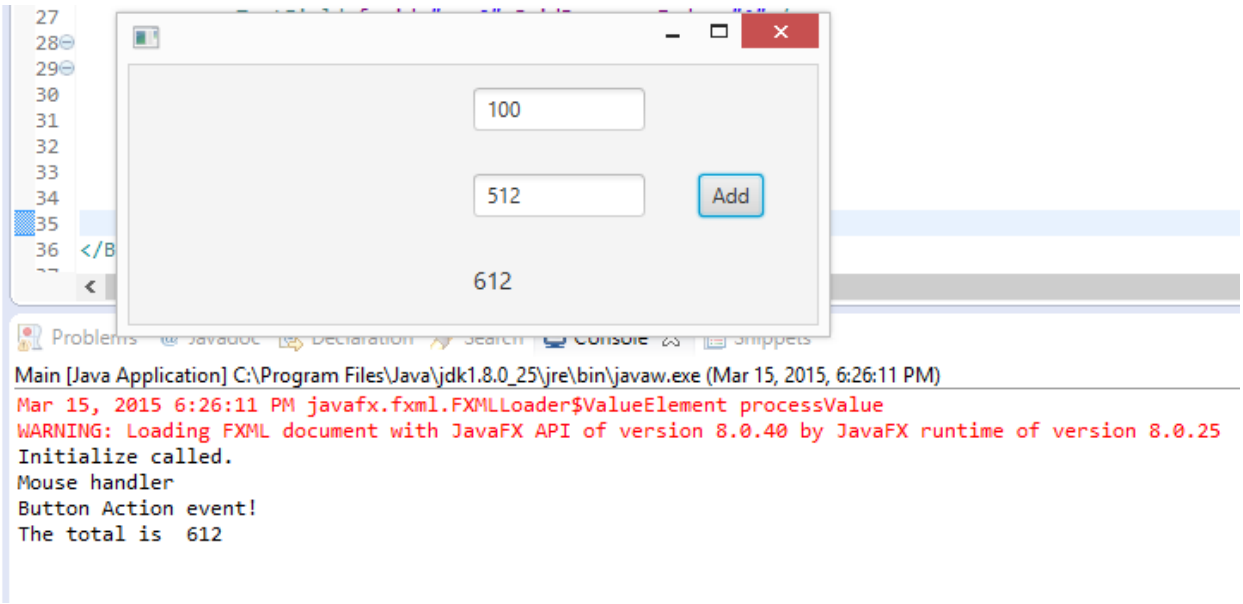
CIS 494 – Business Systems Development with Java – Spring 2015

In-class Exercise 8: Using FXML and Scene Builder

Due on Blackboard: Tonight, March 16 by 11:59 PM

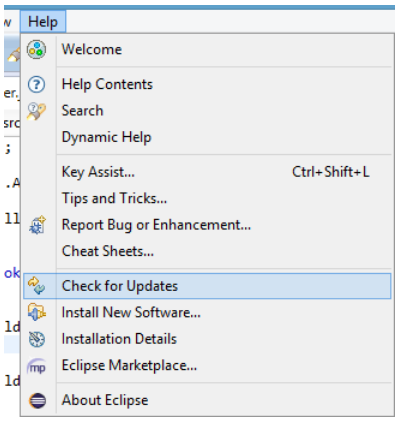
Utilize FXML generated using the Scene Builder application to generate a JavaFX user interface. Learn how to implement GUI controls using XML.

Sample Output



Note: You might need the newest version of Eclipse – either through update or by doing a fresh install!

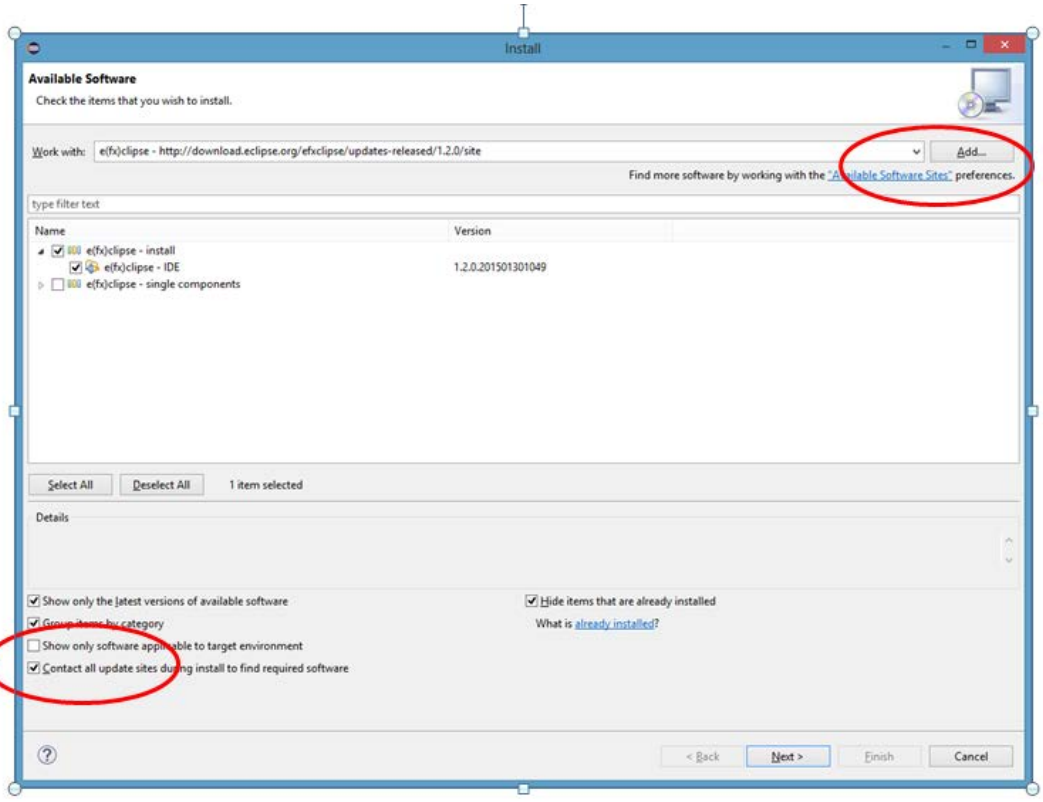
1. Copy your Eclipse folder into a different location and create a backup. Then, update your Eclipse by going into the Help menu and clicking **Check for Updates** (see below). Restart Eclipse as necessary throughout these instructions.



