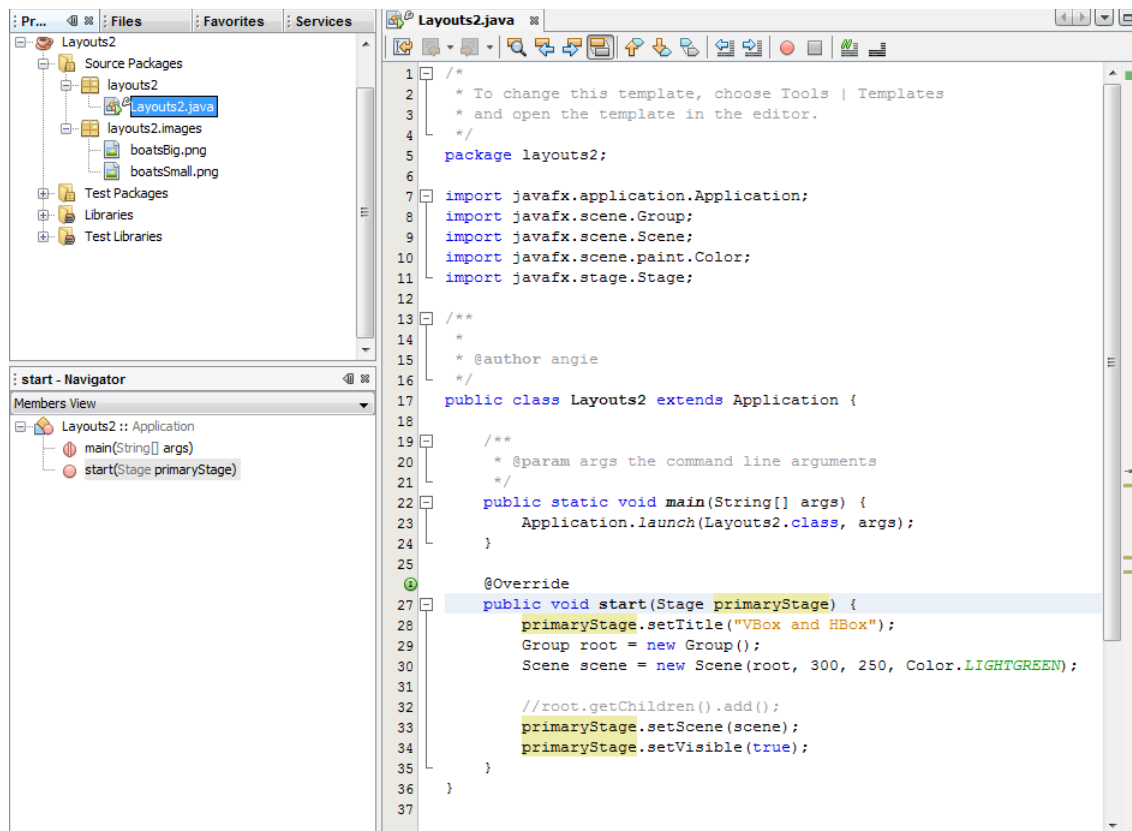
To do more complex UI, you normally nest different pane. Lets try to build the following application:



You can notice that it doesn't fit completely on a **VBox**, nor on a **HBox**, but a combination of these panes will be perfect. The upper part will fit in a **HBox**, having 3 elements. Then the **HBox** with the bigger image at the bottom will perfectly fit inside a **VBox**.



- Open Layouts2 project:
 - Click on the open project icon or use the menu File → Open Project... option.
 - Navigate to the location where you unzipped the lab file
 - Extend exercises → exercise3
 - Open Layouts2 project
- The Layouts2 project already has the basic structure for you put your JavaFX content, and it has an `images` directory with the required resources.



- Make you scene 290 x 420 and with a black background.
- Create an ImageView with the following characteristics
 - Name it smallImg
 - Display images/boatsSmall.png image
 - Size 30x70
 - Set preserveRatio to false
 - Set smooth to true
 - Use the Image constructor to set these values

```

ImageView smallImg = new ImageView(
    new Image(Layouts2.class.getResourceAsStream(
        "images/boatsSmall.png"), 30, 70, false, true));

```

- Create a second ImageView:
 - Name it smallRotatedImg
 - Display images/boatsSmall.png image
 - Set the width to 40
 - Set the height to 80

- Set `preserveRatio` to `false`
- Set `smooth` to `true`
- Rotate the image 90 degree
- Use the `ImageView`'s methods to set those values

```
ImageView smallRotatedImg = new ImageView(
    new Image(Layouts2.class.getResourceAsStream(
        "images/boatsSmall.png")));
smallRotatedImg.setFitWidth(40);
smallRotatedImg.setFitHeight(80);
smallRotatedImg.setPreserveRatio(false);
smallRotatedImg.setSmooth(true); //better filter
smallRotatedImg.setRotate(90);
```

