Lab 6 - Working with JUnit and JavaFX

Part A - JUnit

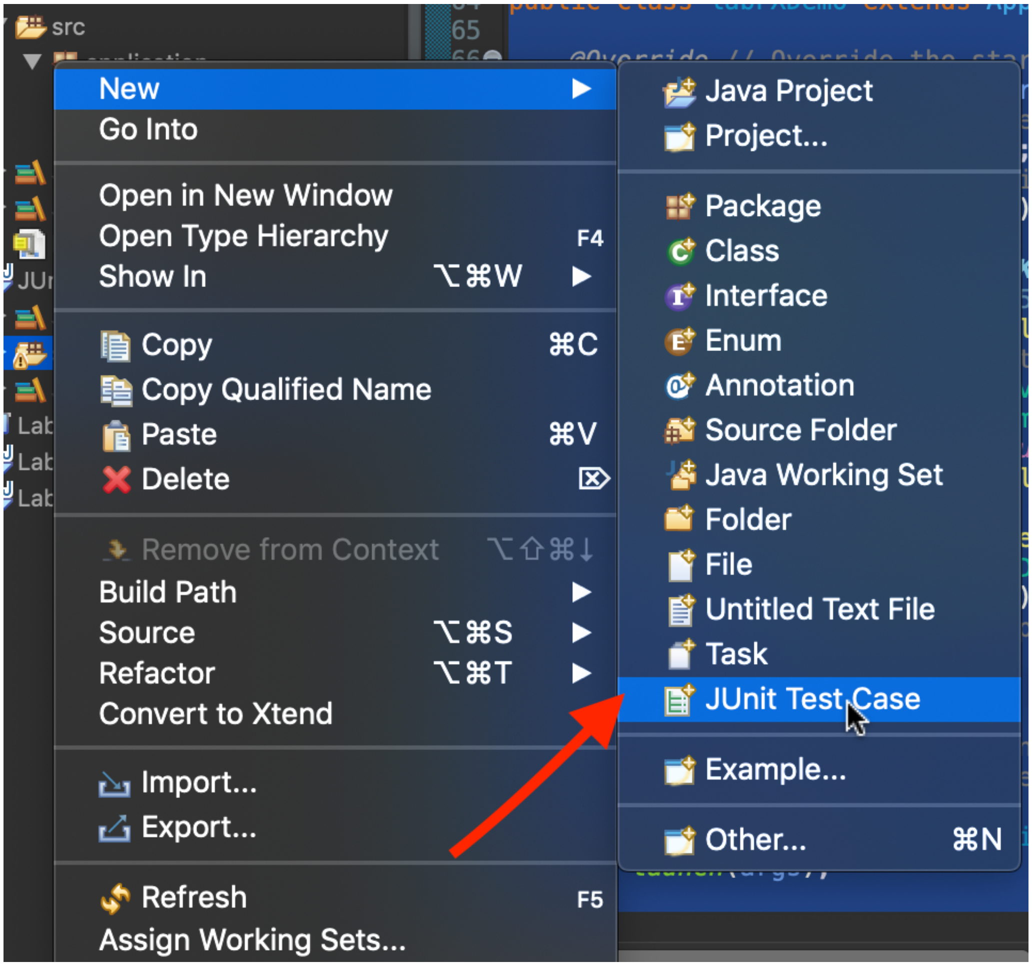
This section of the lab will provide you with experience in how JUnit works in Eclipse.

