How to Install OpenJDK, Apache NetBeans, and OpenJFX on Windows

John P. Baugh, Ph.D.

Professor and Chair, CIS, Oakland Community College – OR
Lecturer II, CIS, University of Michigan – Dearborn

Document history

Version History	Notes	Date of Updated Document
1.0	Initial document	10/5/2019

Current technology versions used in this document

Software/Library	Version
OpenJDK	13
NetBeans	11.1
OpenJFX (JavaFX)	13
Scene Builder	11.0.0

1. Introduction, background, and preliminaries

This document covers a fresh installation of the OpenJDK version 13, NetBeans 11.1, and OpenJFX (JavaFX). It assumes you already have the Java Runtime installed (JRE.) If you do not have Java installed on your system, NetBeans will not launch, and neither will any programs you write, or any other applications written in Java.

If you do not have Java installed, please refer to this document:
https://java.com/en/download/help/download_options.xml

There has been a lot of upheaval and confusion in the Java world as of the writing of this document. Oracle has made some changes to how it licenses certain older versions (and newer versions) of the **JDK**, so an initiative was launched and open versions of **Java Development Kit (JDK)** have been released under the name **OpenJDK**.

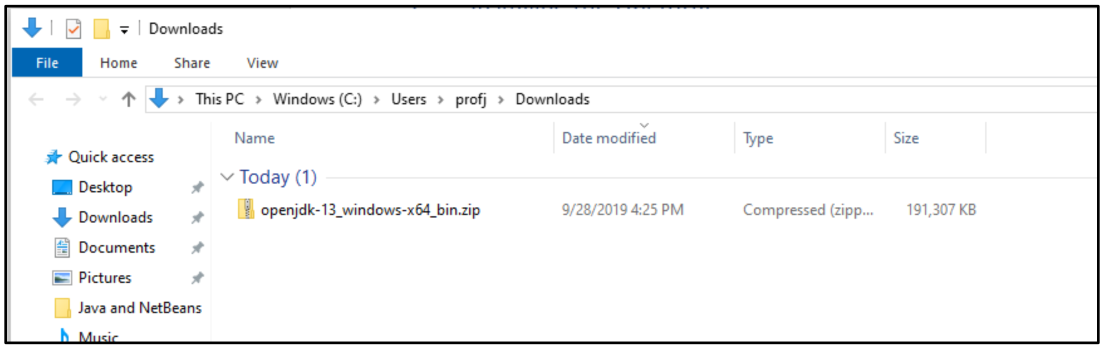
Also, **NetBeans** has fallen under the control of the **Apache Foundation**, and now the newer versions will be known as **Apache NetBeans**. The older versions of NetBeans (e.g., 8.2) will not remain appropriate for developers looking to use Java as it always has been in the past – an open language and open framework. The newest incarnations of Apache NetBeans have included multiple build environment options, such as Ant, Gradle, and Maven. This can all be very confusing to those coming from previous versions of the old NetBeans IDE, where things "just worked".

Finally, **JavaFX**, the framework used in Java to (eventually) supersede Swing as the GUI library of choice, has fallen under the control of yet another organization, **Gluon**, which creates the **Scene Builder** application for rapid GUI development with JavaFX. JavaFX is an excellent library, but NetBeans (and other IDEs) have not fully integrated it into their application types. NetBeans displays an option to create a JavaFX application, but then informs you that you can't create one using that technique. Eventually, this will hopefully be sorted out, but for now, roll up your sleeves and get ready to do a little bit of work to get everything set up.

2. Installing the OpenJDK

For installing the OpenJDK, there really isn't all that much to it. You download a zip file to a preferably organized and thoughtful location, unzip (extract) the contents, and then you just use it when Apache NetBeans needs to know where the JDK lives.

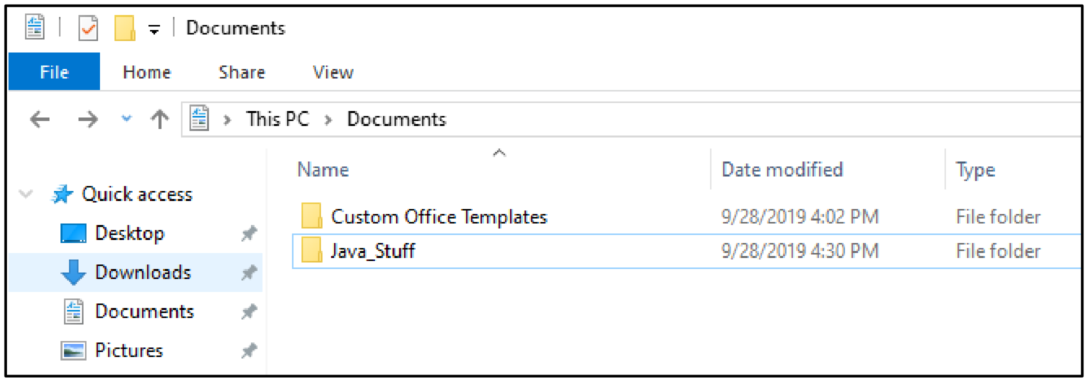
1. In order to download the OpenJDK, version 13, go to https://jdk.java.net/13/
2. **Download the appropriate compressed format** for your platform (for Windows, this is the Windows zip file)



Name	Date modified	Type	Size
Today (1)			
openjdk-13_windows-x64_bin.zip	9/28/2019 4:25 PM	Compressed (zipp...	191,307 KB

Figure 1 OpenJDK downloaded

3. Since I'm not a total slob, I will create a nice little home for all this Java related stuff. I created a folder under Documents called **Java_Stuff**



Name	Date modified	Type
Custom Office Templates	9/28/2019 4:02 PM	File folder
Java_Stuff	9/28/2019 4:30 PM	File folder

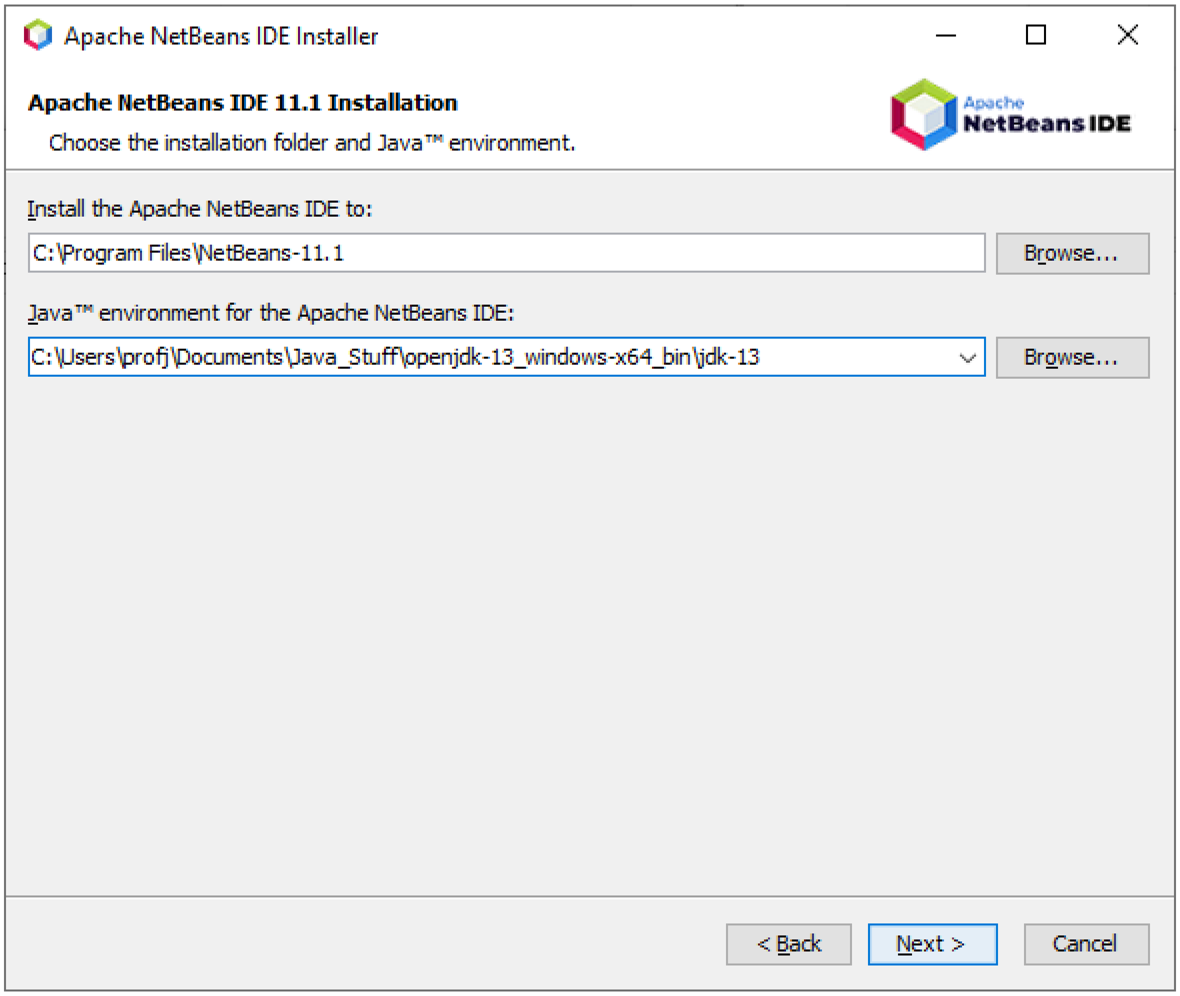
Figure 2 Java_Stuff folder created under Documents

4. **Move the zip file** containing the OpenJDK to your Java_Stuff folder (or whatever you named it)
5. **Unzip (extract) the zip file** (Right-click, Extract All) so that it creates a folder with the same name in the Java_Stuff folder
6. At this point, you can **delete the zip file** (do **not** delete the folder that you extracted from the zip file)

3. Installing Apache NetBeans

Installing Apache NetBeans isn't particularly difficult. As of this writing, there is now an installer provided for Apache NetBeans 11.1. Apache NetBeans just needs to know where the JDK lives (the OpenJDK we installed in the previous section), and if you have the JDK installed (and of course, Java itself, as discussed in the introduction) then you should be good to go.