- Create a third `ImageView`
 - Name it `viewPortImg`
 - Set the image's height to 60
 - Set `preserve ratio` to `true`

```
ImageView viewPortImg = new ImageView(
    new Image(Layouts2.class.getResourceAsStream(
        "images/boatsSmall.png")));
viewPortImg.setFitHeight(60);
viewPortImg.setPreserveRatio(true);
```

- You can set the view port for your `ImageView`.
 - Create a `Rectangle2D` for the view port
 - Set `Rectangle2D`'s `x` and `y` to 0, width to 120 and height to 60
- Set the `ImageView`'s view port

```
Rectangle2D rectangle2D = new Rectangle2D(0, 0, 120, 60);
viewPortImg.setViewport(rectangle2D); //image's viewport
```



- Now, create a `HBox` to place the `ImageViews` we just created.
 - Create an `HBox`

```
HBox hBox = new HBox();
```

- Set the spacing between components in the `HBox` to 30

```
hBox.setSpacing(30);
```

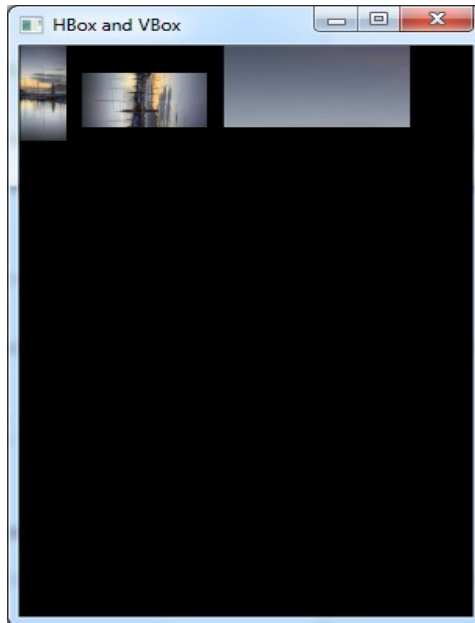
- Add `smallImg`, `smallRotatedImg` and `viewPortImg` to the `HBox`.

```
hBox.getChildren().addAll(smallImg, smallRotatedImg, viewPortImg);
```

- Add your `HBox` to the root

```
root.getChildren().add(hBox);
```

- Fix imports and make sure you always select the classes from the JavaFX libraries
- Run your project.



- Create a 4th `ImageView`
 - Name it `bigImage`
 - Set its width to 250
 - Set the `preserveRatio` to `true`

```
ImageView bigImg = new ImageView(
    new Image(Layouts2.class.getResourceAsStream(
        "images/boatsSmall.png")));
bigImg.setFitWidth(250); //preferred width
bigImg.setPreserveRatio(true);
```

- Create a `vbox` and add the `HBox` and `bigImg` to it.

```
VBox vb = new VBox(10);
vb.getChildren().addAll(hBox, bigImg);
```

- Update the root's childrens to be `vb` instead of `hBox`

```
root.getChildren().add(vb);
```

- Fix imports
- Run project.



- Now, we are only missing the space around the `vbox`.
 - Add the pane's padding to be
 - Top = 20
 - Right = 20
 - Bottom = 20
 - Left = 20

```
vb.setPadding(new Insets(20,20,20,20));
```

- Fix imports if required
- Your code should look like this:

```
5. package layouts2;  
6.  
7. import javafx.application.Application;  
8. import javafx.geometry.Insets;  
9. import javafx.geometry.Rectangle2D;  
10. import javafx.scene.Group;  
11. import javafx.scene.Scene;  
12. import javafx.scene.image.Image;  
13. import javafx.scene.image.ImageView;
```