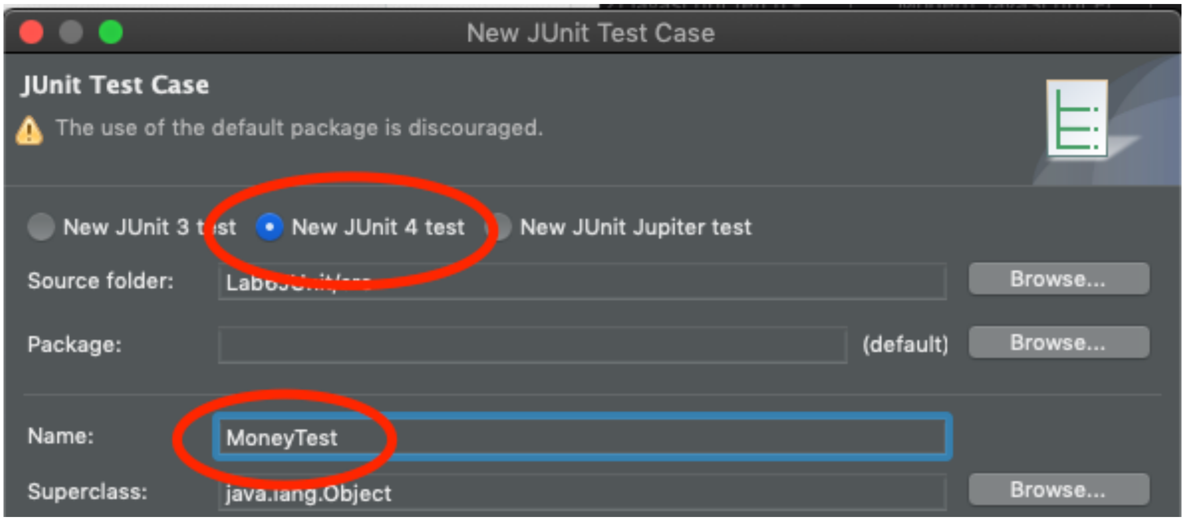
For this section you need to:

1. Make a new **Java 8** project called `Lab6JUnit`
2. Add a new class to you project called `Money`
3. Place the following code in your `Money` class

```
public class Money {
      private int cAmount;
      private String cCurrency;
      // constructor for creating a money object
      public Money(int amount, String currency) {
           cAmount = amount;
           cCurrency = currency;
      }

      // set money
      public int getAmount() {
           return cAmount;
      }

      // get money
      public String getCurrency() {
           return cCurrency;
      }

      public Money add(Money m) throws Exception {
      if (m.getAmount()<0)
          throw new Exception("Money cannot be negative");
      return new Money(cAmount - m.getAmount(), getCurrency());
      }

  @Override
  public boolean equals(Object anObject) {
     if (anObject instanceof Money) {
        Money passedMoney = (Money) anObject;
        if (this.cAmount == passedMoney.getAmount()
          && this.cCurrency.equals(passedMoney.getCurrency()))
          return true;
       }
       return false;
   }
}
```

4. Add a new `JUnit Test Case` to your project (by right-clicking on the `src`). Note the following before you start:
    - Be sure to choose `New JUnit 4 test` on the Create screen
    - Name your test case `MoneyTest`
    - After selecting `Finish`, if Eclipse asks you about adding JUnit to the build path, choose `OK`





New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

○ New JUnit 3 test ● New JUnit 4 test ○ New JUnit Jupiter test

Source folder: Lab6JUnit/src

Package: (default)