1. **Download Apache NetBeans 11.1 installer** from https://www.apache.org/dyn/closer.cgi/netbeans/netbeans/11.1/Apache-NetBeans-11.1-bin-windows-x64.exe (for Windows)
   a. Make sure to select a "mirror site" near you, which will make the download quicker
   b. For example, since I live in southeast Michigan, I picked the HTTP mirror site at Wayne State University (http://ftp.wayne.edu/apache/netbeans/netbeans/11.1/Apache-NetBeans-11.1-bin-windows-x64.exe)
2. **Run the installer** and make sure you select the correct location of the OpenJDK that you installed earlier


Apache NetBeans IDE Installer

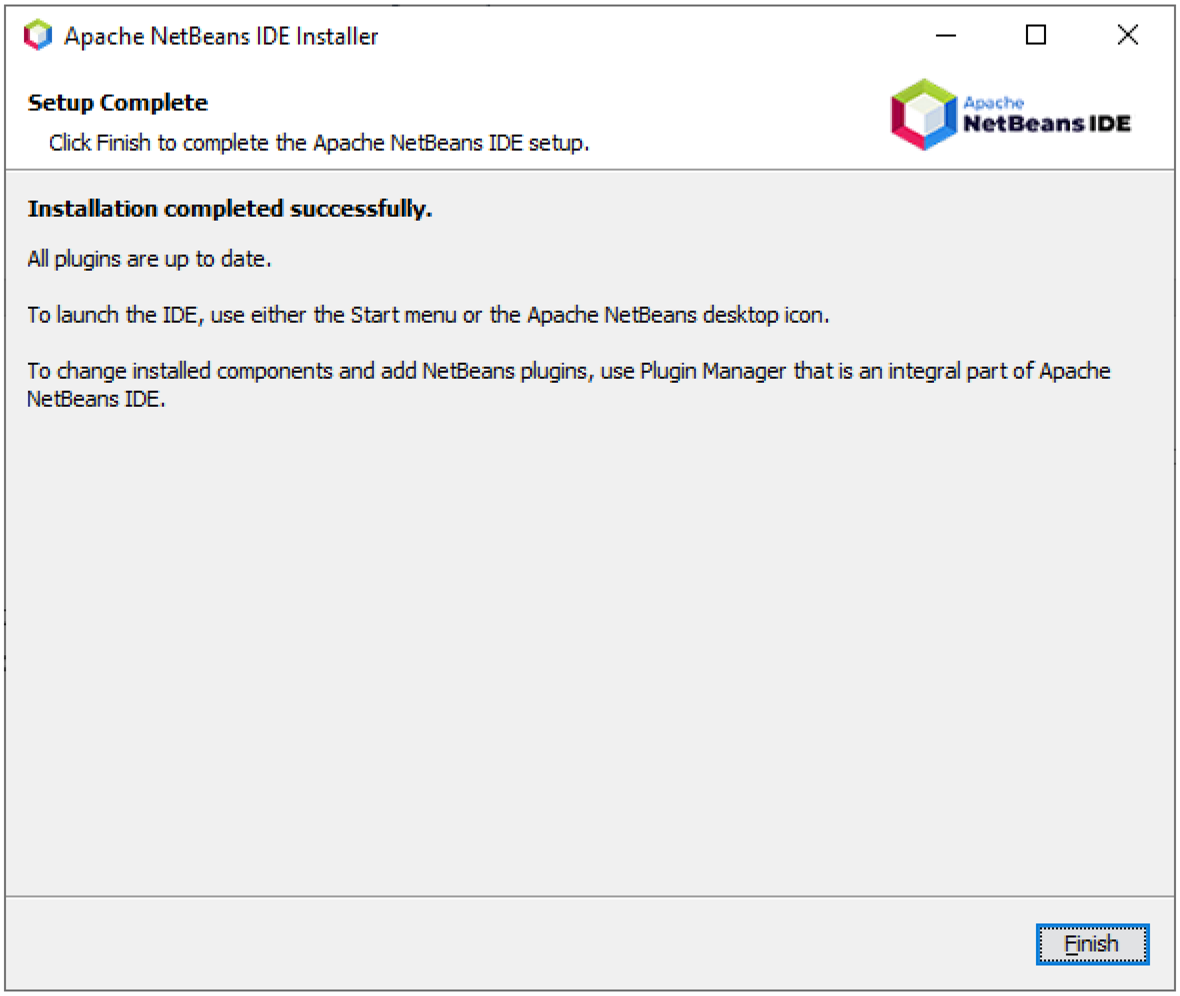
Apache NetBeans IDE 11.1 Installation
Choose the installation folder and Java™ environment.

Install the Apache NetBeans IDE to:
C:\Program Files\NetBeans-11.1

Java™ environment for the Apache NetBeans IDE:
C:\Users\profj\Documents\Java_Stuff\openjdk-13_windows-x64_bin\jdk-13

Figure 3 Apache NetBeans installer

3. Click **Next**
4. If prompted to check for updates, answer in the affirmative by clicking **Next** again
5. You will most likely get a confirmation dialog on the installer, so you can click **Finish**



Setup Complete

Click Finish to complete the Apache NetBeans IDE setup.

Installation completed successfully.

All plugins are up to date.

To launch the IDE, use either the Start menu or the Apache NetBeans desktop icon.

To change installed components and add NetBeans plugins, use Plugin Manager that is an integral part of Apache NetBeans IDE.

Figure 4 Installation completed successfully

4. Launching Apache NetBeans to test it out

Whether you're satisfied with the primary Java installation or you want to also install JavaFX, you should take the time to test out the basic installation and make sure everything looks good before proceeding. For those that aren't interested in JavaFX GUI development, you can go through this section to verify your installation and you're good to go! For those that want to install JavaFX, go through this section and then proceed to the next section for instructions on getting JavaFX set up properly.

1. Using the Windows search feature, or any shortcuts you may have created, launch Apache NetBeans
2. You may dismiss the Welcome screen / Start screen on Apache NetBeans, and go straight to creating a new project
3. Go to **File → New Project**