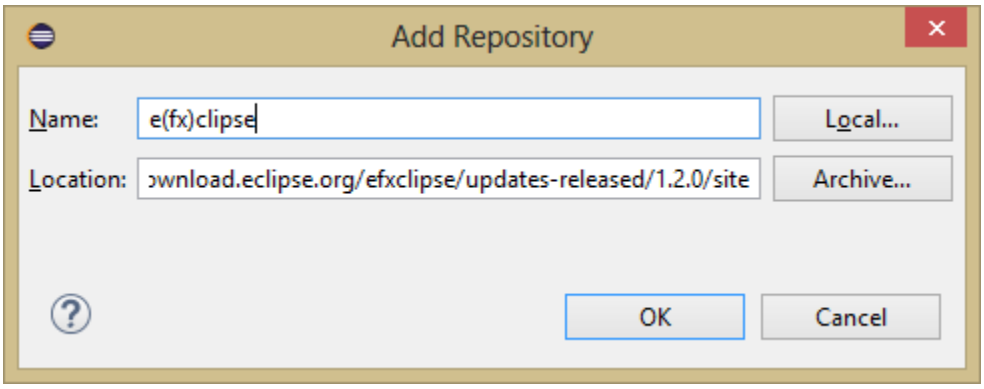
Note: Scene Builder can work with a slightly different JDK version number ***within Java 8*** and you can get away with a compiler warning. As a good practice, you should make sure your JDK is updated too.

2. Install e(fx)clipse.

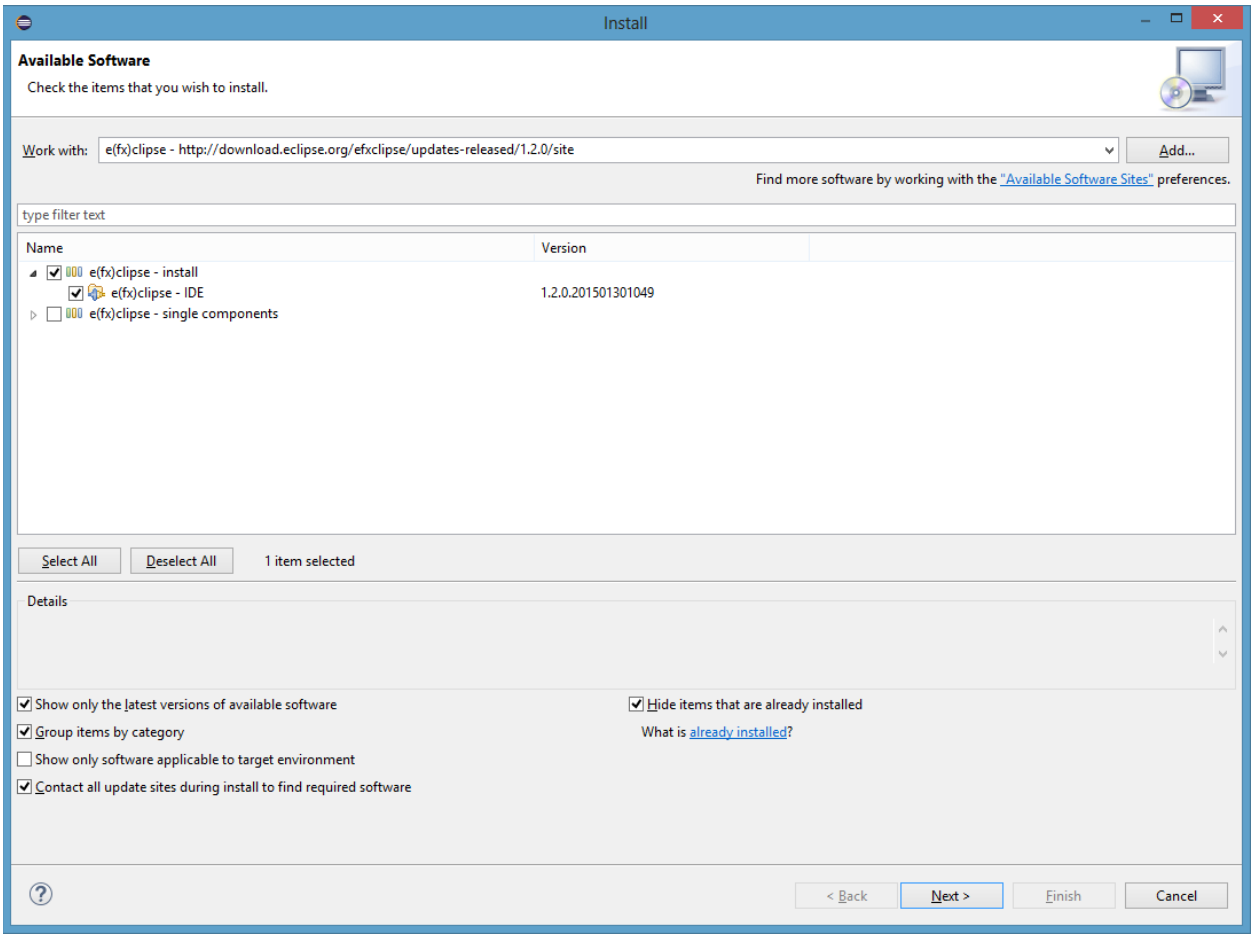
2.1 Go into the Help menu and click **Install New Software (see above)**.



