



# Simple FX Implementation Reference Guide



# Table of Contents

<a href="#">1. Requirements</a>	3
<a href="#">2. Development Environment Setup</a>	4
<a href="#">3. Configuring Your Classpath</a>	5
<a href="#">4. Getting Started Using SimpleFX Framework Using A Hands-On Example</a>	6
<a href="#">5. Simple Exercises Using The Example Application</a>	11

This guide is meant to get you started with attaching the SimpleFXFramework to your JavaFX application. There will be step-by-step instructions as well as an example to help you get your application wrapped up within the simplefx framework. Good Luck!

## 1. Requirements

The below sections describe what is required for your development process using the simplefx framework.

### Required Software and Third-party libraries

JDK 1.8 update 40 or greater

Eclipse IDE (Works with Java 1.8)

### Third Party libraries (minimal) [jars that are required]

activation.jar

commons-lang-2.3.jar

icu4j-3.4.4.jar

jasyp-1.9.2.jar

mail.jar

### Third Party libraries (all) [ant jars added for running embedded ant scripts from application]

activation.jar

ant.jar

ant-contrib-1.0b3.jar

ant-javamail.jar

ant-launcher.jar

commons-lang-2.3.jar

icu4j-3.4.4.jar

jasyp-1.9.2.jar

mail.jar

Location of the above jars are located here: <http://isuwsphere2svr/SharedJar/simplefx/thirdpartylibs>

### JavaDocs

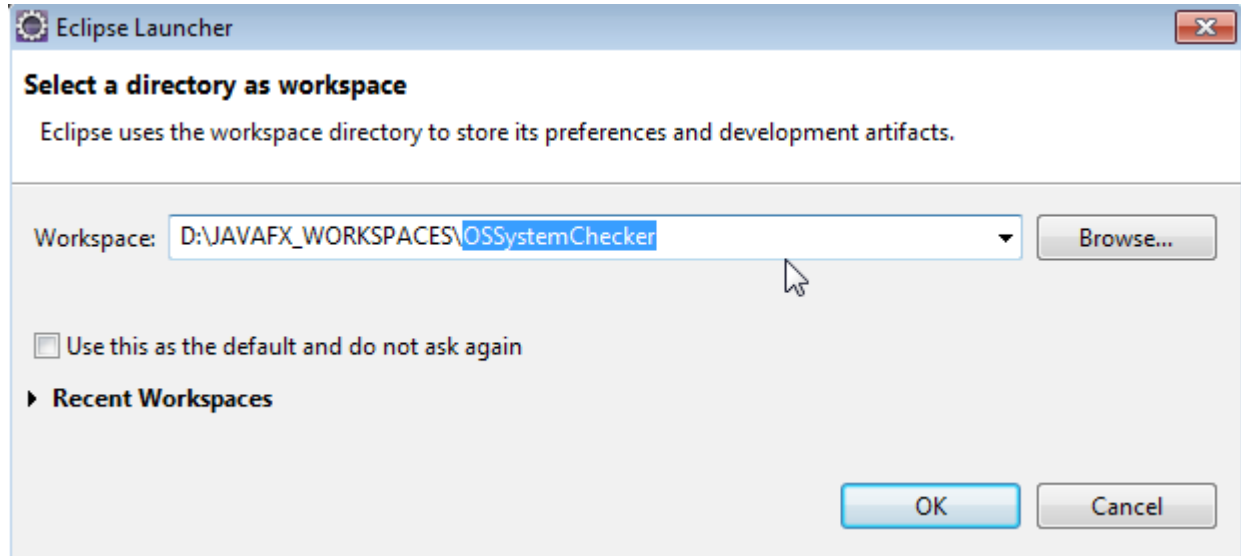
Javadocs are comments in the source code which are used to provide developers with in-depth information about the features of the simplefx-framework. Javadocs are intended to help you during your development life-cycle. The javadocs for the SimpleFX Framework can be found below:

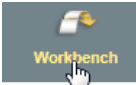
<http://isuwsphere2svr/SharedJar/simplefx/javadoc>