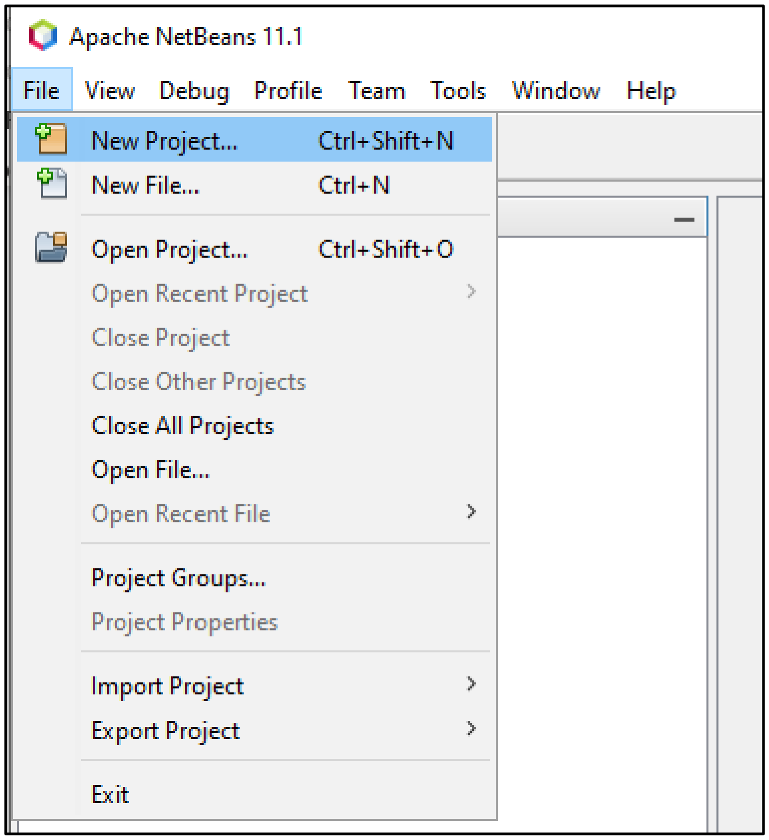
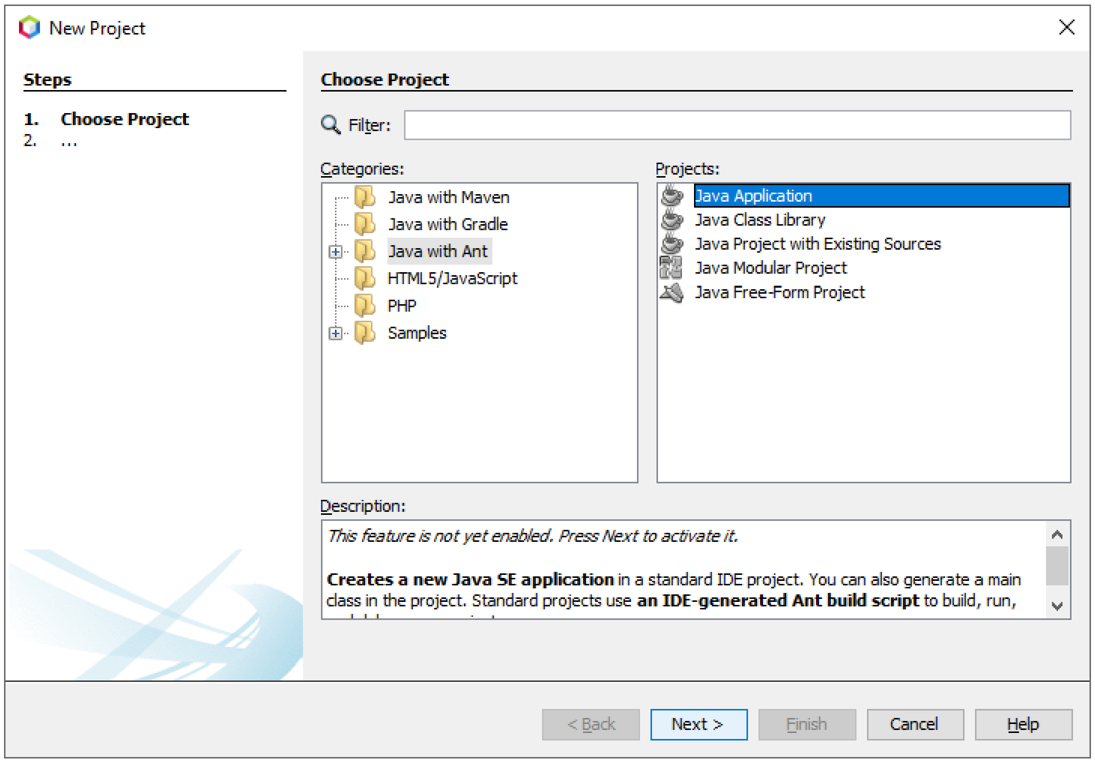


Figure 5 Creating a new project in Apache NetBeans

4. I recommend starting with using Ant, so go to **Java with Ant** and create a **Java Application** to do your initial test

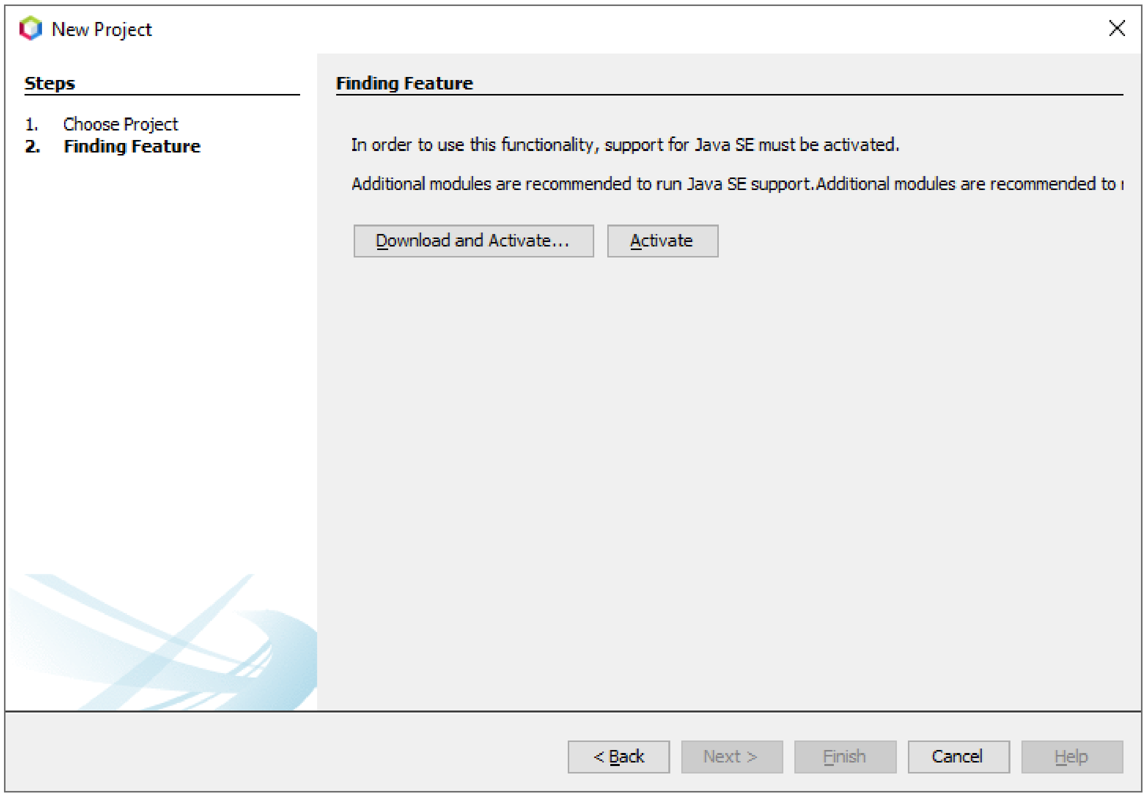


This feature is not yet enabled. Press Next to activate it.

Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run,

Figure 6 Creating a Java Application with Ant

5. Click **Next**
6. If you receive the following screen, select **Download and Activate**

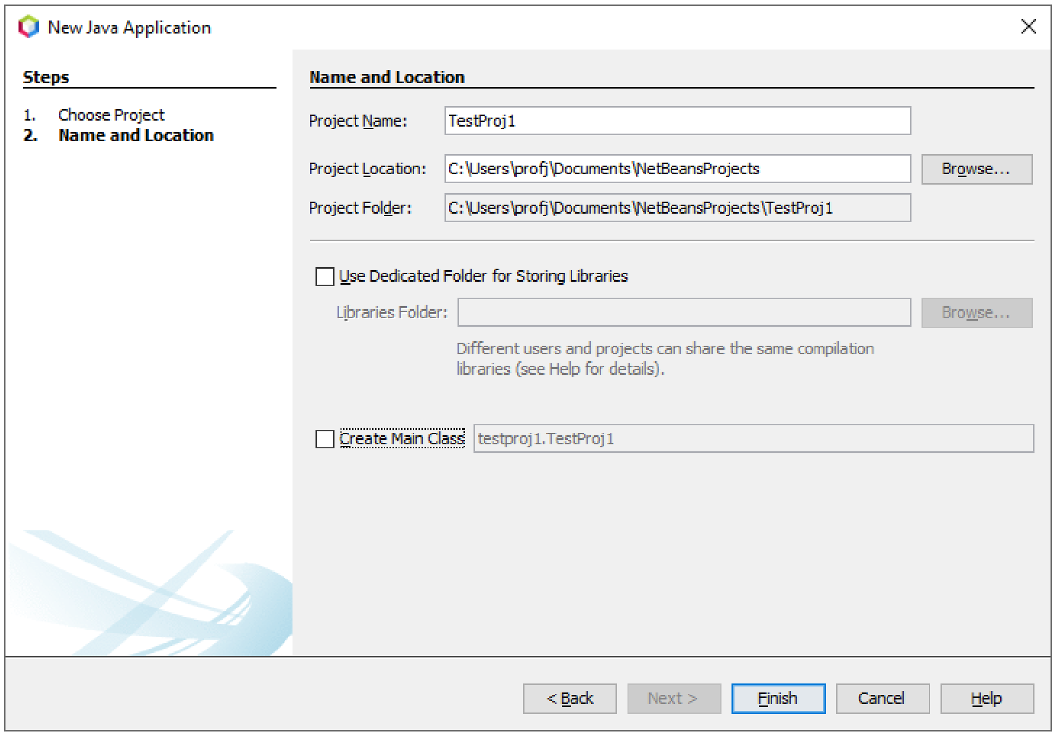


In order to use this functionality, support for Java SE must be activated.