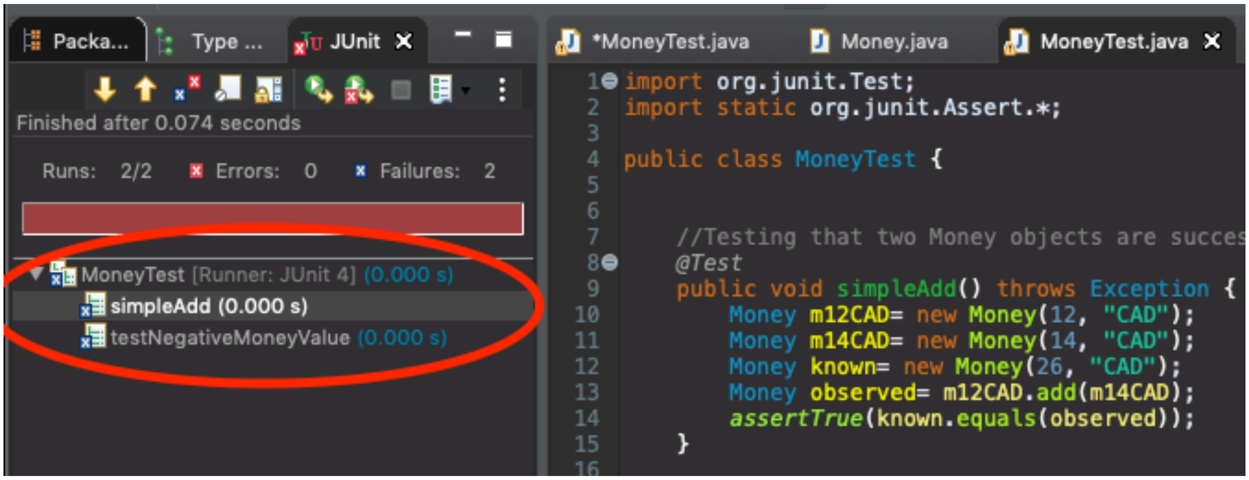
Name: MoneyTest

Superclass: java.lang.Object

5. Adding the following code to your JUnit Test Case file

```
import org.junit.Test;
import static org.junit.Assert.*;

public class MoneyTest {


  //Testing that two Money objects are successfully added together
  @Test
  public void simpleAdd() throws Exception {
    Money m12CAD= new Money(12, "CAD");
    Money m14CAD= new Money(14, "CAD");
    Money known= new Money(26, "CAD");
    Money observed= m12CAD.add(m14CAD);
    assertTrue(known.equals(observed));
  }



  //testing that exception is thrown correctly
  @Test (expected = Exception.class)
  public void testNegativeMoneyValue () throws Exception{
    Money m12CAD= new Money(12, "CAD");
    Money m14CAD= new Money(14, "CAD");
    Money observed= m12CAD.add(m14CAD);
  }
}
```

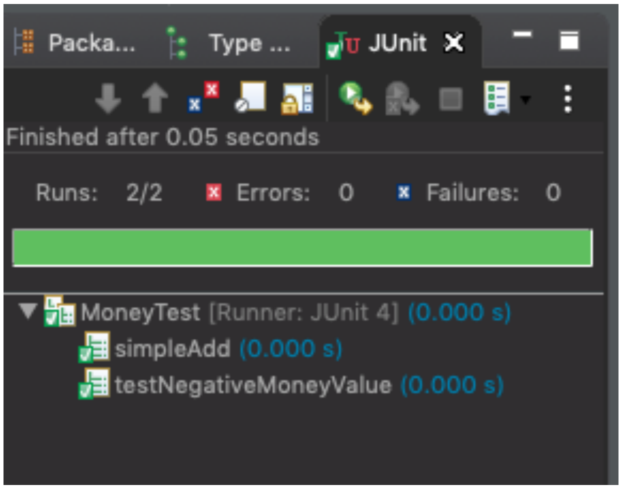
6. **Run** your JUnit Test Case. Notice that both test cases fail.



```
Finished after 0.074 seconds

Runs: 2/2    Errors: 0    Failures: 2

MoneyTest [Runner: JUnit 4] (0.000 s)
    simpleAdd (0.000 s)
    testNegativeMoneyValue (0.000 s)
```