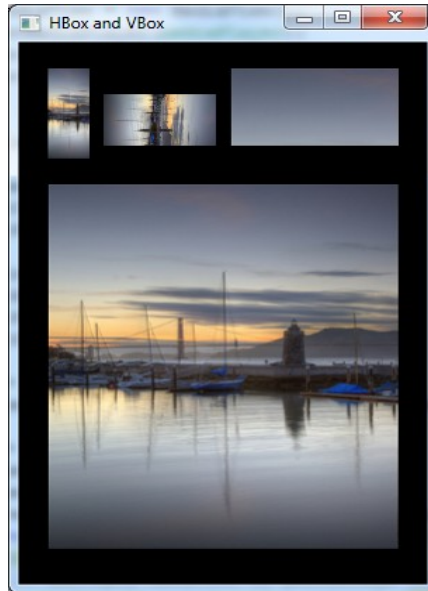
```

14. import javafx.scene.layout.HBox;
15. import javafx.scene.layout.VBox;
16. import javafx.scene.paint.Color;
17. import javafx.stage.Stage;
18.
19. /**
20.  *
21.  * @author angie
22.  */
23. public class Layouts2 extends Application {
24.
25.     /**
26.      * @param args the command line arguments
27.      */
28.     public static void main(String[] args) {
29.         Application.launch(Layouts2.class, args);
30.     }
31.
32.     @Override
33.     public void start(Stage primaryStage) {
34.         primaryStage.setTitle("HBox and VBox");
35.         Group root = new Group();
36.         Scene scene = new Scene(root, 290, 420, Color.BLACK);
37.         ImageView smallImg = new ImageView(
38.             new Image(Layouts2.class.getResourceAsStream(
39.                 "images/boatsSmall.png"), 30, 70, false, true));
40.         ImageView smallRotatedImg = new ImageView(
41.             new Image(Layouts2.class.getResourceAsStream(
42.                 "images/boatsSmall.png")));
43.         smallRotatedImg.setFitWidth(40);
44.         smallRotatedImg.setFitHeight(80);
45.         smallRotatedImg.setPreserveRatio(false);
46.         smallRotatedImg.setSmooth(true); //better filter
47.         smallRotatedImg.setRotate(90);
48.         ImageView viewPortImg = new ImageView(
49.             new Image(Layouts2.class.getResourceAsStream(
50.                 "images/boatsSmall.png")));
51.         viewPortImg.setFitHeight(60);
52.         viewPortImg.setPreserveRatio(true);
53.         Rectangle2D rectangle2D = new Rectangle2D(0, 0, 120, 60);
54.         viewPortImg.setViewport(rectangle2D); //image's viewport
55.         HBox hBox = new HBox();
56.         hBox.setSpacing(30);
57.         hBox.getChildren().addAll(smallImg, smallRotatedImg, viewPortImg);
58.         ImageView bigImg = new ImageView(
59.             new Image(Layouts2.class.getResourceAsStream(
60.                 "images/boatsSmall.png")));
61.         bigImg.setFitWidth(250); //preferred width
62.         bigImg.setPreserveRatio(true);
63.
64.         VBox vb = new VBox(10);
65.         vb.getChildren().addAll(hBox, bigImg);

```

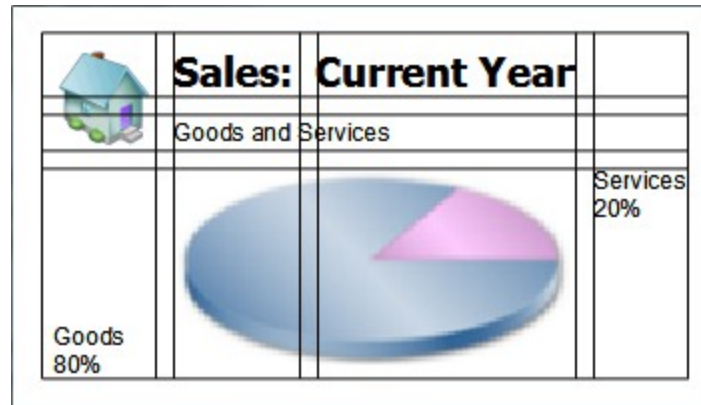
```
58.         vb.setPadding(new Insets(20,20,20,20));
59.         root.getChildren().add(vb);
60.         primaryStage.setScene(scene);
61.         primaryStage.show();
62.     }
63. }
```

- Run your project and make sure it looks like the following snapshot.



Grid Pane

The GridPane layout pane enables you to create a flexible grid of rows and columns in which to lay out nodes. Nodes can be placed in any cell in the grid and can span cells as needed. A grid pane is useful for creating forms or any layout that is organized in rows and columns. The following image shows a grid pane that contains an icon, title, subtitle, text and a pie chart. Grid lines show the rows, columns, and gaps between the rows and columns.



Gap properties can be set to manage the spacing between the rows and columns. The padding property can be set to manage the distance between the nodes and the edges of the grid pane.

- Open Layouts3 project:
 - Click on the open project icon or use the menu File → Open Project... option.
 - Navigate to the location where you unzipped the lab file
 - Extend exercises → exercise3
 - Open Layouts3 project
- The Layouts3 project already has the basic structure for you put your JavaFX content, and it has an images directory with the required resources.
- Open Layouts3.java file by double clicking to it
- Set the scene to be 350x200 with no filling color.

```
Scene scene = new Scene(root, 350, 200);
```

- Create a GridPane named grid

```
GridPane grid = new GridPane();
```

- As you might already notice, the columns and the rows have a little gap between them, set this gap to be 10

```
grid.setHgap(10);
grid.setVgap(10);
```

- There is also a padding around the grid, set it to be 20x20x20x20

```
grid.setPadding(new Insets(20, 20, 20, 20));
```

Now that we have the grid lets start creating the UI components and placing them in the appropriate position in the grid. Take into consideration that the grid numbers start from 0,0 for the component on first row, first column

- Create and place the Sales text
 - Create a `Text` object
 - Name: category
 - Initial value: "Sales:"
 - Set the font to be Bold Tahoma with size 20
 - Add it to the grid into: first row, second column

```
Text category = new Text("Sales:");
category.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
grid.add(category, 1, 0);
```

- Create and place the text "Current Year"
 - Create a `Text` object:
 - Name: chartTitle
 - Initial value: "Current Year"
 - Set the font to be Bold Tahoma with size 20
 - Add it to the grid into: first row, third column

```
Text chartTitle = new Text("Current Year");
chartTitle.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
grid.add(chartTitle, 2, 0);
```