Additional modules are recommended to run Java SE support.Additional modules are recommended to r

Figure 7 Download and Activate

7. If you are prompted to install some plugins, agree to the terms of service, and continue the installation/activation
8. Once that is done, it should say something like "Activating Java SE", and then you will see the project settings information for creating your test project


New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: TestProj1

Project Location: C:\Users\profj\Documents\NetBeansProjects

Project Folder: C:\Users\profj\Documents\NetBeansProjects\TestProj1

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class testproj1.TestProj1

Figure 8 Project creation with Ant

9. I called my project **TestProj1** and I also deselected "Create Main Class", but this part is of course optional
10. Click **Finish** once you are happy with the name, location, and other settings
11. In the Project navigation, **right-click the package** you want to add your first Java source file to. Although against recommendation, I just right-clicked default package and selected **Java Class…**

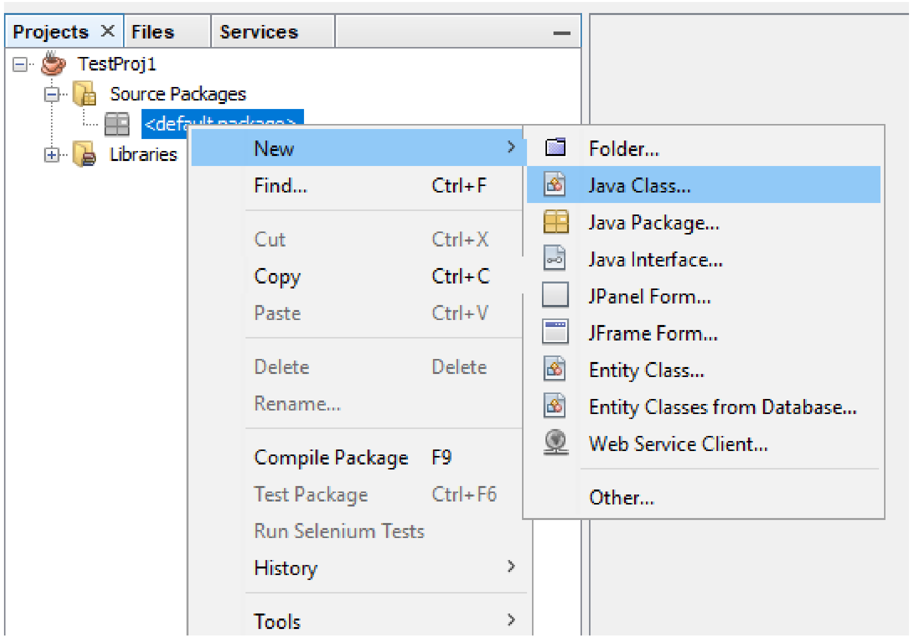
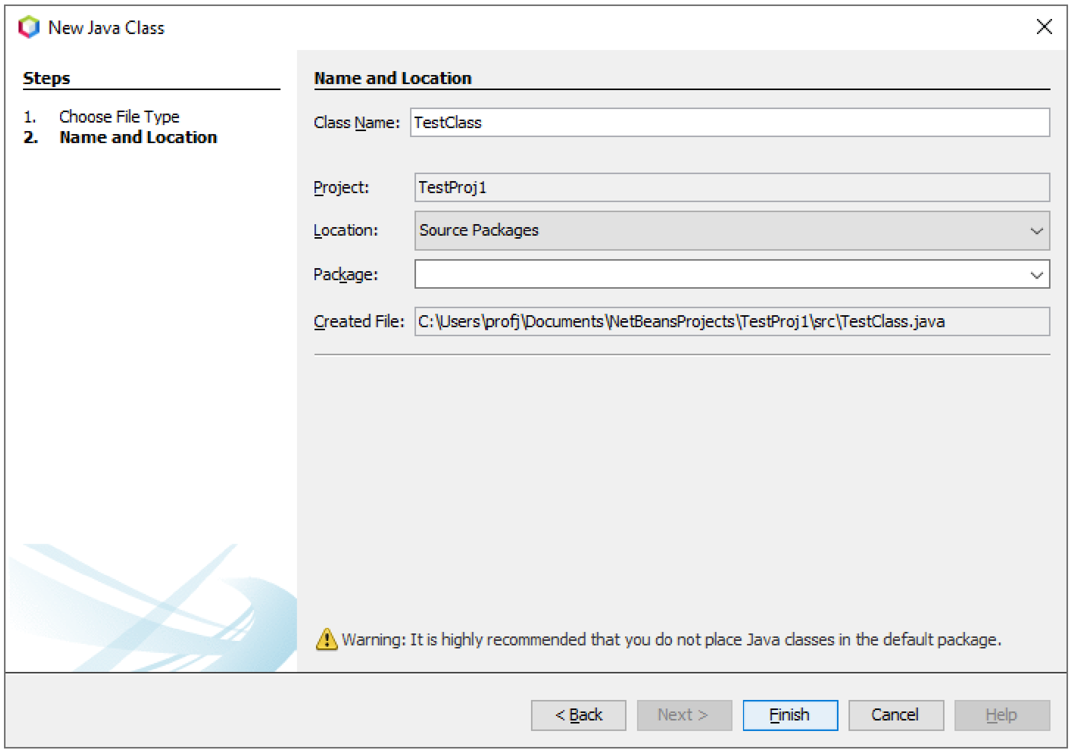


Figure 9 Creating a Java class

12. I named my class TestClass and didn't specify a new package



New Java Class

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: TestClass

Project: TestProj1

Location: Source Packages

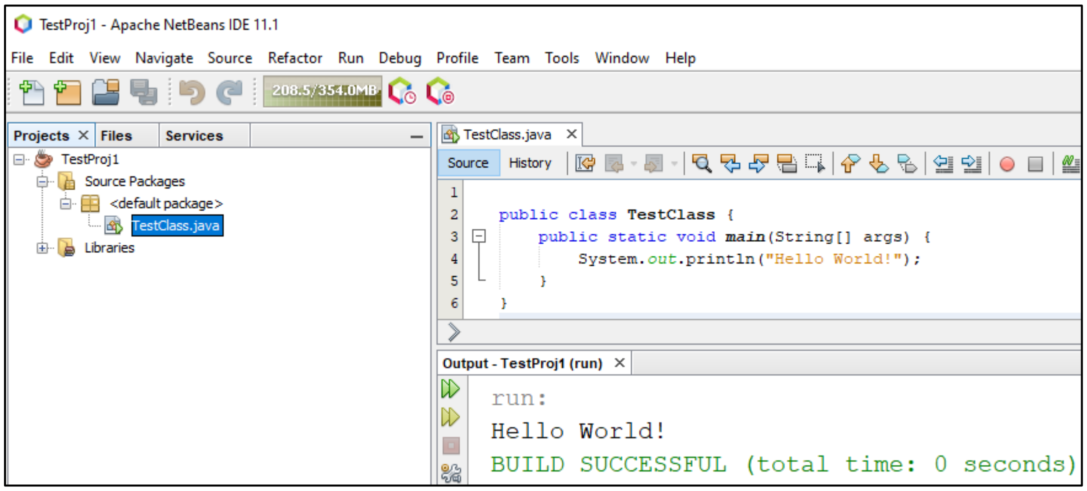
Package:

Created File: C:\Users\profj\Documents\NetBeansProjects\TestProj1\src\TestClass.java

Warning: It is highly recommended that you do not place Java classes in the default package.

Figure 10 Creation of a new class

13. Click **Finish**
14. I filled in the code for a simple "Hello World" type application, and then **right-clicked** the source file (TestClass1.java) in Project navigator and selected **Run**
    a. There are of course many other ways to run the application, but this is what I use often
15. The output should be verified, and you're all set!



TestProj1 - Apache NetBeans IDE 11.1

```

public class TestClass {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

```
run:
Hello World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

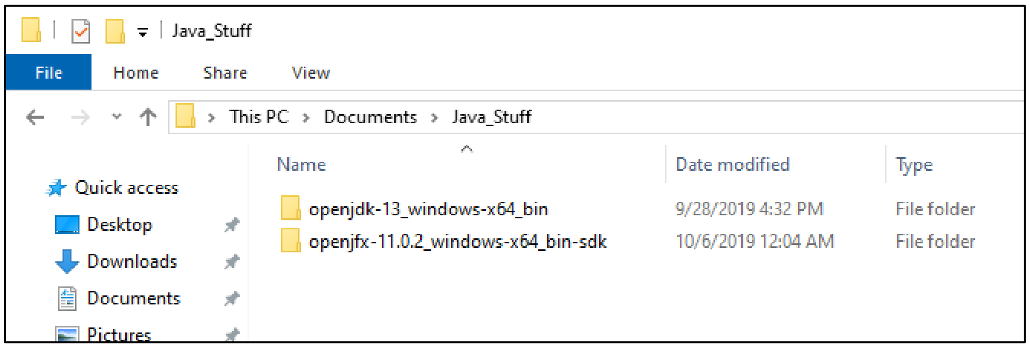
Figure 11 Simple Hello World application tested

5. Installing JavaFX with Apache NetBeans, and Scene Builder

Installing JavaFX, as of this writing, was the most challenging to get working properly. Although websites such as https://openjfx.io/openjfx-docs/#install-javafx in the NetBeans Non-Modular IDE instructions section were absolutely crucial to not pulling my hair out, I also discovered that the instructions were lacking a bit, at least for the Windows platform (as of this writing). The documentation on this site may be updated and corrected to include new requirements or new behaviors of current or future installers, but again, as of this writing, the instructions were insufficient.