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class MoneyTest {


    //Testing that two Money objects are succes
    @Test
    public void simpleAdd() throws Exception {
        Money m12CAD= new Money(12, "CAD");
        Money m14CAD= new Money(14, "CAD");
        Money known= new Money(26, "CAD");
        Money observed= m12CAD.add(m14CAD);
        assertTrue(known.equals(observed));
    }
```

7. Resolve the issue with both test cases so that they complete successfully. Hint:
   - The `SimpleAdd` test fails is not being done correctly in `Money.java`
   - The `TestNegativeMoneyValue` test fails because of a problem with the test itself (i.e. it's not testing the correct thing)

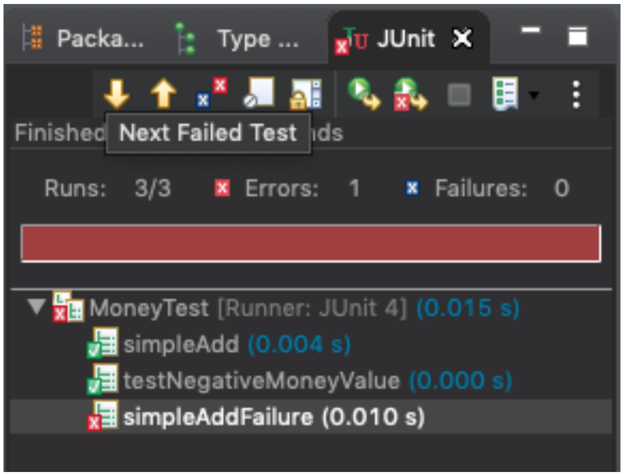


```
Finished after 0.05 seconds

Runs: 2/2    Errors: 0    Failures: 0

MoneyTest [Runner: JUnit 4] (0.000 s)
    simpleAdd (0.000 s)
    testNegativeMoneyValue (0.000 s)
```

8. Add the following test case to the code you already have. Run your test cases again.

```java
//Test is unable to complete successfully because exception is thrown
@Test
public void simpleAddFailure() throws Exception {
  Money m12CAD= new Money(12, "CAD");
  Money m14CAD= new Money(-14, "CAD");
  Money known= new Money(26, "CAD");
  Money observed= m12CAD.add(m14CAD);
  assertTrue(known.equals(observed));
}
```

9. Notice that the new test case is marked as `Error` rather then `Failure`. This test was unable to complete because `add` threw *an exception*. In JUnit there is a difference betwen a test not producing the expected result (*Failure*) and a test that is unable to complete (*Error*). Determine why during this test `add` threw the exception and fix the test case.



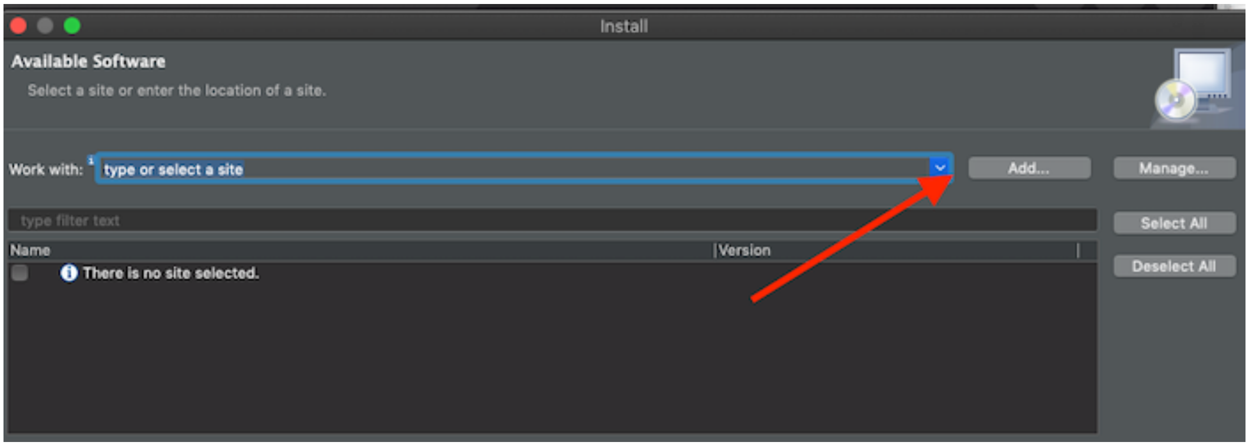
```
Runs:  3/3     Errors:  1     Failures:  0