- Create and place the text "Goods and Services"
 - Create a `Text` object
 - Name: chartSubtitle
 - Initial value: "Goods and Services"
 - Place the text in the grid
 - Row: Second
 - Column: In this case the text will expand 2 columns (column 2 and 3)

```
Text chartSubtitle = new Text("Goods and Services");
grid.add(chartSubtitle, 1, 1, 2, 1);
```

- Create and place the house image
 - Create an `ImageView`
 - Name: imageHouse
 - File: images/house.png
 - Size: 50x50
 - PreserveRatio: true
 - Smooth: true

- Add it to the grid:
 - First and second row
 - First column

```
ImageView imageHouse = new ImageView(
    new Image(Layouts3.class.getResourceAsStream(
        "images/house.png"), 50,50,true, true));
grid.add(imageHouse, 0, 0, 1, 2);
```

- Create and place the text “Goods 80%”
 - Create the `Text` object
 - Name: goodsPercent
 - Text value “Goods\n80%”
 - From the master image, we can notice that the Text is placed at the bottom of the grid cell, set the vertical alignment to be BOTTOM
 - Add the text to the third row, first column

```
Text goodsPercent = new Text("Goods\n80%");
GridPane.setValignment(goodsPercent, VPos.BOTTOM);
grid.add(goodsPercent, 0, 2);
```

- Create and place the pie chart image
 - Create an `ImageView`
 - Name: imageChart
 - File: images/pieChart.png
 - Size: 200x200
 - PreserveRatio: true
 - Smooth: true
 - Add it to the grid:
 - Third row
 - Second and third column

```
ImageView imageChart = new ImageView(
    new Image(Layouts3.class.getResourceAsStream(
        "images/pieChart.png"), 200,200,true, true));
grid.add(imageChart, 1, 2, 2, 1);
```

- Create and place the text “Services 20%”
 - Create the `Text` object
 - Name: servicesPercent
 - Text value “Services\n20%”
 - From the master image, we can notice that the Text is placed at the top of the grid cell, set the vertical alignment to be TOP
 - Add the text to the third row, fourth column

```
Text servicesPercent = new Text("Services\n20%");
GridPane.setValignment(servicesPercent, VPos.TOP);
```



```
grid.add(servicesPercent, 3, 2);
```

- Add the grid to the root

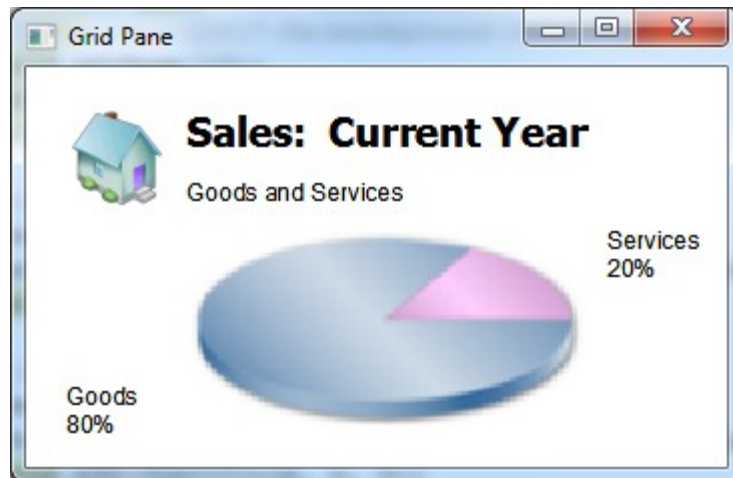
```
root.getChildren().add(grid);
```

```
5. package layouts3;
6.
7. import javafx.application.Application;
8. import javafx.geometry.Insets;
9. import javafx.geometry.VPos;
10. import javafx.scene.Group;
11. import javafx.scene.Scene;
12. import javafx.scene.image.Image;
13. import javafx.scene.image.ImageView;
14. import javafx.scene.layout.GridPane;
15. import javafx.scene.paint.Color;
16. import javafx.scene.text.Font;
17. import javafx.scene.text.FontWeight;
18. import javafx.scene.text.Text;
19. import javafx.stage.Stage;
20.
21. /**
22.  *
23.  * @author angie
24.  */
25. public class Layouts3 extends Application {
26.
27.     /**
28.      * @param args the command line arguments
29.      */
30.     public static void main(String[] args) {
31.         Application.launch(Layouts3.class, args);
32.     }
33.
34.     @Override
35.     public void start(Stage primaryStage) {
36.         primaryStage.setTitle("Grid Pane");
37.         Group root = new Group();
38.         Scene scene = new Scene(root, 350, 200);
39.
40.         GridPane grid = new GridPane();
41.         grid.setHgap(10);
42.         grid.setVgap(10);
43.         grid.setPadding(new Insets(20, 20, 20, 20));
44.
45.         // Category in column 2, row 1
46.         Text category = new Text("Sales:");
47.         category.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
48.         grid.add(category, 1, 0);
49.
50.         // Title in column 3, row 1
```

```

51.         Text chartTitle = new Text("Current Year");
52.         chartTitle.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
53.         grid.add(chartTitle, 2, 0);
54.
55. // Subtitle in columns 2-3, row 2
56.         Text chartSubtitle = new Text("Goods and Services");
57.         grid.add(chartSubtitle, 1, 1, 2, 1);
58.
59. // House icon in column 1, rows 1-2
60.         ImageView imageHouse = new ImageView(
                                new Image(Layouts3.class.getResourceAsStream
                                ("images/house.png"), 50,50,true, true));
61.         grid.add(imageHouse, 0, 0, 1, 2);
62.
63. // Left label in column 1 (bottom), row 3
64.         Text goodsPercent = new Text("Goods\n80%");
65.         GridPane.setValignment(goodsPercent, VPos.BOTTOM);
66.         grid.add(goodsPercent, 0, 2);
67.
68. // Chart in columns 2-3, row 3
69.         ImageView imageChart = new ImageView(
                                new Image(Layouts3.class.getResourceAsStream
                                ("images/pieChart.png"), 200,200,true, true));
70.         grid.add(imageChart, 1, 2, 2, 1);
71.
72. // Right label in column 4 (top), row 3
73.         Text servicesPercent = new Text("Services\n20%");
74.         GridPane.setValignment(servicesPercent, VPos.TOP);
75.         grid.add(servicesPercent, 3, 2);
76.
77.         root.getChildren().add(grid);
78.         primaryStage.setScene(scene);
79.         primaryStage.show();
80.     }
81. }