## 2. Development Environment Setup

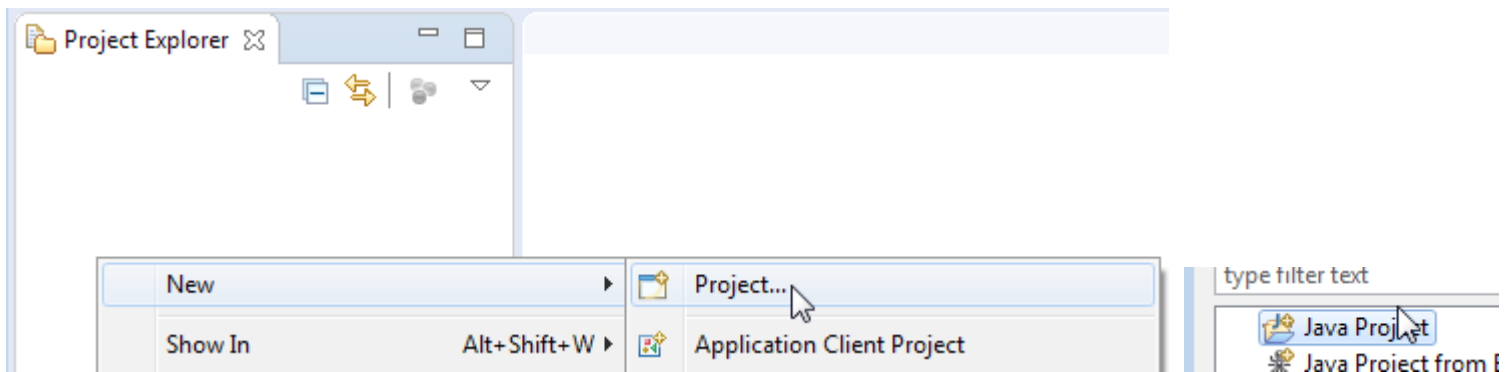
This section will help you to get your Eclipse environment set up to start programming. You should already be familiar with setting up a workspace.