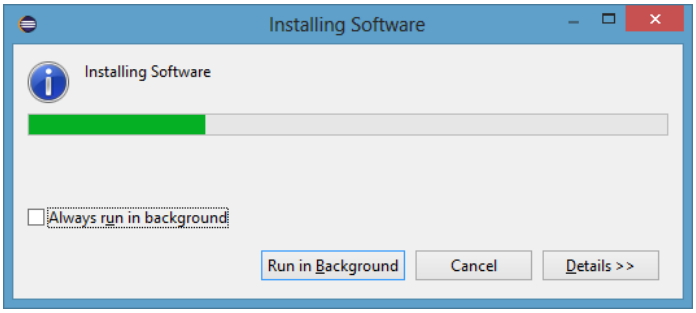
2.2. Within the dialog box for installing new software, Click the Add button and use the following URL as the location of the repository: http://download.eclipse.org/efxclipse/updates-released/1.2.0/site



If you get multiple options, select only YOUR eclipse version (Luna). In the picture below, there was only a standard Eclipse-related add-on available.



Let the installation proceed. Restart as necessary.



3. Install Scene Builder.

Oracle does NOT provide compiled binaries – only the source code. (Do NOT download OLD archived binaries, which belong to previous versions.) You can go Oracle's site to get the source for Screen Builder and compile it yourself OR you can download compiled binaries from third parties such as GluonHQ.com:

3.1 Download and install Scene Builder

FYI: When downloading binaries from this site, download managers did not work.

http://gluonhq.com/products/downloads/

OR

If you are feeling adventuruous, go to Oracle. It has source as well as documentation (you will want to read the documentation, which has related "tutorials").

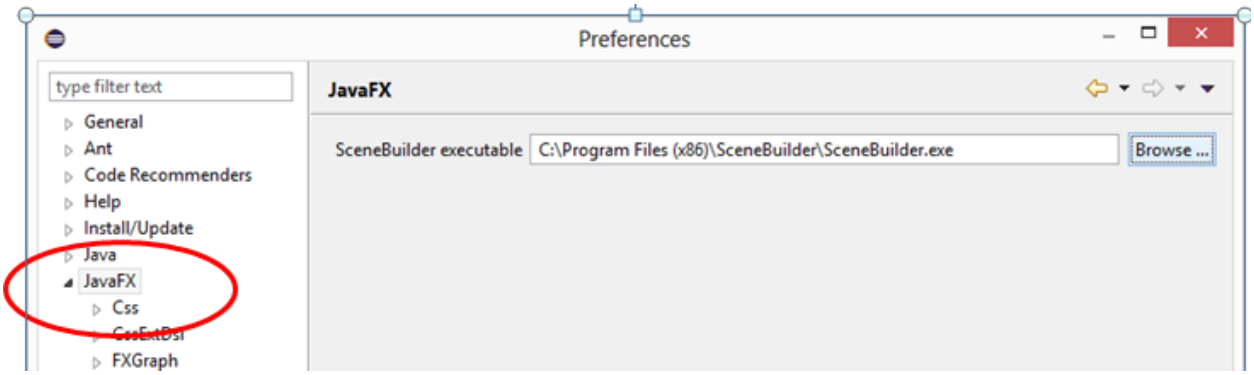
http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html

3.2 Configure SceneBuilder to work with Eclipse.

This will work only if you setup e(fx)clipse properly. *(Have you been restarting Eclipse?)*

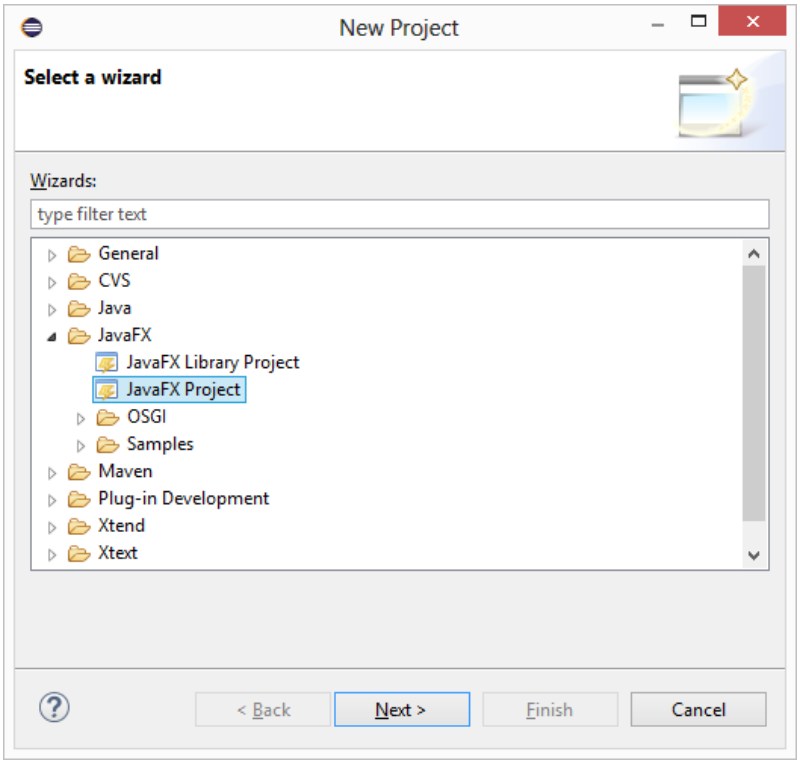
Go to Preferences and search for JavaFX.

Click Browse and enter the location of Scene Builder.