5.1 Downloading and setting up OpenJFX

1. **Download OpenJFX** (Open JavaFX) Public Version labeled “JavaFX Windows SDK” at https://gluonhq.com/products/javafx/
2. **Move the zip file** to the Java_Stuff folder (or wherever you want to “install” it)
3. **Unzip/extract** its contents as we did with the OpenJDK earlier, deleting the zip file when you’re done



Name	Date modified	Type
openjdk-13_windows-x64_bin	9/28/2019 4:32 PM	File folder
openjfx-11.0.2_windows-x64_bin-sdk	10/6/2019 12:04 AM	File folder

Figure 12 JavaFX SDK installed into my Java_Stuff folder

4. Now, **launch** Apache NetBeans
5. Go to **File→New Project** as usual
   a. Although tempting, selecting “JavaFX” under Java with Ant and then trying to create a JavaFX project that way will only bring you sadness and disappointment
   b. Ant is not yet (as of this writing) equipped to create JavaFX projects
6. **Select** the regular **Java with Ant → Java Application**
7. Click **Next**
8. Give it a name, such as **HelloFXTest1** and click finish
9. Before going further, we need to create a globally accessible Library to add to any regular project to make it JavaFX-capable
10. Go to **Tools → Libraries**

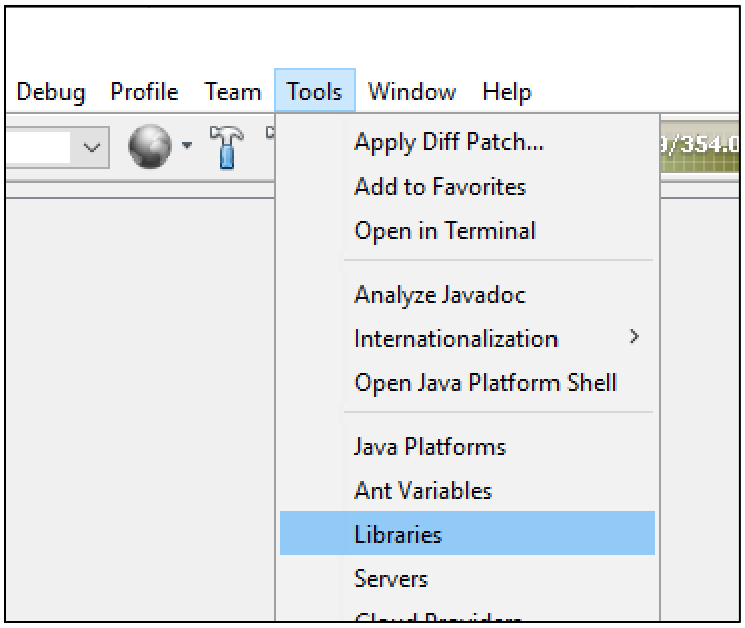
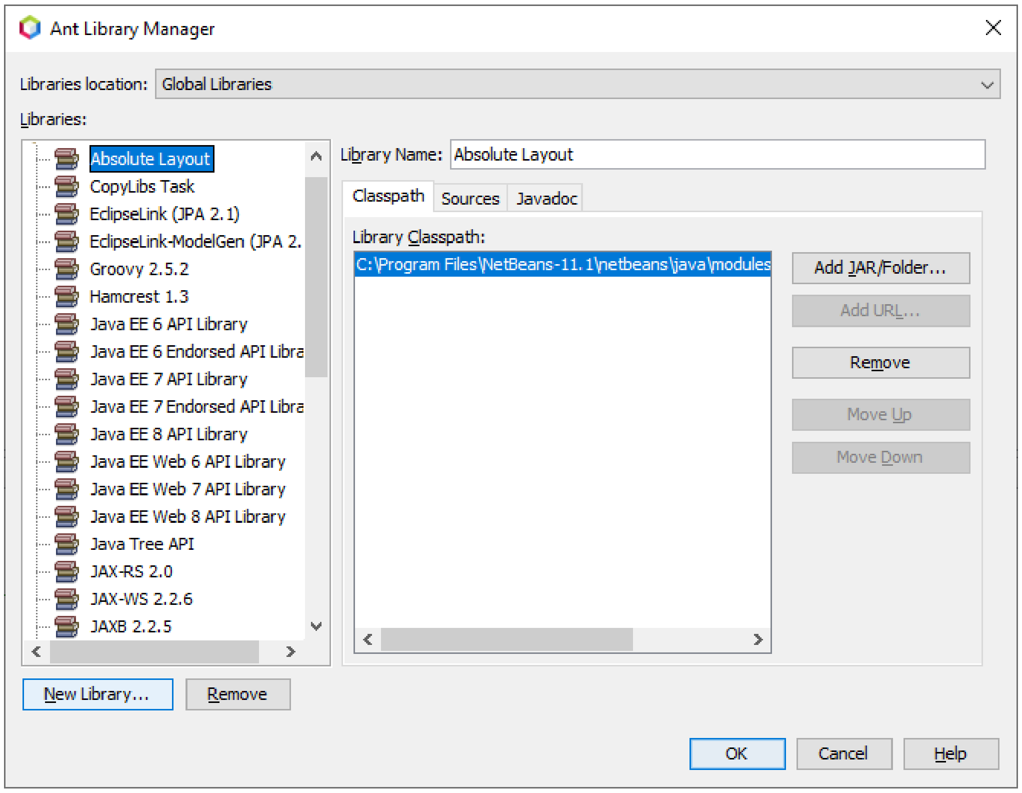


Figure 13 Creating a new library in Apache NetBeans

11. Select the **New Library** button from within the Ant Library Manager
    a. Make sure the "Libraries location" is set to **Global Libraries**



Libraries location: Global Libraries

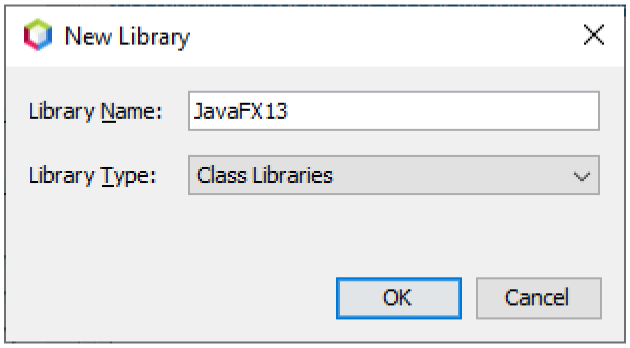
- Absolute Layout
- CopyLibs Task
- EclipseLink (JPA 2.1)
- EclipseLink-ModelGen (JPA 2.
- Groovy 2.5.2
- Hamcrest 1.3
- Java EE 6 API Library
- Java EE 6 Endorsed API Libra
- Java EE 7 API Library
- Java EE 7 Endorsed API Libra
- Java EE 8 API Library
- Java EE Web 6 API Library
- Java EE Web 7 API Library
- Java EE Web 8 API Library
- Java Tree API
- JAX-RS 2.0
- JAX-WS 2.2.6
- JAXB 2.2.5

Library Name: Absolute Layout

C:\Program Files\NetBeans-11.1\netbeans\java\modules

Figure 14 Creating a new library in the Ant Library Manager

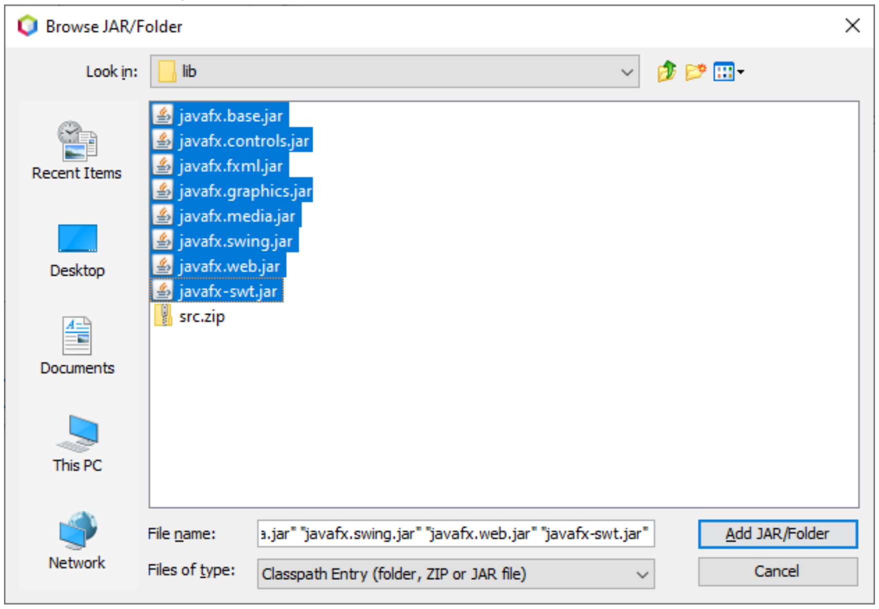
12. Set the library name as something useful like **JavaFX13**
    a. Technically the most recent version of OpenJFX is JavaFX 11, but I'm using it with OpenJDK 13, so I named it JavaFX13 so I remember this is the one to use with OpenJDK 13