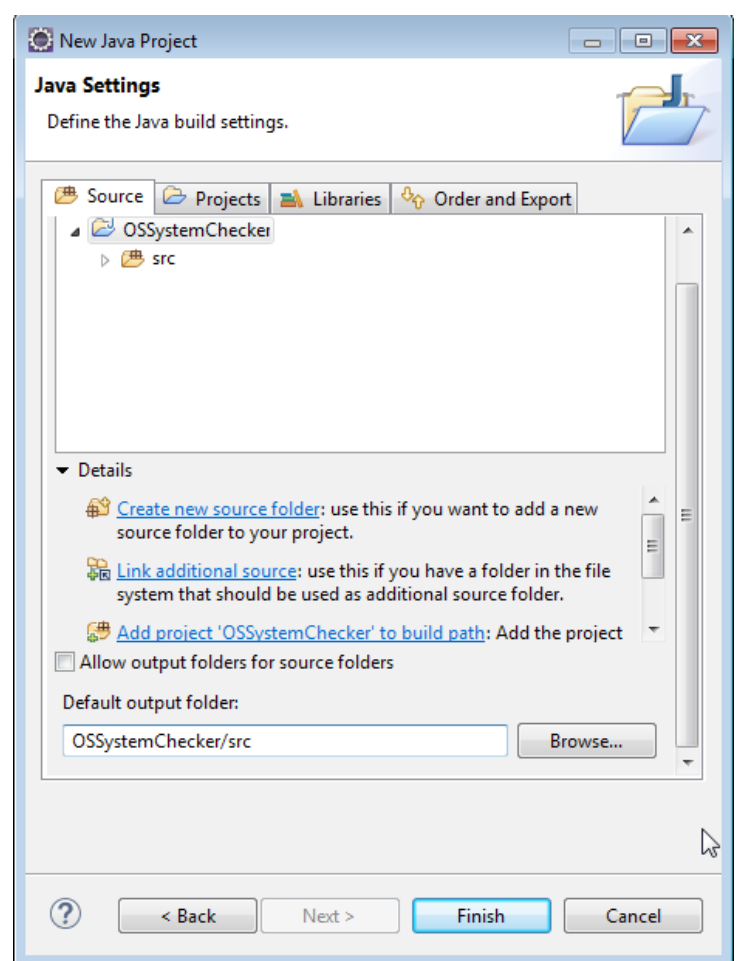
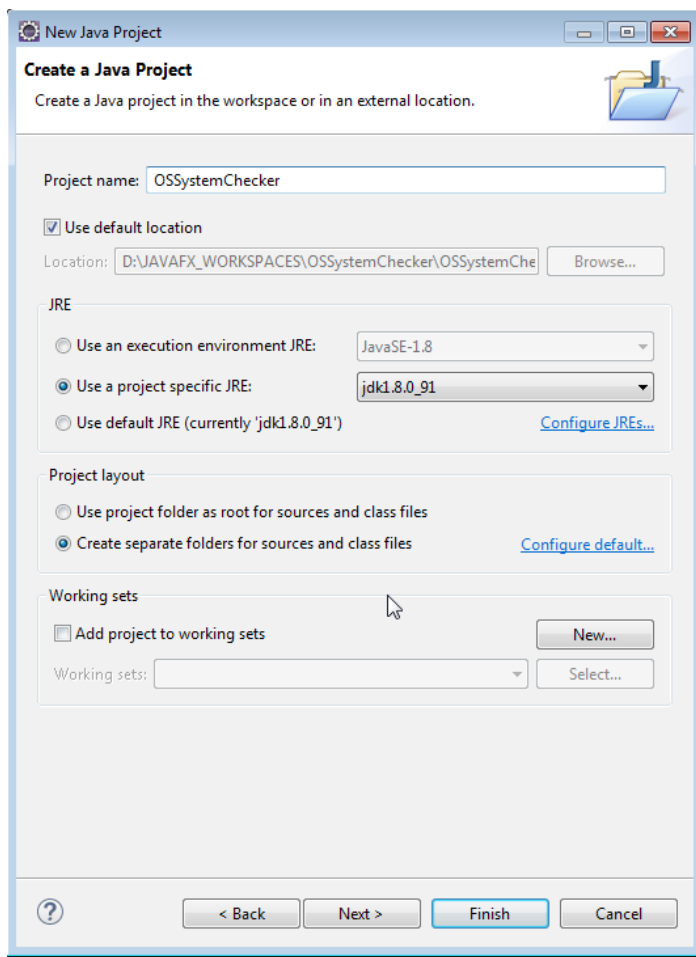
1. **Open** Eclipse and create a new workspace for you JavaFX application. *Hint:* You can type a path into the **Workspace** textbox and eclipse will create a workspace for you. *For help refer to the screen shots below.*



2. Open  (Workbench).

3. Create a new **Java Project** by right-clicking within the **Project Explorer** View and selecting **New > Project... > Java Project**. *For help refer to the screen shots below.*

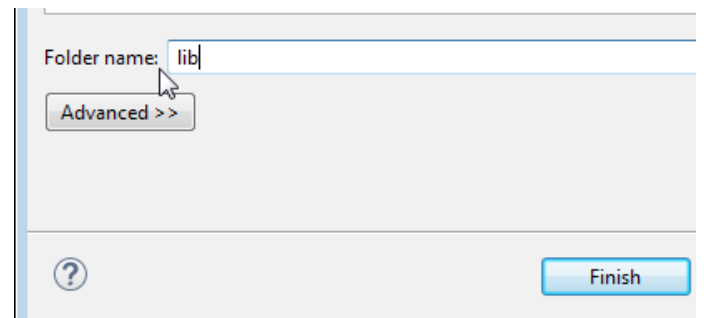
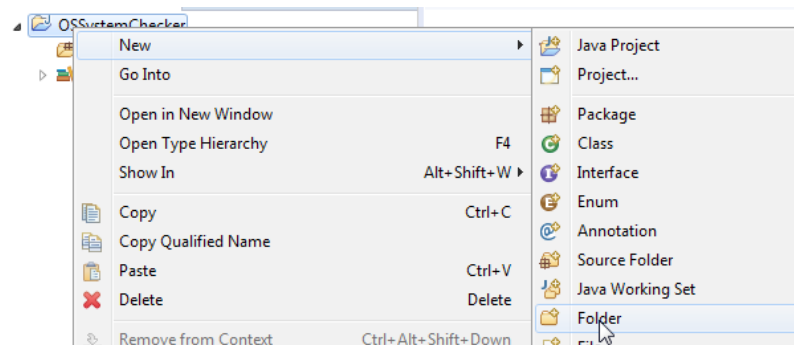




### 3. Configuring Your Classpath

This section will show you how to configure your classpath used for compiling your JavaFX GUI project powered by the SimpleFX Framework.