```



Exercise 4: Embedded Browser (10 mins)

JavaFX SDK introduces the embedded browser, a user interface component that provides a web viewer and full browsing functionality through its API. The embedded browser component is based on [WebKit](#), an open source web browser engine. It supports HTML5, Cascading Style Sheets (CSS), JavaScript, and the Document Object Model (DOM).

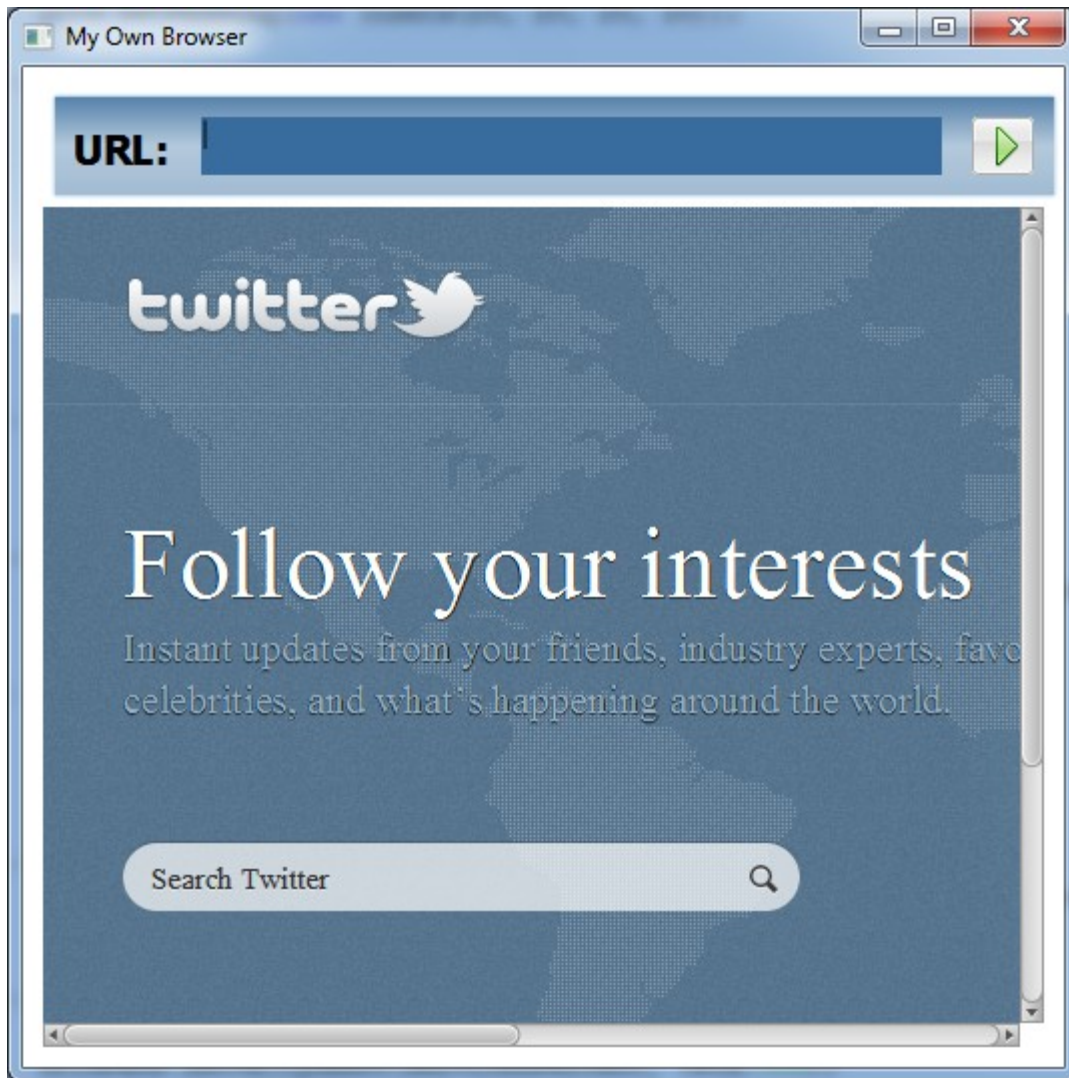
WebEngine Class

The **WebEngine** class provides basic web page functionality. It supports user interaction such as navigating links and submitting HTML forms. The **WebEngine** class handles one web page at a time. The basic browsing features of loading HTML content and accessing the DOM are available in a built-in web engine.

WebView Class

The **WebView** class is an extension of the **Node** class. It encapsulates a **WebEngine** object, incorporates HTML content into an application's scene, and provides fields and methods to apply effects and transformations. To create a **WebView** object, you create the **WebView** object and call the **setEngine** method.

In this exercise we are going to create a browser like the one we show in the following image



Steps

- Open Browser project:
 - Click on the open project icon or use the menu File → Open Project... option.
 - Navigate to the location where you unzipped the lab file
 - Extend exercises → exercise4
 - Open Browser project
- The Browser project already has the basic structure for you put your JavaFX content, and it has an images directory with the required resources.
- Open Browser.java file by double clicking to it
- Set the scene to be 520x500 with no filling color.

- Create an **HBox** for the top bar
 - Set some spacing between components
 - Set some padding
 - Set the components alignment to be **BOTTOM_CENTER**
- Add the following nodes to the **HBox**
 - “URL” text
 - **TextArea** for the URL (it could also be a **TextField**)
 - **ImageView** for the search icon

```

HBox barContents = new HBox();
barContents.setSpacing(15);
barContents.setPadding(new Insets(15, 15, 15, 15));