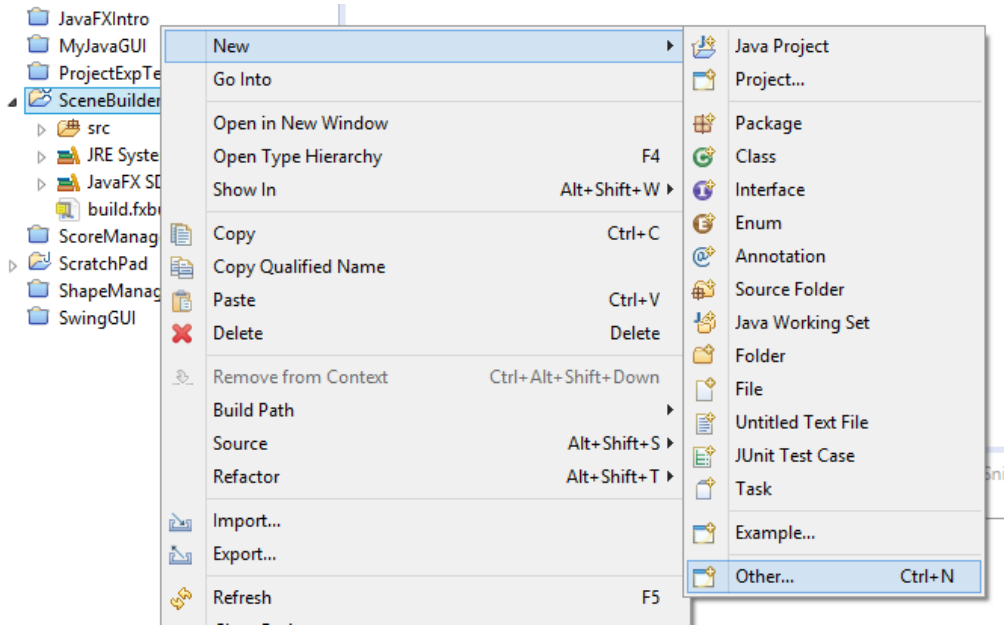


4. Build a JavaFX application using Scene Builder!

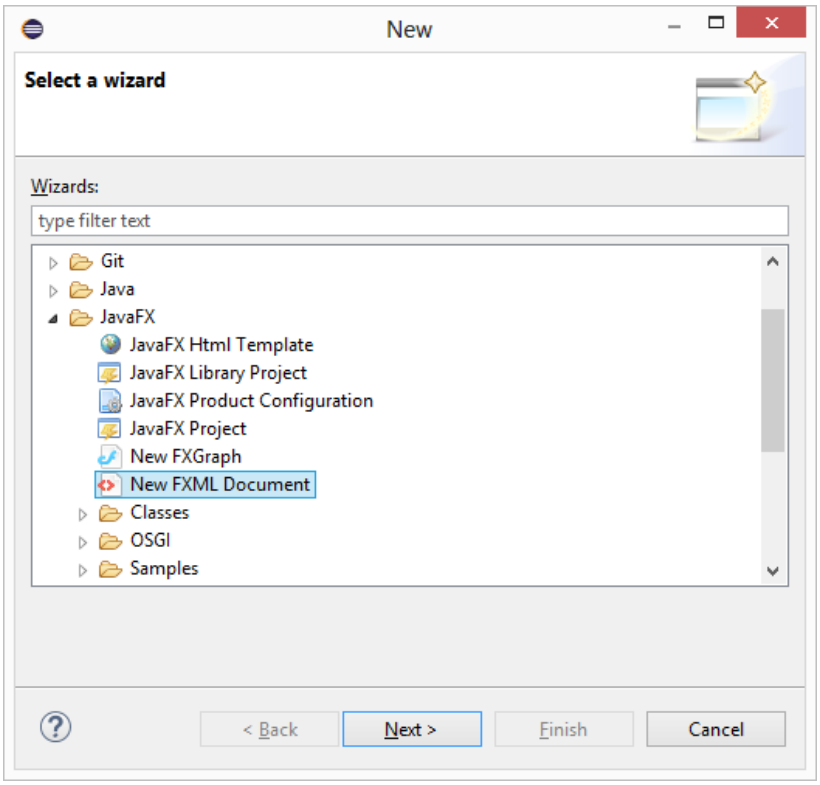
4.1 Start a new JavaFX project. (New option available due to e(fx)clipse). This time make sure you give your project a Package name, e.g. “yourlastname.CIS494.IC9”. “application” is specified by default.