Library Name: JavaFX13
Library Type: Class Libraries

Figure 15 Creating the JavaFX13 library

13. Hit OK
14. Now, in the Ant Library Manager, with JavaFX13 library selected, you have to add what you want to be a part of this library – that is in our case – JavaFX jar files
15. Click the **Add JAR/Folder** option
16. Navigate to where you have JavaFX installed
17. Descend into the folder contents until you see the lib folder, and then go into it
18. Inside the lib folder, you should select all the **.jar** files
    a. **DO NOT select the src.zip file**



Look in: lib

javafx.base.jar
javafx.controls.jar
javafx.fxml.jar
javafx.graphics.jar
javafx.media.jar
javafx.swing.jar
javafx.web.jar
javafx-swt.jar
src.zip

File name: a.jar" "javafx.swing.jar" "javafx.web.jar" "javafx-swt.jar"
Files of type: Classpath Entry (folder, ZIP or JAR file)

Figure 16 The JavaFX jar files

19. Click **Add JAR/Folder**
20. When you're done with this, hit **OK**

21. You have now created the global JavaFX library – this is something you shouldn't have to do very often, hopefully, only once and then you'll be set to use it in the projects you want to enable with JavaFX
22. From within the Apache NetBeans project, right-click the project and go to **Properties** at the very bottom of the context menu
23. Under the **Libraries** category, under the **Compile** tab, hit the **+** symbol next to **Classpath**

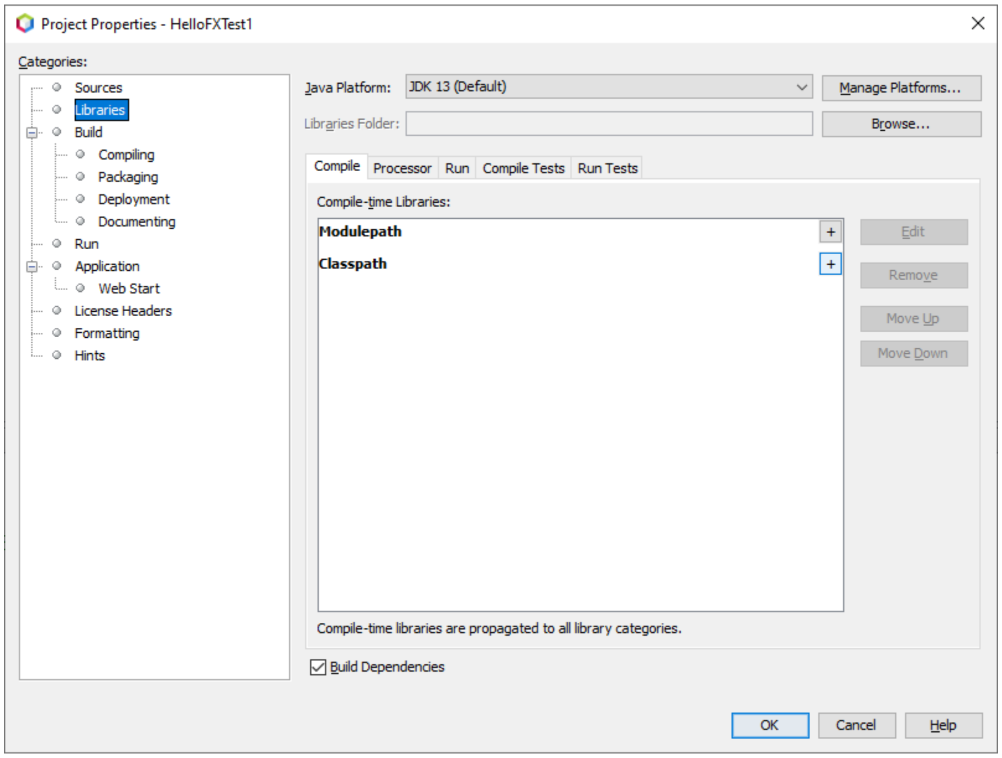


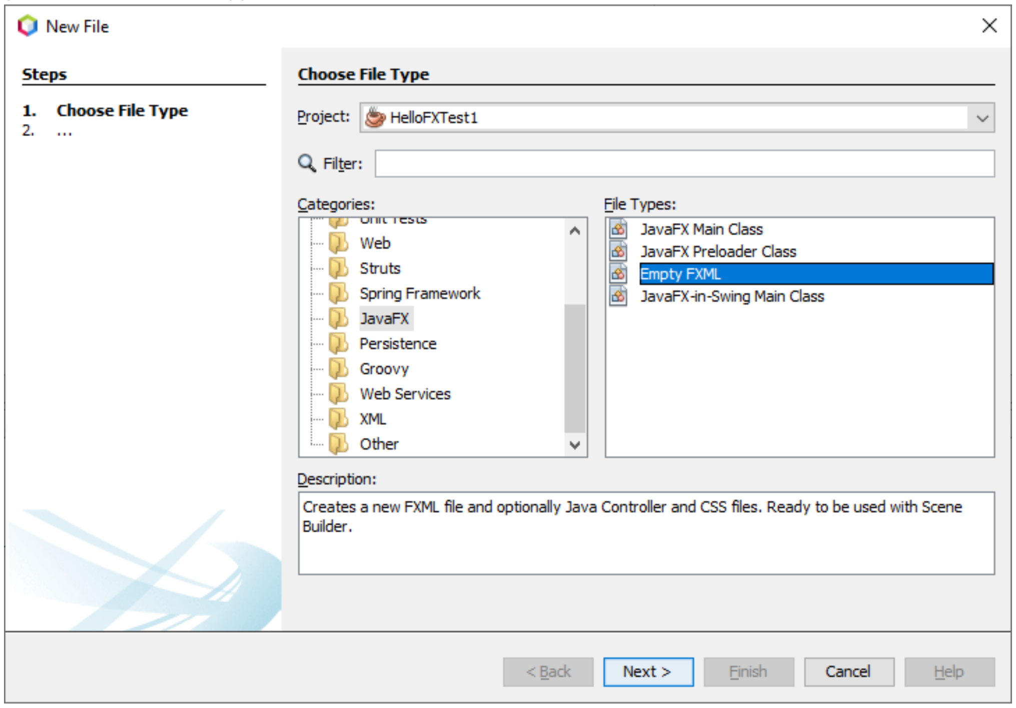
Figure 17 Adding a library to the library classpath

24. Select **Add Library**
25. Choose the **JavaFX13** (or whatever you named it) library from the list
26. Click the **Add Library** button on that screen as well
27. Hit **OK**

5.2 Adding JavaFX code (using FXML)

Note: Not everyone uses FXML, but I like to use it with Scene Builder, which we will install and use shortly. I think FXML provides a good separation of concerns and allows for stronger MVC-style architectural pattern for JavaFX projects as opposed to creating and launching all controller/GUI objects directly from Java coding.

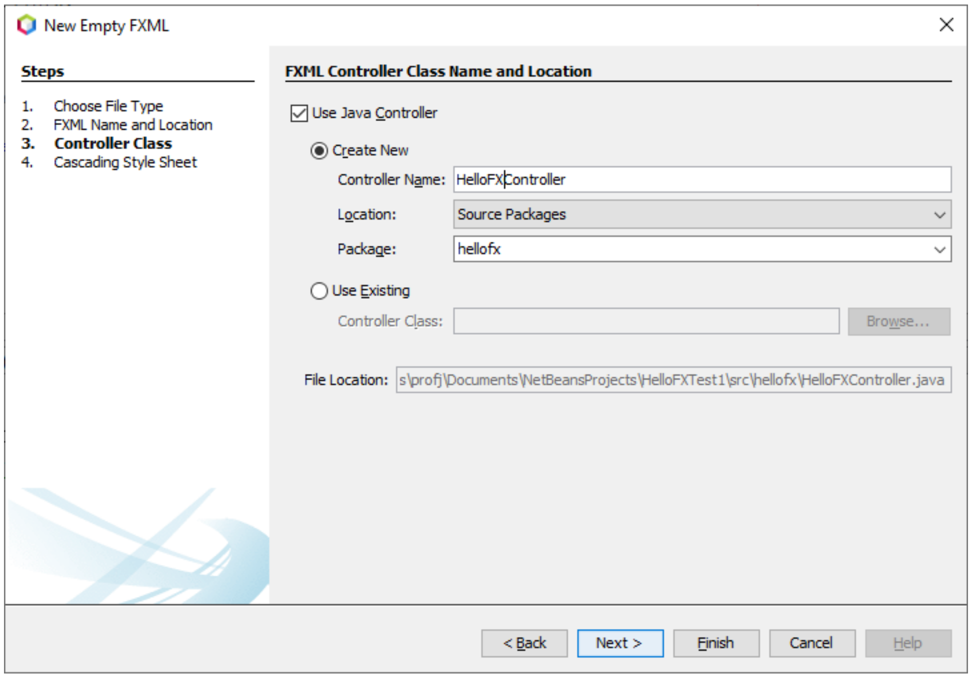
1. Create a Java package in the Project
    a. Right-click in the Project and go to New→Java Package
    b. I named mine **hellofx**
2. Within the package hellofx, **create** a **regular Java class file** named **HelloFX** (or whatever you'd prefer) which will serve as the starting/entry point JavaFX file
3. Although Ant is not fully equipped to handle JavaFX projects yet, it does have some features that can make your life a little easier
4. Right-click the package **hellofx** and go to New→Other and then from this dialog, look for a **JavaFX category and click on it**
    - If you can't find this category, you can always just go to Other→Empty file and make it .fxml at the end, and then use Scene Builder to populate it
5. **Select Empty FXML** from File Types



Creates a new FXML file and optionally Java Controller and CSS files. Ready to be used with Scene Builder.