barContents.setAlignment(Pos.BOTTOM_CENTER);

Text urlTxt = new Text("URL:");
urlTxt.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));

final TextArea url = new TextArea();
url.setText("http://www.twitter.com");
url.setPrefSize(370, 25);
url.setPrefRowCount(1);
url.setStyle("-fx-background-color: #336699");

ImageView go = new ImageView(
    new Image(Browser.class.getResourceAsStream(
        "images/search.png")));

barContents.getChildren().addAll(urlTxt, url, go);

```

- Create a background rectangle with a **LinearGradient** as the top bar's filling
 - Set the **Stroke** color
 - Set the **arcs** for the rectangle
- Create a **stack pane**
 - Add the background rectangle
 - Add the bar contents (**HBox**)

```

Rectangle barBackground = new Rectangle(500.0, 50.0);
barBackground.setFill(
    new LinearGradient(0, 0, 0, 1, true, CycleMethod.NO_CYCLE,
        new Stop[]{
            new Stop(0, Color.web("#4977A3")),
            new Stop(0.5, Color.web("#B0C6DA")),
            new Stop(1, Color.web("#9CB6CF")),
        }));
barBackground.setStroke(Color.web("#D0E6FA"));

```

```
barBackground.setArcHeight(3.5);
barBackground.setArcWidth(3.5);

StackPane stack = new StackPane();
stack.setPadding(new Insets(10, 10, 10, 10));
stack.getChildren().addAll(barBackground, barContents);
```

- Create a WebEngine and load <http://www.twitter.com>
- Create a WebView, and use the previous WebEngine
- Set Layouts and size

```
private WebEngine myWeb;
...
WebView webView = new WebView();
myWeb = webView.getEngine();
try {
    myWeb.load("http://www.twitter.com");
} catch (Exception e) {
    System.out.println("Exception while opening the url");
}
webView.setLayoutX(10);
webView.setLayoutY(70);
webView.setPrefSize(500, 420);
```

- Set the setOnMouseClicked for the go image
 - Get the value of the url
 - load the new page in the browser

```
go.setOnMouseClicked(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent me) {
        try {
            myWeb.load(url.getText());
        } catch (Exception e) {
            System.out.println("Exception while opening the url");
        }
    }
});
```