MoneyTest [Runner: JUnit 4] (0.015 s)
    simpleAdd (0.004 s)
    testNegativeMoneyValue (0.000 s)
    simpleAddFailure (0.010 s)
```

10. Your final result for this section should be three successful test cases. Take a screenshot which includes the code in your MoneyTest Class, and the three tests with green checkmarks. Include this screenshot in a Google/Word Document

Part B - JavaFX

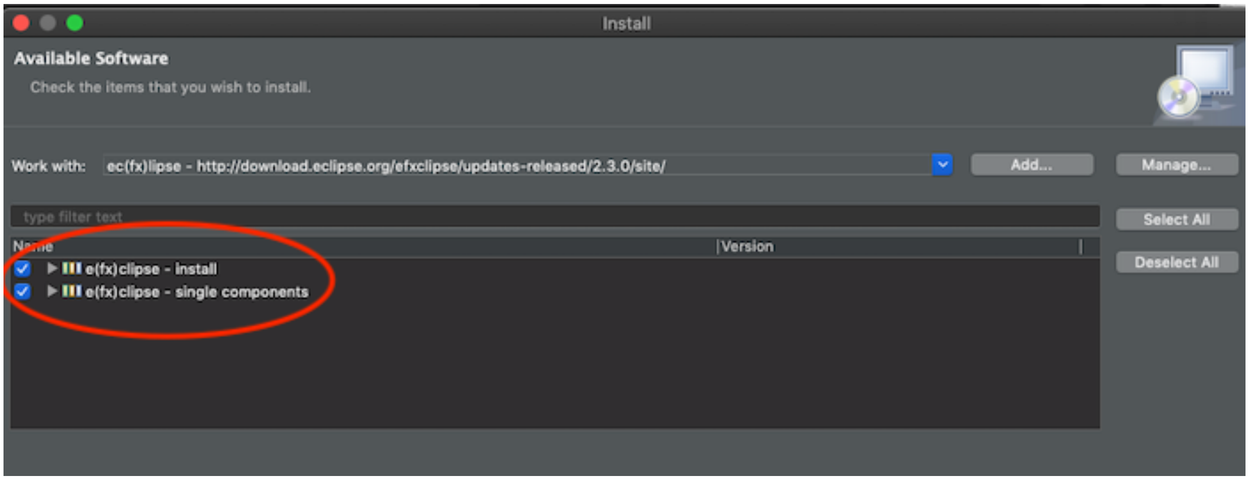
This section you will learn to install JavaFX correctly, and work with basic example.

1. In Eclipse, go to `Help` in th menu bar, and choose `Install new software`
2. In the **Available Software** window, press the `Add` button



3. In the **Add Repository** window enter the following information, then press `Add`:
   - name: `ec(fx)lipse`
   - location: `http://download.eclipse.org/efxclipse/updates-released/2.3.0/site/`

4. Check the box beside both components that appear, and select `Next`



Install

Available Software

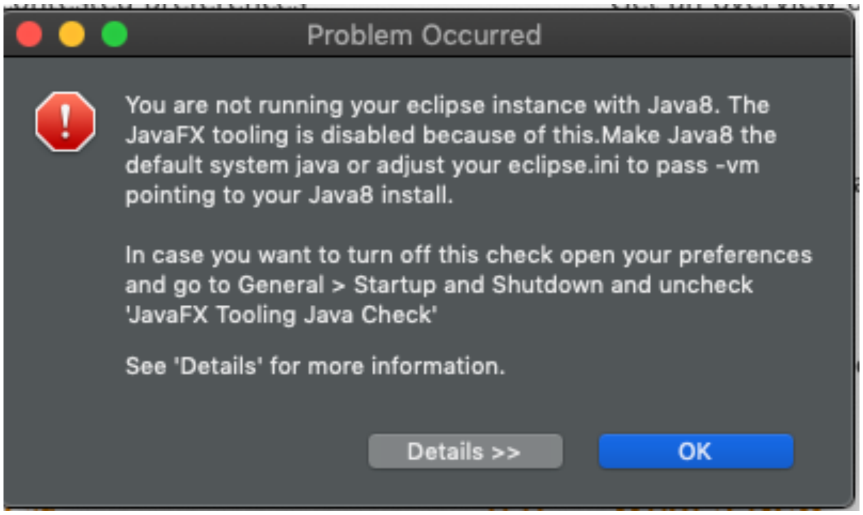
Check the items that you wish to install.

Work with: ec(fx)lipse - http://download.eclipse.org/efxclipse/updates-released/2.3.0/site/

type filter text

Name	Version
☑ e(fx)clipse - install	
☑ e(fx)clipse - single components	

5. Select `Next` again on the following screen, then on the final screen, **agree to the terms and conditions** then press `Finish`
6. Allow Eclipse to restart
7. If you have Java 13 installed, you will see the following error on restart.



Problem Occurred