Figure 18 Creating an Empty FXML file

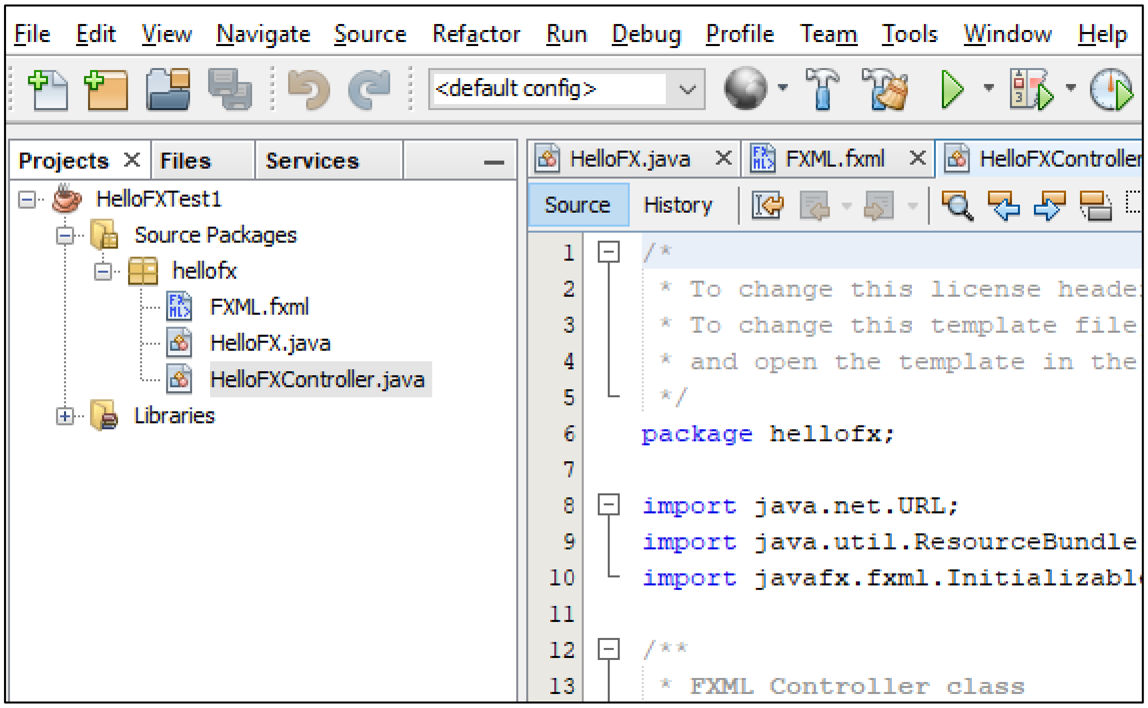
6. Hit the **Next** button
7. On this new part of the dialog, you can now check the checkbox next to **Use Java Controller**



Use Java Controller [✓]
Create New [✓]
Controller Name: HelloFXController
Location: Source Packages
Package: hellofx
Use Existing
Controller Class:
File Location: s\profj\Documents\NetBeansProjects\HelloFXTest1\src\hellofx\HelloFXController.java

Figure 19 Creating a new JavaFX controller

8. Select the **Create New** radio button
9. Give the Controller a useful name such as **HelloFXController**
    - By convention, I always tend to put the name of my main file (HelloFX) followed by Controller, but this is not required to name it ending in Controller, or even related to the main file name
10. Hit **Next**
11. Hit **Finish**
12. You should now have 3 files in your package



```
/*
 * To change this license heade
 * To change this template file
 * and open the template in the
 */
package hellofx;

import java.net.URL;
import java.util.ResourceBundle
import javafx.fxml.Initializabl

/**
 * FXML Controller class
```

Figure 20 Three files in the hellofx package

13. In your HelloFX class (the class that will house main), extend the Application class and put launch(args) inside of the main method, thus:

```
public class HelloFX extends Application {
    public static void main(String[] args) {
        launch(args);
    }
}
```

14. Now, you will have some errors, starting with the file not knowing what Application is
15. Click inside the word Application and hit **Alt + Enter** keys on Windows
    - Alternatively, you can click the little icon in the margins
16. You should select the option “Add import for javafx.application.Application”
17. Now with that error resolve, launch(args) error is resolved also
18. However, now you have a new error: HelloFX is underlined now
    - This is because you are claiming to extend an abstract class with an abstract method that needs to be implemented, but you haven’t put an implementation in yet
19. The easiest way to get the error to go away is to, again, click inside the word HelloFX and hit the Alt+Enter keys on the keyboard
20. Select the option “Implement all abstract methods”
21. You should now notice some new code was automatically generated, namely, for the **start** method
22. Delete the exception throw statement
23. Replace it with the following code:

```
@Override
public void start(Stage stage) throws Exception {
  Parent root = FXMLLoader.load(getClass().getResource("FXML.fxml"));
  stage.setTitle("Hello World");
  stage.setScene(new Scene(root));
  stage.show();
}
```

24. Carefully correct the red underlined errors using the Alt + Enter technique as before

Ultimately, this will add several import statements to the file, so your complete file should look something like this:

```
package hellofx;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class HelloFX extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXML.fxml"));
        stage.setTitle("Hello World");
        stage.setScene(new Scene(root));
        stage.show();
    }
}
```

25. **BE CAREFUL** and make sure that the string parameter passed to getResource() matches your FXML file name
    - My file is just named FXML.fxml, so that's what the string parameter to getResource() contains

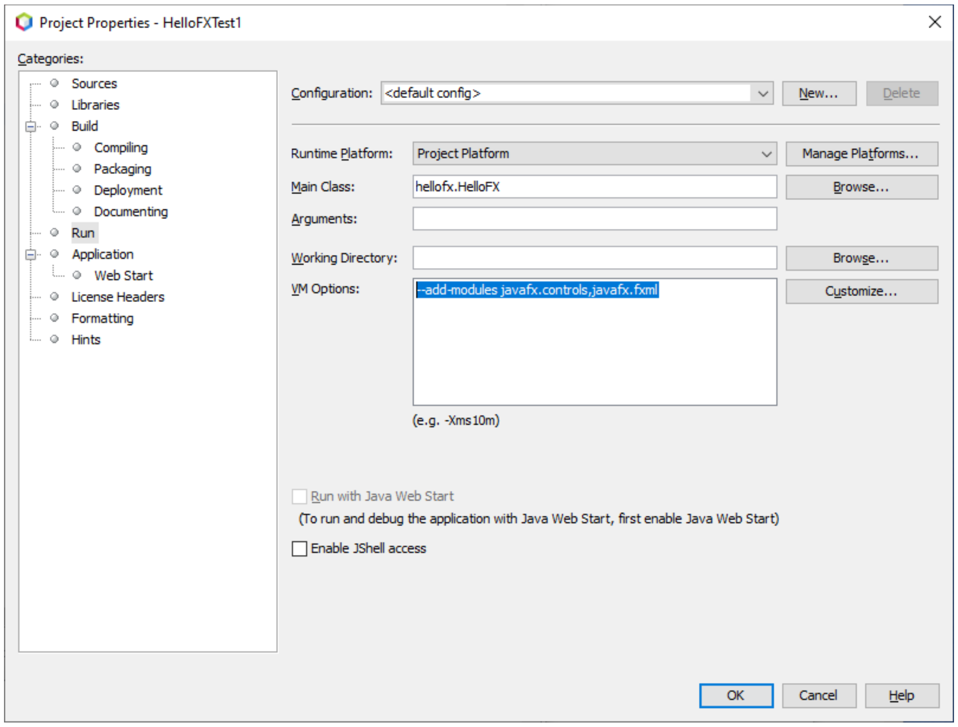
5.3 Correcting errors trying to run the JavaFX application