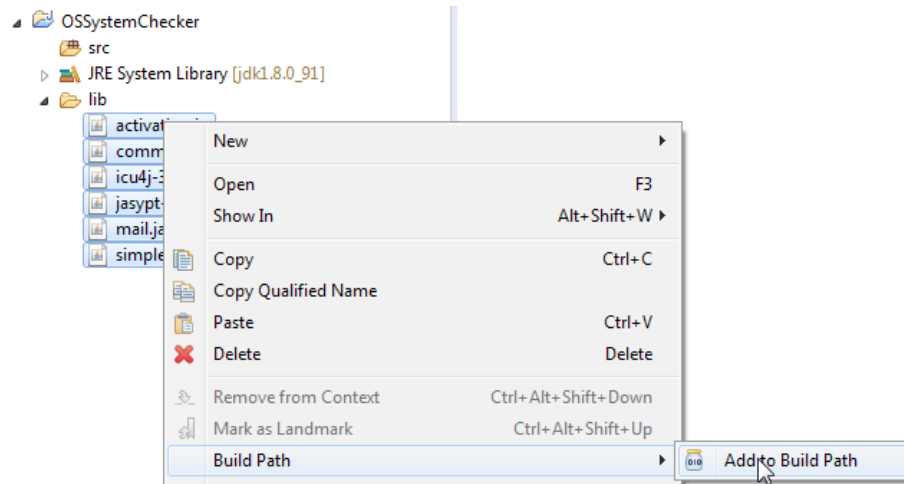
1. Right-click on your **Java Project** > **New** > **Folder** > Type **lib** inside of the **Folder name** textbox > Click **Finish**. For help refer to the screen shots below.



2. Locate the newest version of the simplefx-framework-x.x.x.jar and copy it into the lib folder created within the above step. (simplefx-framework-x.x.x.jar Location is [\\isusphere2svr\SharedJar\simplefx](#))

Locate the minimal required java libraries and copy them into the lib folder created above. (Minimal Required Libraries Location is [\\isusphere2svr\SharedJar\simplefx\thirdpartylibs\minimal](#)).

3. **Open** the **lib** folder, then **select all the jars** within the folder. **Right-click** on one of the jar files **> Build Path > Add to Build Path**. This will add all the libraries to your classpath. *For help refer to the screen shot below.*



## 4. Getting Started Using SimpleFX Framework Using A Hands-On Example

This section will help you get your JavaFX GUI project set up for development using the SimpleFX framework. This section also provides you a simple example application used to help you get started. Javadocs are provided.

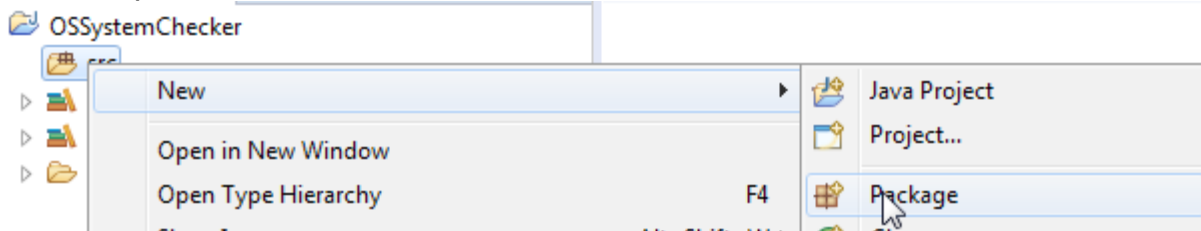
1. Create 2 packages. Named similar to the following packages:

**gov.doc.isu.ossystem.main**  
**gov.doc.isu.ossystem.view**

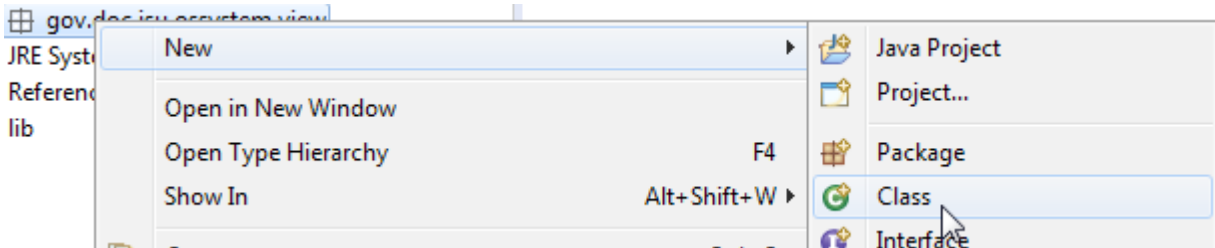
If you do not know how to create packages follow the below steps or else move on to step 2.

- a) **Right-click** on the **src** directory **> New > Package > Enter name of package > Click Finish**
- b) Repeat step **a** above

*For help refer to the screen shot below.*



2. Create a class named **SystemPresenter** within the **gov.doc.isu.ossystem.view** package. During the creation of this class make sure to set the superclass to **SFXViewBuilder** and select the checkbox labeled **Inherited abstract methods**. For help refer to the screen shot below.



Source folder:  

Package:  

☐ Enclosing type:  

---

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:  

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

3. For now copy the example code below in **bold** into your **SystemPresenter** class you created in the above step. This is code is used only for a learning process.

```
package gov.doc.isu.ossystem.view;
```

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

```
import gov.doc.isu.simple.fx.application.SFXViewBuilder;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Parent;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontWeight;
```

```
public class SystemPresenter extends SFXViewBuilder {
```

```
@Override
```

```
protected Parent buildParent() {
```

```
    VBox rootLayout = new VBox(10);  
    rootLayout.setPadding(new Insets(5, 5, 5, 5));
```

```
    VBox details = new VBox(10);  
    displayOSDetails(details);
```

```
    HBox bLayout = new HBox(20);  
    bLayout.setAlignment(Pos.CENTER);  
    Button osButton = new Button("OS Details");  
    Button javaButton = new Button("Java Details");
```

```
    osButton.setOnAction(e -> displayOSDetails(details));  
    javaButton.setOnAction(e -> displayJavaDetails(details));
```

```
    bLayout.getChildren().addAll(osButton, javaButton);
```

```
    rootLayout.getChildren().addAll(details, bLayout);  
    return rootLayout;
```

```
}// end method
```

```
private void displayOSDetails(VBox verticalBox) {
```

```
    verticalBox.getChildren().clear();  
    Label header = new Label("Operating System Details");  
    header.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
```

```
    verticalBox.getChildren().add(header);
```

```
    Font labelFt = Font.font("Courier New", FontWeight.NORMAL, 12);
```

```
    Label[] labels = {new Label("Name: "), new Label("Version: "), new Label("Processors: "), new  
Label("Architecture: ")};
```

```
    List<String> osDetails = getOSDetails();  
    for(int i = 0, j = osDetails.size(); i < j; i++){  
        Label label = labels[i];  
        label.setFont(labelFt);  
        label.setText(label.getText() + osDetails.get(i));
```

```
    }// end for
```

```
    verticalBox.getChildren().addAll(labels);
```

```
}// end method
```



```

private void displayJavaDetails(VBox verticalBox) {
    verticalBox.getChildren().clear();
    Label header = new Label("Java Runtime Details");
    header.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));

    verticalBox.getChildren().add(header);

    Font labelFt = Font.font("Courier New", FontWeight.NORMAL, 12);
    Label[] labels = {new Label("Vendor: "), new Label("Version: "), new Label("User Id: ")};
    List<String> osDetails = getJavaDetails();
    for(int i = 0, j = osDetails.size(); i < j; i++){
        Label label = labels[i];
        label.setFont(labelFt);
        label.setText(label.getText() + osDetails.get(i));
    }// end for
    verticalBox.getChildren().addAll(labels);
}

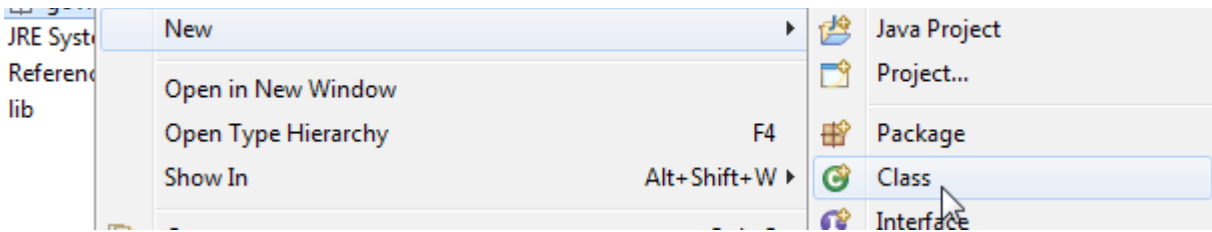
private List<String> getOSDetails() {
    List<String> details = new ArrayList<>();
    details.add(System.getProperty("os.name", "Could Not Find"));
    details.add(System.getProperty("os.version", "Could Not Find"));
    details.add(String.valueOf(Runtime.getRuntime().availableProcessors()));
    details.add(System.getProperty("os.arch", "Could Not Find"));
    return details;
}

private List<String> getJavaDetails() {
    List<String> details = new ArrayList<>();
    details.add(System.getProperty("java.vendor", "Could Not Find"));
    details.add(System.getProperty("java.version", "Could Not Find"));
    details.add(System.getProperty("user.id", "Could Not Find"));
    return details;
}

}