You are not running your eclipse instance with Java8. The JavaFX tooling is disabled because of this.Make Java8 the default system java or adjust your eclipse.ini to pass -vm pointing to your Java8 install.

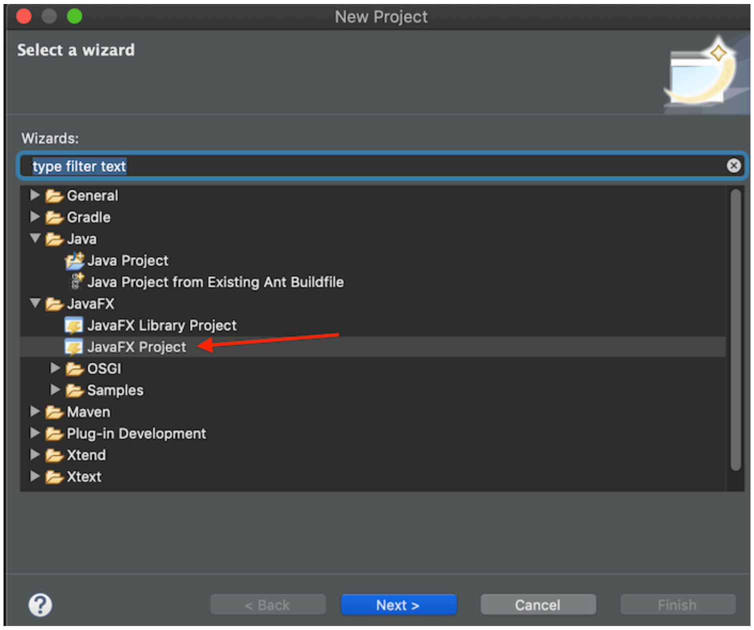
In case you want to turn off this check open your preferences and go to General > Startup and Shutdown and uncheck 'JavaFX Tooling Java Check'

See 'Details' for more information.

You have a couple of options:

- You can ignore it.
  - It is however, important to note that **JavaFX doesn't work in Java 13**, so it's **essential** that any JavaFX projects you create are **Java 8** projects
- You can disable the check in `Preferences > General > Startup and Shutdown` (which stops the error from appearing)
  - You will still need to ensure any JavaFX projects you create are Java 8 projects
- You can configure Eclipse to default to Java 8 projects in `Preferences > Java > Compiler` by setting the `Compiler Compliance Level` to `1.8`.
  - This will make any project you create default to Java 8, so unless you change it at project creation time it should work correctly.

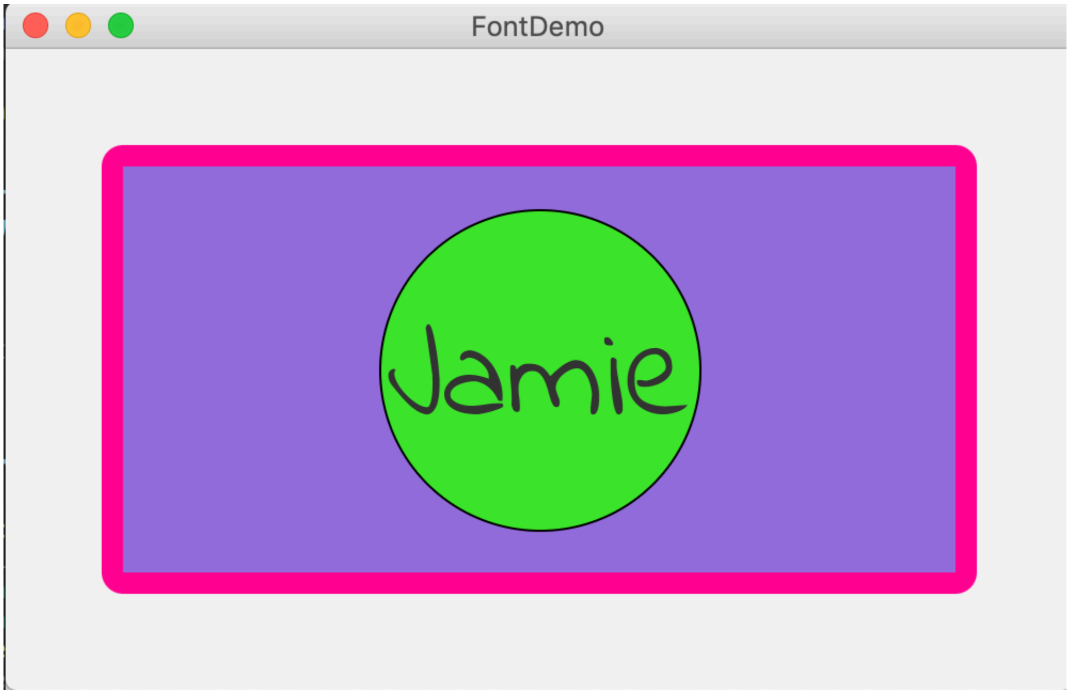
8. Make a new JavaFX project through `File > New Project` then selecting `JavaFX Project` in the wizard.



9. Include the following code in your `Main.java` (you will need to allow Eclipse to rename the file to fix the error resulting from the File name and the Class name being different). Run your project to see the results.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.*;
import javafx.scene.control.*;
import javafx.stage.Stage;
public class labFXDemo extends Application {

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new StackPane();
    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
    pane.getChildren().add(circle); // Add circle to the pane
    // Create a label and set its properties
    Label label = new Label("JavaFX");
    label.setFont(Font.font("Times New Roman",
      FontWeight.BOLD, FontPosture.ITALIC, 20));
    pane.getChildren().add(label);
    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 400, 300);
    primaryStage.setTitle("FontDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

  }
  /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
      launch(args);
  }
}
```

10. Modify the code to add a `Rectangle` object behind the `Circle`. Things of note:
    - The project includes a *stack* pane, so objects are stacked *bottom* to *top* in the order they are added, so to see everything you need to add them to the pane in the correct order.
    - The `Rectangle` object has the expected `height` and `width` properties as well as `arcHeight` and `arcWidth` (both of which used together produce rounded corners). All of these properties have `set` methods to set their values (similar to what's already there for Circle)
11. Play with the properties for your `rectangle`, `circle` and `label` to product a nice, unique output containing your name.



12. In your document from Part A, include the code from your java file (either as a screen capture, or as text), and a screen capture of your final FX Pane when your project is running (like mine above).
13. Turn your Document into a pdf and submit it via the Dropbox on Blackboard.