- Finally add the stack and the web view into the root
- Your code should look like this

```
5. package browser;
6.
7. import javafx.application.Application;
8. import javafx.beans.property.StringProperty;
9. import javafx.event.EventHandler;
10. import javafx.geometry.Insets;
11. import javafx.geometry.Pos;
12. import javafx.scene.Group;
```

```

13. import javafx.scene.Scene;
14. import javafx.scene.control.TextArea;
15. import javafx.scene.image.Image;
16. import javafx.scene.image.ImageView;
17. import javafx.scene.input.MouseEvent;
18. import javafx.scene.layout.HBox;
19. import javafx.scene.layout.StackPane;
20. import javafx.scene.paint.Color;
21. import javafx.scene.paint.CycleMethod;
22. import javafx.scene.paint.LinearGradient;
23. import javafx.scene.paint.Stop;
24. import javafx.scene.shape.Rectangle;
25. import javafx.scene.text.Font;
26. import javafx.scene.text.FontWeight;
27. import javafx.scene.text.Text;
28. import javafx.scene.web.WebEngine;
29. import javafx.scene.web.WebView;
30. import javafx.stage.Stage;
31.
32. /**
33.  *
34.  * @author angie
35.  */
36. public class Browser extends Application {
37.     private WebEngine myWeb;
38.     /**
39.      * @param args the command line arguments
40.      */
41.     public static void main(String[] args) {
42.         Application.launch(Browser.class, args);
43.     }
44.
45.     @Override
46.     public void start(Stage primaryStage) {
47.         primaryStage.setTitle("My Own Browser");
48.         Group root = new Group();
49.         Scene scene = new Scene(root, 520, 500);
50.
51.         HBox barContents = new HBox();
52.         barContents.setSpacing(15);
53.         barContents.setPadding(new Insets(15, 15, 15, 15));
54.         barContents.setAlignment(Pos.BOTTOM_CENTER);
55.
56.         Text urlTxt = new Text("URL:");
57.         urlTxt.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
58.
59.         final TextArea url = new TextArea();
60.         url.setPrefSize(370, 25);
61.         url.setPrefRowCount(1);
62.         url.setStyle("-fx-background-color: #336699");
63.
64.         ImageView go = new ImageView(

```



```

        new Image(Browser.class.getResourceAsStream
            ("images/search.png"))));
65.     barContents.getChildren().addAll(urlTxt, url, go);
66.
67.     StackPane stack = new StackPane();
68.     stack.setPadding(new Insets(10, 10, 10, 10));
69.     Rectangle barBackground = new Rectangle(500.0, 50.0);
70.     barBackground.setFill(
71.         new LinearGradient(0, 0, 0, 1, true, CycleMethod.NO_CYCLE,
72.             new Stop[]{
73.                 new Stop(0, Color.web("#4977A3")),
74.                 new Stop(0.5, Color.web("#B0C6DA")),
75.                 new Stop(1, Color.web("#9CB6CF")),
76.             });
77.     barBackground.setStroke(Color.web("#D0E6FA"));
78.     barBackground.setArcHeight(3.5);
79.     barBackground.setArcWidth(3.5);
80.     stack.getChildren().addAll(barBackground, barContents);
81.
82.     myWeb = new WebEngine();
83.     try {
84.         myWeb.load("http://www.twitter.com");
85.     } catch (Exception e) {
86.         System.out.println("Exception while opening the url");
87.     }
88.     WebView webView = new WebView(myWeb);
89.     webView.setLayoutX(10);
90.     webView.setLayoutY(70);
91.     webView.setPrefSize(500, 420);
92.
93.     go.setOnMouseClicked(new EventHandler<MouseEvent>() {
94.         public void handle(MouseEvent me) {
95.             try {
96.                 myWeb.load(url.getText());
97.             } catch (Exception e) {
98.                 System.out.println("Exception while opening the url");
99.             }
100.        }
101.    });
102.
103.    root.getChildren().addAll(stack, webView);
104.    primaryStage.setScene(scene);
105.    primaryStage.show();
106. }
107. }

```

Exercise 5: Media (Advanced exercise)

The JavaFX media concept is based on the following entities.

- Media – A media resource, containing information about the media, such as its source, resolution, and metadata
- MediaPlayer – The key component providing the controls for playing media
- MediaView – A Node object to support animation, translucency, and effects

Each element of the media functionality is available through the JavaFX API.

In this exercise you will be creating the following Media Player, you already has a lot of expertise with JavaFX to build it on your own. Take advantage of the JavaFX APIs docs if you got any question.

Use the following URL for your media file:

- "<http://sun.edgeboss.net/download/sun/transfer/oow2010.flv>"



Suggested solution

```
5. package mediaTest;  
6.  
7. import javafx.application.Application;  
8. import javafx.application.Platform;
```

```

9. import javafx.beans.InvalidListener;
10. import javafx.beans.Observable;
11. import javafx.event.ActionEvent;
12. import javafx.event.EventHandler;
13. import javafx.geometry.Insets;
14. import javafx.geometry.Pos;
15. import javafx.scene.Group;
16. import javafx.scene.Scene;
17. import javafx.scene.control.Button;
18. import javafx.scene.control.Slider;
19. import javafx.scene.layout.BorderPane;
20. import javafx.scene.layout.HBox;
21. import javafx.scene.media.Media;
22. import javafx.scene.media.MediaPlayer;
23. import javafx.scene.media.MediaPlayer.Status;
24. import javafx.scene.media.MediaView;
25. import javafx.scene.paint.Color;
26. import javafx.stage.Stage;
27. import javafx.util.Duration;
28.
29. /**
30.  *
31.  * @author angie
32.  */
33. public class MediaTest extends Application {
34.
35.     private static final String MEDIA_PATH = "media/omgrobots.flv";
36.     private MediaPlayer mediaPlayer;
37.     private Slider timeSlider;
38.     private Button playButton;
39.     private final boolean repeat = false;
40.     private boolean stopRequested = false;
41.     private boolean atEndOfMedia = false;
42.     private Duration duration;
43.
44.     /**
45.      * @param args the command line arguments
46.      */
47.     public static void main(String[] args) {
48.         Application.launch(MediaTest.class, args);
49.     }
50.
51.     @Override
52.     public void start(Stage primaryStage) {
53.         primaryStage.setTitle("Media Player");
54.         Group root = new Group();
55.         Scene scene = new Scene(root, 400, 360, Color.DARKGREY);
56.
57.         mediaPlayer = new MediaPlayer(new Media(getClass().getResource(
58.             MEDIA_PATH).toString()));
59.         MediaView mediaView = new MediaView();
60.         mediaView.setMediaPlayer(mediaPlayer);