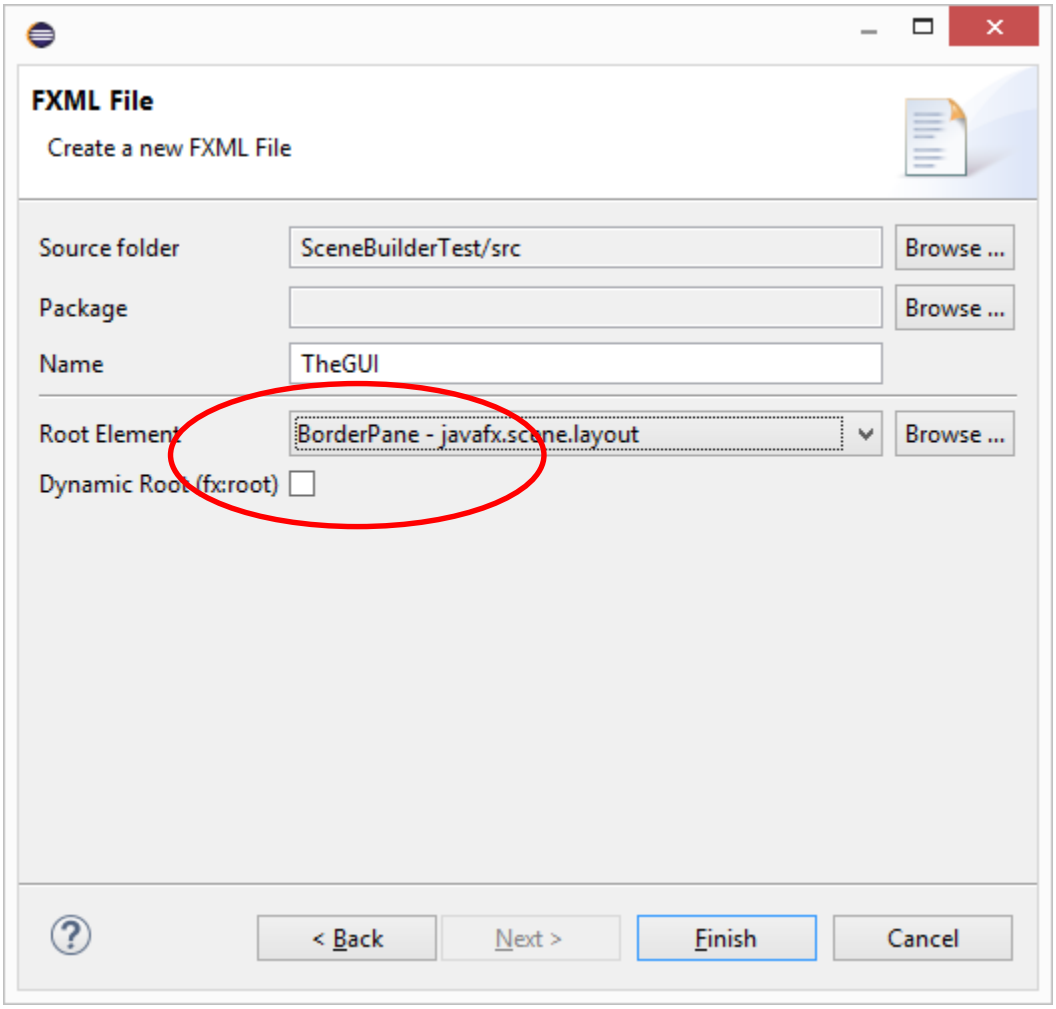


4.2 Add a new “Other” File to the project.

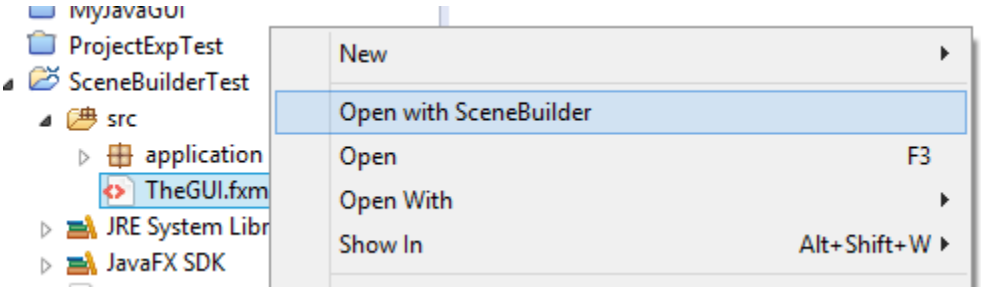


4.3 From within the JavaFX folder, select FXML document.





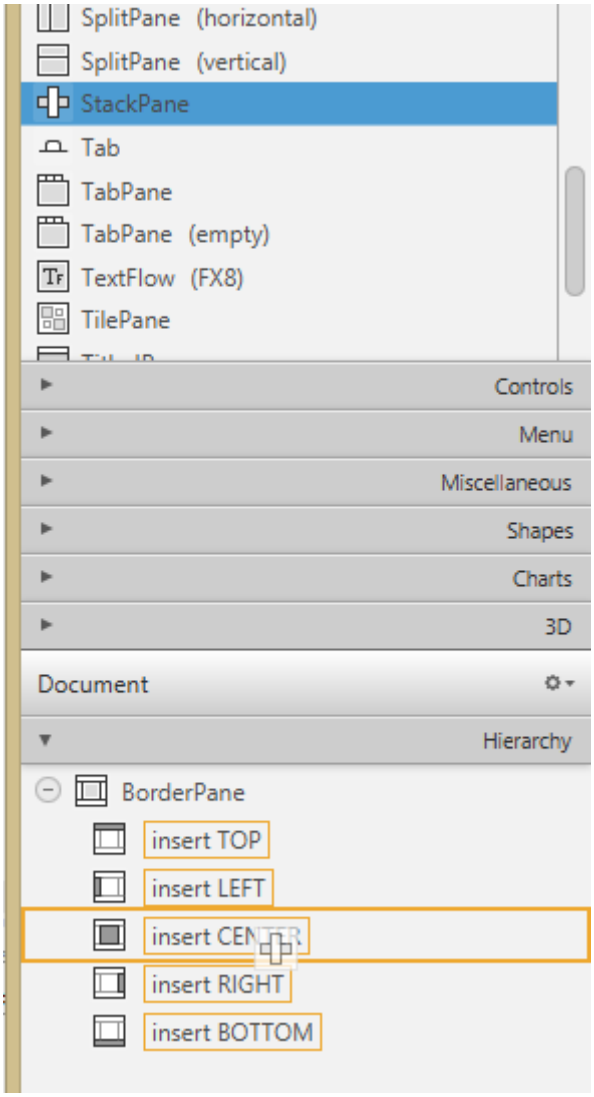
4.4 Right click the FXML document found in your SRC folder and open it with Scene Builder. (Make sure you have already configured the source location of the Scene Builder executable file).



4.5 Use drag and drop to create the following GUI. In the picture below, the root element is a **BorderPane**, which has inside in a **GridPane**. The following controls are inside the GridPane:

2 TextFields, **num1** and **num2**
1 Button, **okButton**
1 Label, **result**

Start with the root element of the BorderPane. Drag the other container/controls onto the region that you want.