```

4. Create a class named **SystemApplication** within the **gov.doc.isu.ossystem.main** package. This class will contain the `main(String[] args)` method used for starting your application. During the creation of this class make sure to set the superclass to **SFXApplication**, Select the checkbox labeled **Inherited abstract methods**, select checkbox labeled **public static void main(String[] args)**, and select the checkbox labeled **Constructor from superclass**. For help refer to the screen shot below.

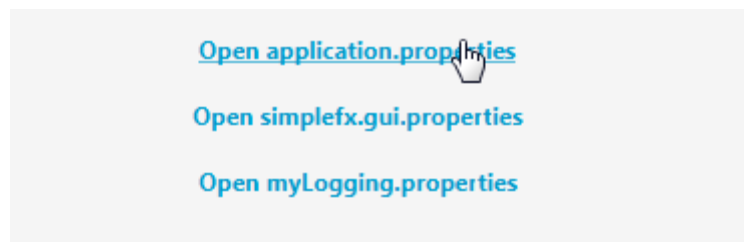
A screenshot of the 'New Class' dialog box in an IDE. The dialog has several fields and checkboxes. The 'Source folder' is 'OSSystemChecker/src'. The 'Package' is 'gov.doc.isu.ossystem.main'. The 'Name' is 'SystemApplication'. The 'Modifiers' section has 'public' selected. The 'Superclass' is 'gov.doc.isu.simple.fx.application.SFXApplication'. The 'Interfaces' section is empty. The 'Which method stubs would you like to create?' section has three checkboxes: 'public static void main(String[] args)', 'Constructors from superclass', and 'Inherited abstract methods', all of which are checked. The 'Do you want to add comments?' section has a checkbox for 'Generate comments' which is unchecked. At the bottom, there are 'Finish' and 'Cancel' buttons.

5. Open the **SystemApplication** class. Edit the return value of the method named **getResourcesParentDirectoryName** to be a String with the value being **SystemApplication**. (Note: You can use any String value that you want here but suggest that you use the name here for example purposes). Edit the **main** method by adding a line that creates a new instance of the **SystemApplication** class using the superclass constructor—you must pass the **args[]** array and the **SystemPresenter.class** to the constructor in order to successfully create an instance of your Application. *For help refer to the screen shots below.*