```

```

61.     mediaView.setLayoutX(20);
62.     mediaView.setLayoutY(20);
63.
64.     setUpMediaPlayerBehavior();
65.
66.     BorderPane mainPane = new BorderPane();
67.     mainPane.setCenter(mediaView);
68.     mainPane.setPadding(new Insets(20,20,20,20));
69.
70.     HBox controls = new HBox();
71.     controls.setAlignment(Pos.CENTER);
72.     controls.setPadding(new Insets(5,5,5,5));
73.     controls.setSpacing(15);
74.
75.     playButton = new Button(">");
76.     playButton.setOnAction(new EventHandler<ActionEvent>() {
77.         public void handle(ActionEvent e) {
78.             Status status = mediaPlayer.getStatus();
79.             if (status == Status.UNKNOWN
80.                 || status == Status.HALTED) {
81.                 // don't do anything in these states
82.                 return;
83.             }
84.             if (status == Status.PAUSED
85.                 || status == Status.READY
86.                 || status == Status.STOPPED) {
87.                 // rewind the movie if we're sitting at the end
88.                 if (atEndOfMedia) {
89.                     mediaPlayer.seek(mediaPlayer.getStartTime());
90.                     atEndOfMedia = false;
91.                 }
92.                 mediaPlayer.play();
93.             } else {
94.                 mediaPlayer.pause();
95.             }
96.         }
97.     });
98.
99.     timeSlider = new Slider();
100.    timeSlider.setMinWidth(50);
101.    timeSlider.setMaxWidth(Double.MAX_VALUE);
102.    timeSlider.valueProperty().addListener(new InvalidationListener() {
103.        public void invalidated(Observable o) {
104.            if (timeSlider.isValueChanging()) {
105.                // multiply duration by percentage calculated by slider position
106.                mediaPlayer.seek
107.                    (duration.multiply(timeSlider.getValue() / 100.0));
108.            }
109.        }
110.    });
111.
112.

```

```

113.         controls.getChildren().addAll(playButton, timeSlider);
114.         mainPane.setBottom(controls);
115.         root.getChildren().add(mainPane);
116.         primaryStage.setScene(scene);
117.         primaryStage.show();
118.     }
119.
120.     private void setUpMediaPlayerBehavior() {
121.         mediaPlayer.currentTimeProperty().
122.             addListener(new InvalidationListener() {
123.                 public void invalidated(Observable o) {
124.                     updateValues();
125.                 }
126.             });
127.
128.         mediaPlayer.setOnPlay(new Runnable() {
129.             public void run() {
130.                 if (stopRequested) {
131.                     mediaPlayer.pause();
132.                     stopRequested = false;
133.                 } else {
134.                     playButton.setText("||");
135.                 }
136.             }
137.         });
138.
139.         mediaPlayer.setOnPaused(new Runnable() {
140.             public void run() {
141.                 playButton.setText(">");
142.             }
143.         });
144.
145.         mediaPlayer.setOnReady(new Runnable() {
146.             public void run() {
147.                 duration = mediaPlayer.getMedia().getDuration();
148.                 updateValues();
149.             }
150.         });
151.
152.         mediaPlayer.setCycleCount(repeat ? MediaPlayer.INDEFINITE : 1);
153.         mediaPlayer.setOnEndOfMedia(new Runnable() {
154.
155.             public void run() {
156.                 if (!repeat) {
157.                     playButton.setText(">");
158.                     stopRequested = true;
159.                     atEndOfMedia = true;
160.                 }
161.             }
162.         });
163.     }
164.

```

```

165.     protected void updateValues() {
166.         if (timeSlider != null) {
167.             Platform.runLater(new Runnable() {
168.                 public void run() {
169.                     Duration currentTime = mediaPlayer.getCurrentTime();
170.                     timeSlider.setDisable(duration.isUnknown());
171.                     if (!timeSlider.isDisabled()
172.                         && duration.greaterThan(Duration.ZERO)
173.                         && !timeSlider.isValueChanging()) {
174.                         timeSlider.setValue(currentTime.divide(duration)
175.                                                 .toMillis()*100.0);
176.                     }
177.                 }
178.             });
179.         }
180.     }

```

Congratulations, you have successfully finished your first JavaFX HOL!!!!