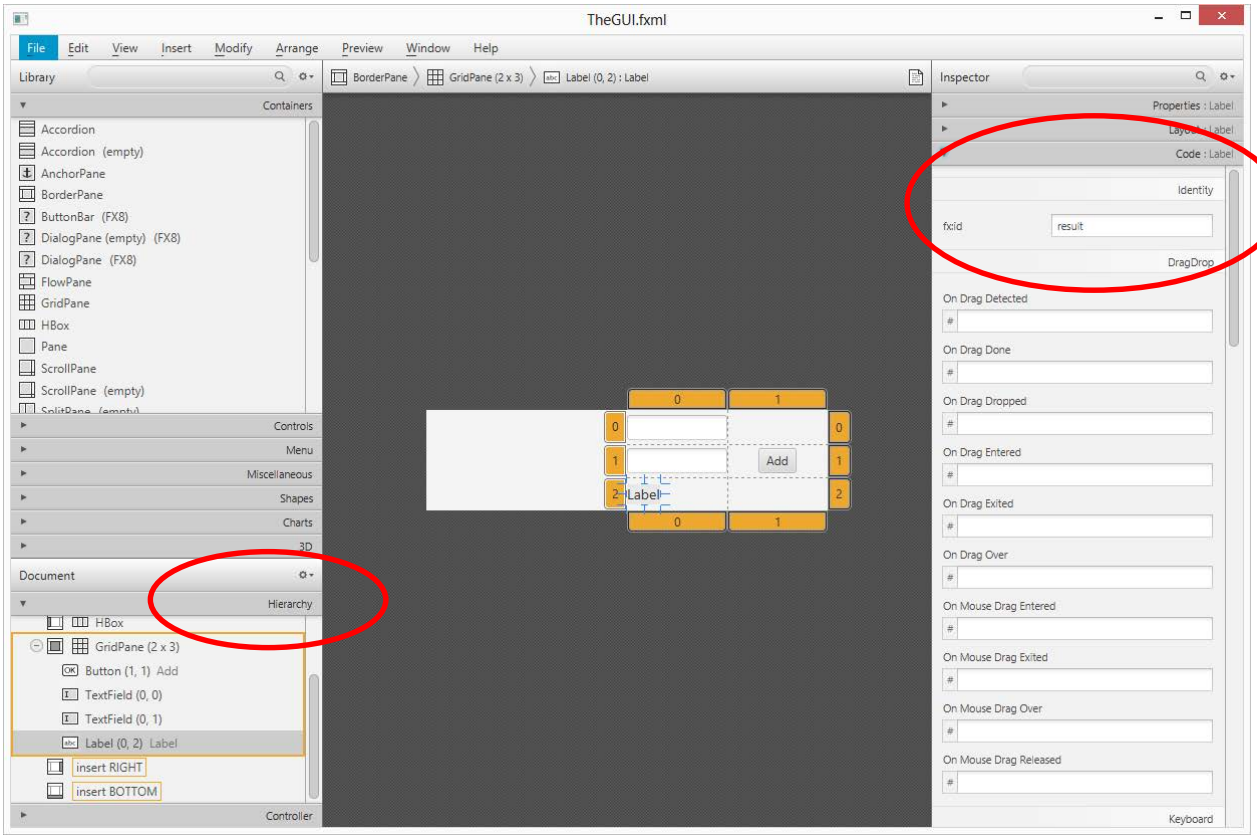


The picture above shows a StackPane being dragged and dropped onto the Center region of the root element BorderPane.

After you are creating your GUI design Make sure you set the Label text from Properties on the right side to a blank value. It is shown as “Label” here to make it visible.

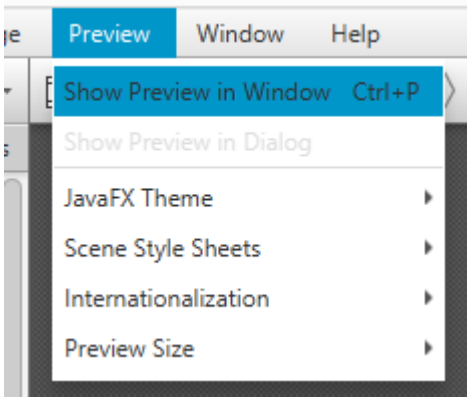
You can access all the Controls from the Hierarchy in the tree on the left. You can set control properties on the right.

Important: set the name of the controls in the fx:id property within **Code.** This JavaFX ID property will help link the GUI control in the XML file to the instance variables in the class file. There is another ID field available in Properties, which is the ID field of XML/HTML elements, which is different. Do not confuse the two.



4.6

Look at a preview to see what your output will look like.



5. Time to go back to Eclipse and hook the FXML document to the JavaFX project.

5.1 Look at your FXML by double-clicking it within Eclipse. Does it look like this?

```
<children>
   <Button fx:id="okButton" mnemonicParsing="false"
   <TextField fx:id="num1" />
   <TextField fx:id="num2" GridPane.rowIndex="1" />
   <Label fx:id="result" GridPane.rowIndex="2">
      <font>
         <Font size="14.0" />
      </font>
```

5.2 Within your start method, declare a new variable of type URL (package name: java.net.URL) and assign the FXML document location to the URL variable. Use the getResource method as shown below.

Note that the FXML document location needs to be accurate in relation to where the main .class file is. In the code below, "../" is used within the path name to indicate that the FXML file is found in the parent folder (one level above indicated by ".."). ***Check where your FXML document is being saved***. If it is being saved in the **bin/src** folders then you **do not** need "../".

```
URL location = getClass().getResource("../TheGUI.fxml");
BorderPane root = FXMLLoader.load(location);
```

Call the static Load method of the FXMLLoader class and supply it with your location. It will return the BorderPane it created based on the FXML document. Assign that to a variable "root" of type BorderPane. Cast it explicity to a BorderPane ***if the compiler complains*** (it is not done above).

Use the BorderPane "root" as the root of your Scene. Set your scene and show scene as you would normally. In essence, rather than creating all the control elements manually, the entire BorderPane has been created using XML generated by Scene Builder.