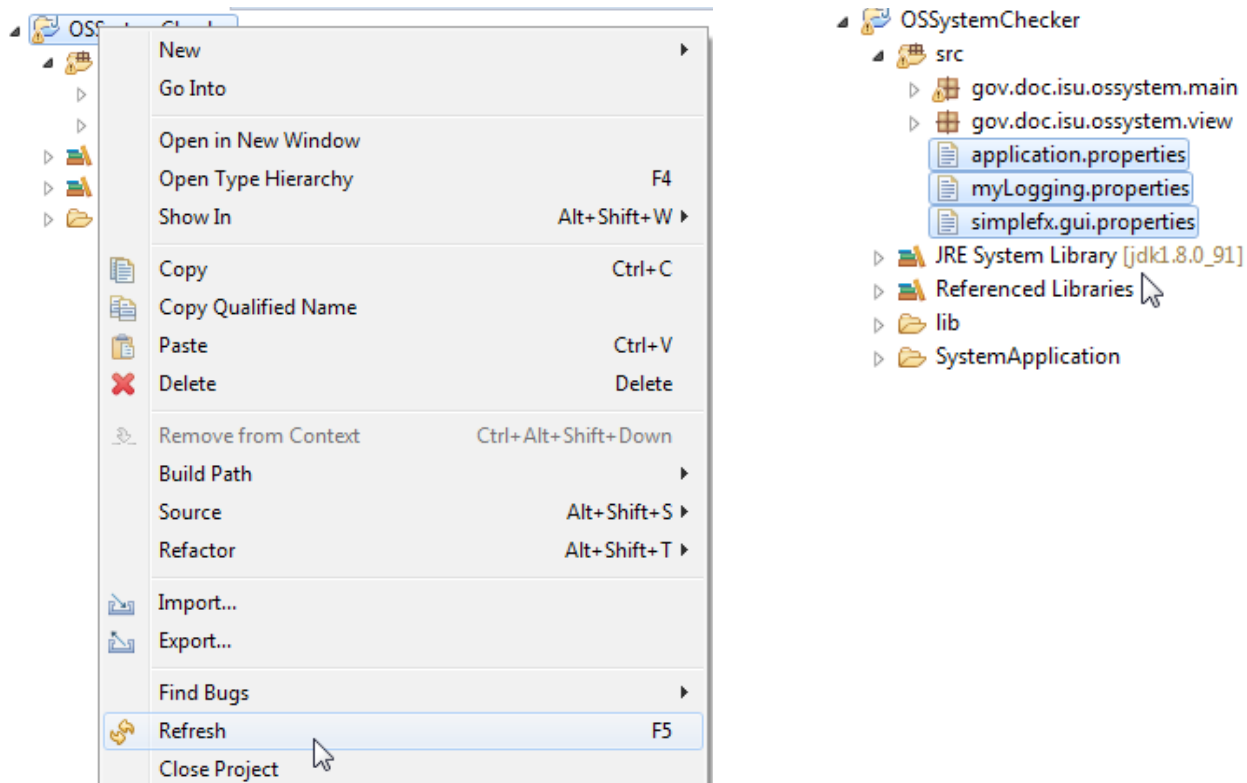
```
@Override
public String getResourcesParentDirectoryName() {
    return "SystemApplication";
}

public static void main(String[] args) {
    new SystemApplication(args, SystemPresenter.class);
} //end method
```

6. **Start** your Java Application and a pop up will display! **Read the pop up window information which will display valuable information for your development process.** Make sure to set up your properties files by clicking on the links displayed within the window. The properties to change should be self explanatory. Once done close the popup window. **NOTE:** These property files will be extracted and written into your **src** folder where you java packages reside. *For help refer to the screen shot below.*



7. Make sure to **Refresh** your **Java Project** and make sure that the \*.properties files do in fact exist. **Start** the **Java Application** again. Your Java GUI Application is now ready to be developed using the simplefx-framework. Go ahead and modify the code as you wish. You are now ready to start Developing and getting familiar with the simplefx-framework. Good Luck! *For help refer to the screen shot below.*



## 5. Simple Exercises Using The Example Application

This section lists a few exercises that will get some hands-on experience using the simplefx-framework. You will also be required to use the javadocs to complete the exercises below.

### Exercise 1

Change the default FX application icon to a different icon.

### Exercise 2

Add the Standard ISU MenuBar to the GUI window.