5.3 Run your application and show see if the stage shows as expected. In the image below, the Label is not shown since its text property was set to blank in Scene Builder.

6. Add Event Handling

This application does not currently respond to events. To make it respond the events, we have a little problem to solve: the controls are in the XML. How will we get access to them in the JavaFX project?

Answer: We will create another class file, which will have private instance variables. These private variable can automatically be mapped to the "ID" field we set in Scene Builder. We will then have a reference to the controls.

6. 1 Create a new class called Controller.

Assign a package name at the top: "package *packagename*;" Make sure this is the same package name as your main class.

6.2 Declare private instance variables okButton, num1, num2 and result. These are the same names used in the FXML document. **Do NOT** instantiate these variables.

However, the variables are private. There needs for Scene Builder to access them without the class needing to make the variables public: add "@FXML" (no quote) on the line above every instance variable. This will allow Scene Builder to find these variables even though they are private.

6.3 Create an okButtonHandler method with an argument of type ActionEvent (just like you have been creating event handlers – no change!)

- Add println statements, so you know that the event handler was executed.

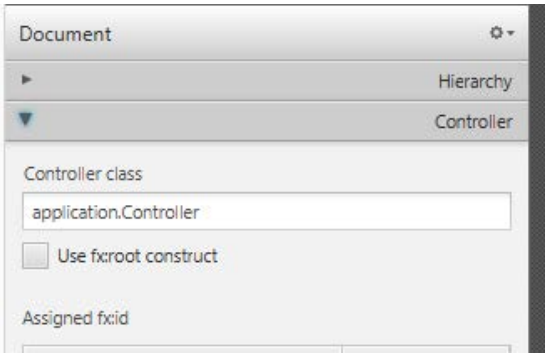
- Parse the text of num1 and num2 TextFields. Add the two numbers and put the answer inside the result Label. *You might want to do this after you know your event handler is being called properly.*

Again, make sure you prefix your method name with @FXML so that Scene Builder can find it.

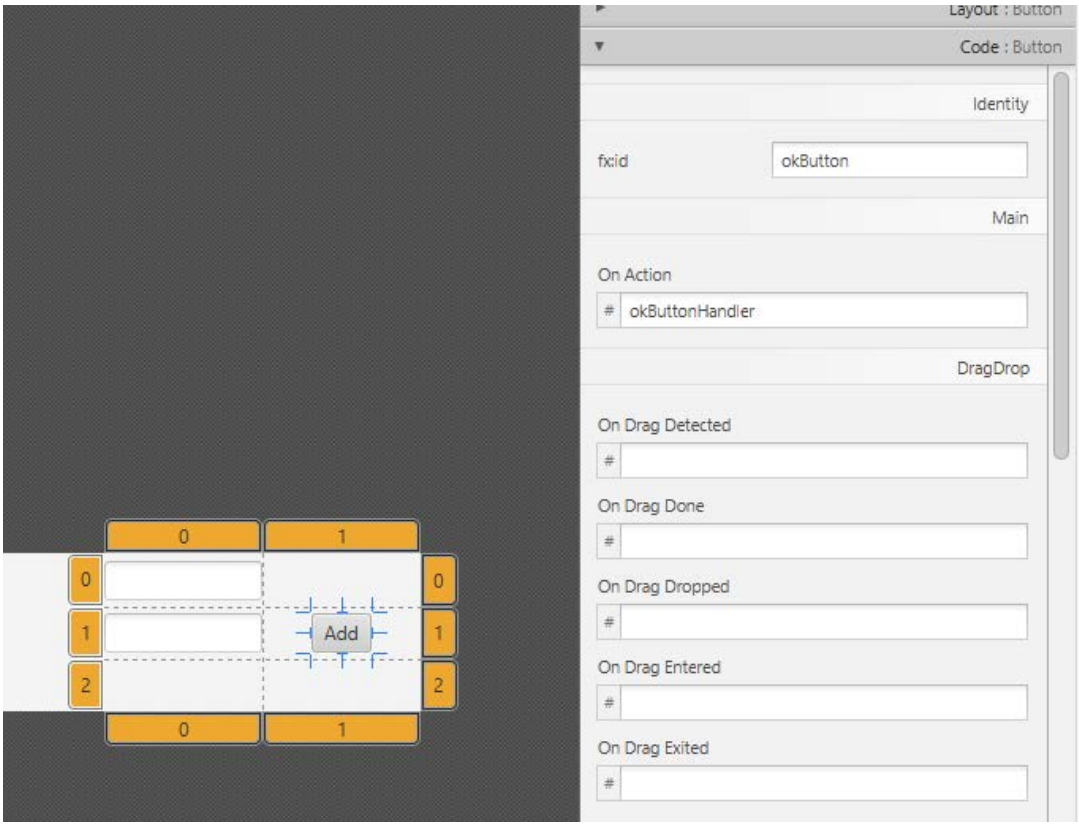
6.4 Create a private method called initialize() that returns void. This is the one of the first methods that will execute automatically. Put in a println here too. Prefix with @FXML.

6.5 Configure Scene Builder – go back to Scene Builder.

6.5.1 Set the name of the Controller class on the left hand side panel titled “Controller”. Make sure you spell out the full name including package name. In the example below the Controller class is found in the “application” package.



6.5.2 Click on the Button and within the Code panel on the right, set the name of the method you want called on the Action event. The ONLY text that will go in is the name of the method.



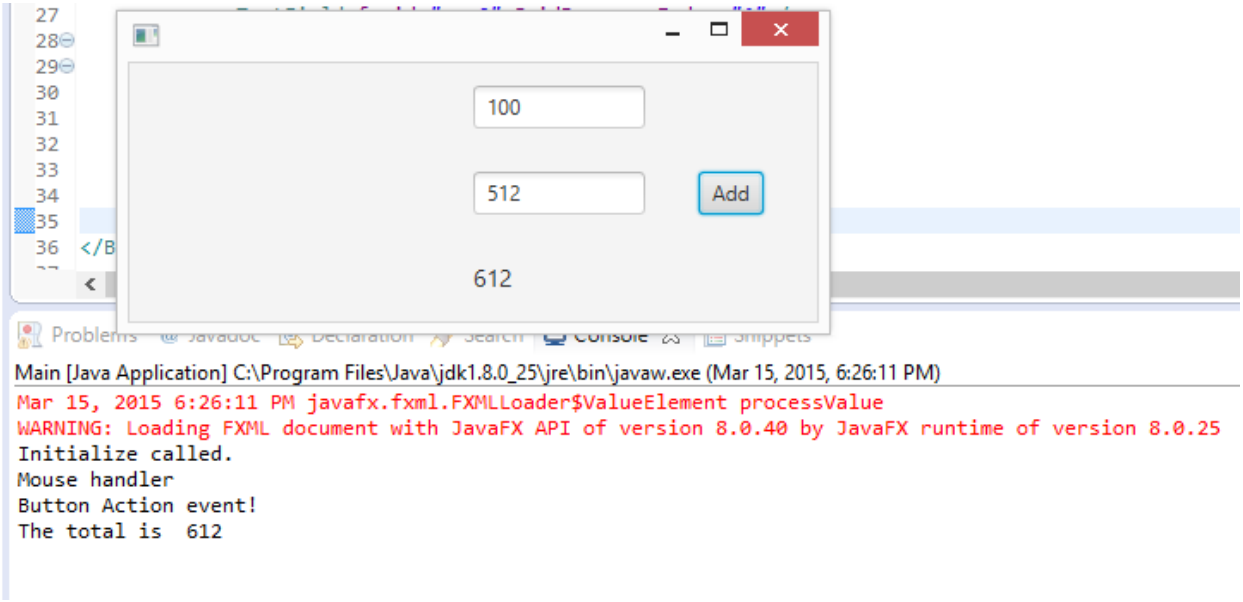
6.5.3 Save your FXML file. Go back to Eclipse and look up the FXML document. Does it show your updated event handling code?

```
<children>
   <Button fx:id="okButton" mnemonicParsing="false" onAction="#okButtonHandler"
   <TextField fx:id="num1" />
   <TextField fx:id="num2" GridPane.rowIndex="1" />
   <Label fx:id="result" GridPane.rowIndex="2">
      <font>
```

6.5.4 Run your application. Click the button. Watch the system console first (assuming you put in a println).

You should have now been able to call the method as an event handler.

Note: You should not have called the constructors for ANY controls, from BorderPane to TextField. Everything should have been instantiated by FXMLLoader. References to the controls will automatically be stored in your private instance variables in the Controller class. It is able to map the private variables based on the FX ID attribute and the specification of the Controller class.



Enhancements and Learning Challenges

1. The FXML document is limited to creating entire a GUI for just one window/stage. You can create multiple FXML documents, each document representing the GUI elements for one pane. Create two separate FXML documents and set their output to be stored in different panes. Then add those panes to different regions of a BorderPane.

2. Add Buttons/Fields to the different FXML documents and create an independent Controller class for each document. Make sure that you can respond to events from the different regions. Now if you needed to exchange data between the controls of different Controller classes, how would you do it?

Submitting Files

Submission should be made using a zip file that contains the entire Eclipse project folder. You will need to ***zip the entire project folder***. The folder will automatically contain the class source files as well as the compiled .class files.

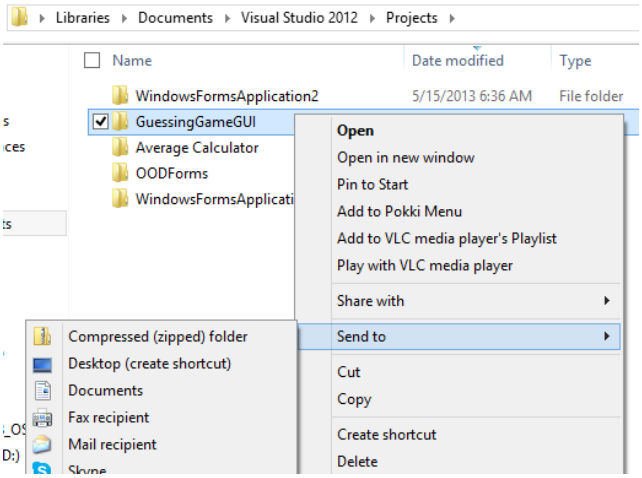
ZIP file should be named: **AX.zip or IC.zip**

> where X is the in-class assignment number, e.g., A1.zip is the submission file for Assignment 1 and IC1.zip is the submission file for InClass1.

Note: *The ZIP filename is independent of the Project name. Do not name your project A1.*

How to Properly ZIP and Submit Your Project Files:

- Go to the folder within {Eclipse Workspace}\
- ZIP the entire top-level folder for your project by right-clicking your project folder and selecting Send to | Compressed (zipped) folder.
- Finally, submit the ZIP file using the submission link on Blackboard by the due-date and time listed on the assignment. Upload the ZIP file.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Verify your files BEFORE and AFTER submission:

- Check for actual class files being present in the folder before you zip it.
- ***Ensure that you are not zipping a short-cut to the folder.***
- After zipping, check file size. A file size under 4K likely does not contain all the files.
- Unzip, extract all files, and verify you see actual files, not a solitary short-cut.
- Uncompress your zip file before submitting and verify that files are present.
- Make sure you have ***submitted*** your file and not just saved a draft on Blackboard. ***A blue clock indicates a submission in progress, i.e. a draft, not a submission. The draft is accessible only to you. You will get a ZERO if you only ever save a draft on Blackboard and never submit your files.***

- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.