1. At this point, **right click the HelloFX.java** file and go to **Run**
2. You will see a runtime error something like this:

```
Error: JavaFX runtime components are missing, and are required to run
this application
C:\Users\profj\AppData\Local\NetBeans\Cache\11.1\executor-
snippets\run.xml:111: The following error occurred while executing this
line:
C:\Users\profj\AppData\Local\NetBeans\Cache\11.1\executor-
snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

3. **Right-click** your project
4. Go to **Properties**
5. Go to **Libraries** category
6. Under the **Run** tab, click the **+** button next to **Modulepath**
7. Select the **Add Library** option **and add the JavaFX13 library** to this Modulepath
8. While still under the **Project Properties**, go to the **Run** category
9. For the VM Options, add the appropriate arguments for adding whatever modules you're using. For our simple project, the following should be enough:

```
--add-modules javafx.controls,javafx.fxml
```



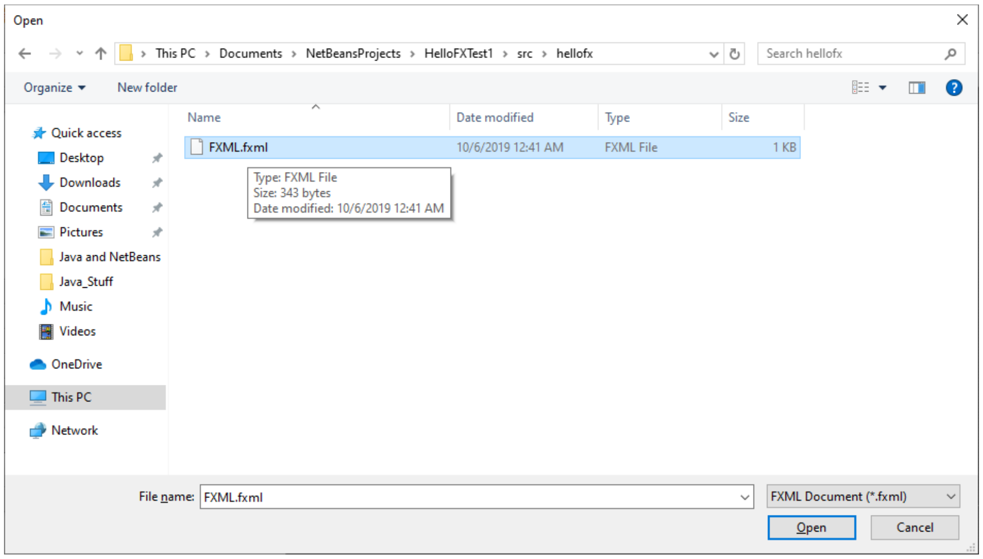
Configuration: <default config>
Runtime Platform: Project Platform
Main Class: hellofx.HelloFX
Arguments:
Working Directory:
VM Options: --add-modules javafx.controls,javafx.fxml
(e.g. -Xms10m)
Run with Java Web Start
(To run and debug the application with Java Web Start, first enable Java Web Start)
Enable JShell access

Figure 21 Adding the VM Options for adding JavaFX modules to our project

10. Hit **OK**
11. Now, right-click the HelloFX class and go to **Run**
12. A very boring, but very *actually working* window should finally pop up!
13. Yay! Now let's make it a little more interesting with Scene Builder

5.4 Installing and using Scene Builder to make our JavaFX GUI

1. Go to https://gluonhq.com/products/scene-builder/#download on Gluon's website and download the Scene Builder installer for Windows
2. The installation of Scene Builder is unremarkable, so just install it as you would any other application
3. Keep the HelloFX project we were working on still open in Apache NetBeans
    a. You closed it, didn't you? Well, open it up again!
4. From within **Scene Builder**, go to **File→Open** and **navigate** to the FXML file in our project
5. Open the FXML file (mine is just named FXML.fxml, which is a bad name… oops. Make yours more descriptive!)



Name	Date modified	Type	Size
FXML.fxml	10/6/2019 12:41 AM	FXML File	1 KB

Type: FXML File
Size: 343 bytes
Date modified: 10/6/2019 12:41 AM

File name: FXML.fxml
FXML Document (*.fxml)

Figure 22 Opening the FXML file from Scene Builder

6. In the **left pane** in Scene Builder, **expand** the **Controls** accordion section
7. Drag a Button and a Label onto the AnchorPane

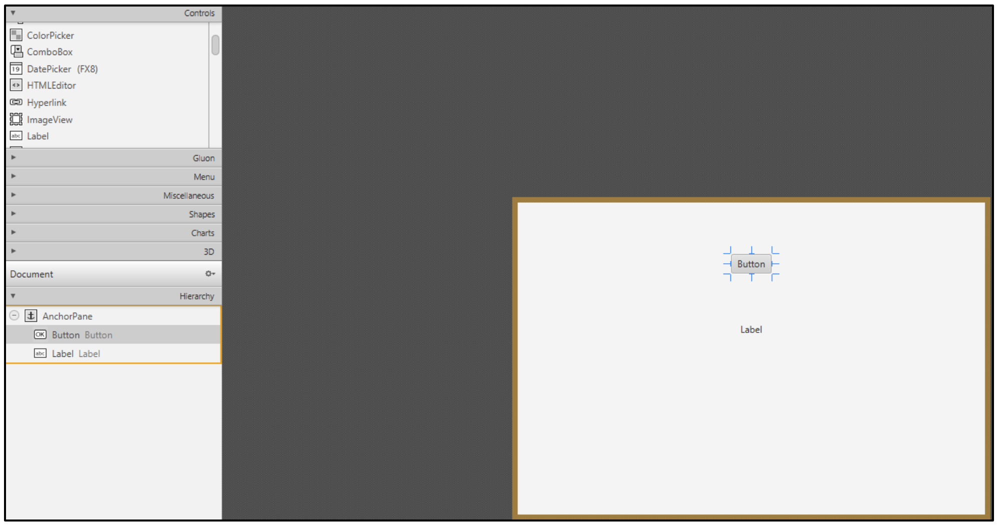
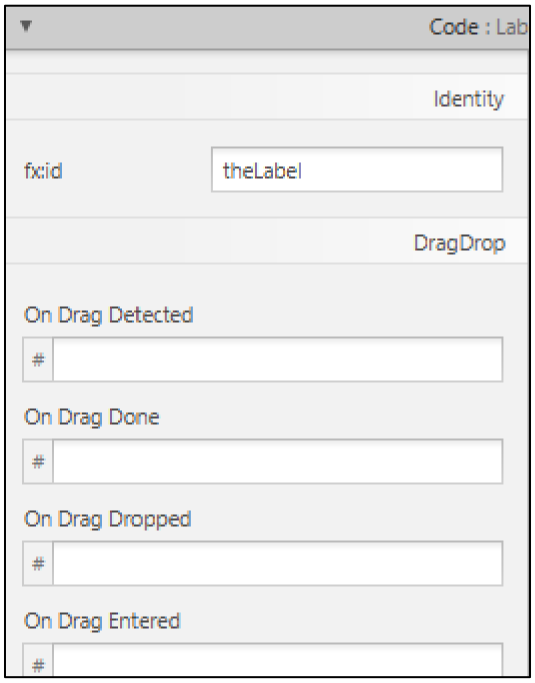


Figure 23 A button and label on the AnchorPane in Scene Builder

8. With the **Label selected** in the main work area, look under the **Code** accordion section in the right pane
9. Change the fx:id to something that would be a valid Java identifier, such as **theLabel**

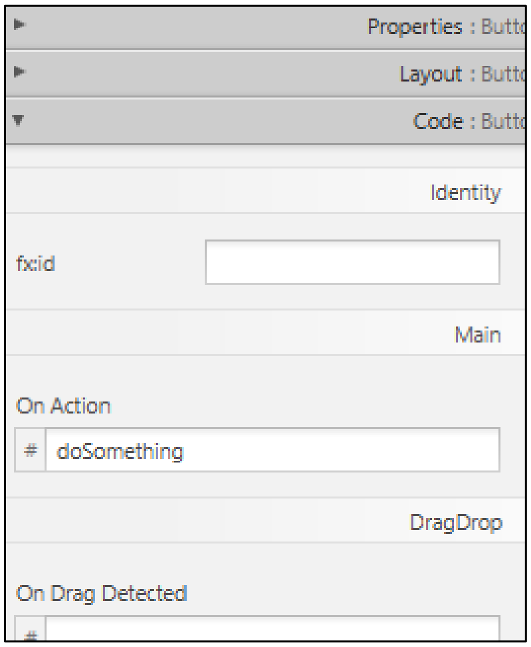


fx:id theLabel

Figure 24 Giving the label an fx:id in the Code section in Scene Builder

10. The fx:id is necessary so we can refer to this label by name in our Java code
    a. Make sure you're under Code and setting the **fx:id**, and not setting the CSS regular old id instead (this id is under the Properties accordion section)

11. Now, select the **Button** in the main work area, look under the **Code** accordion section in the right pane



doSomething

Figure 25 Changing the On Action setting to a valid method name

12. In the **On Action** text field, type a valid Java identifier
    a. This will be used as a method name, triggered as the event handler when the button is clicked
13. You can change the text displayed by the Button and/or the label under the corresponding Properties accordion sections, if you wish
14. When you are done, go to **File → Save**

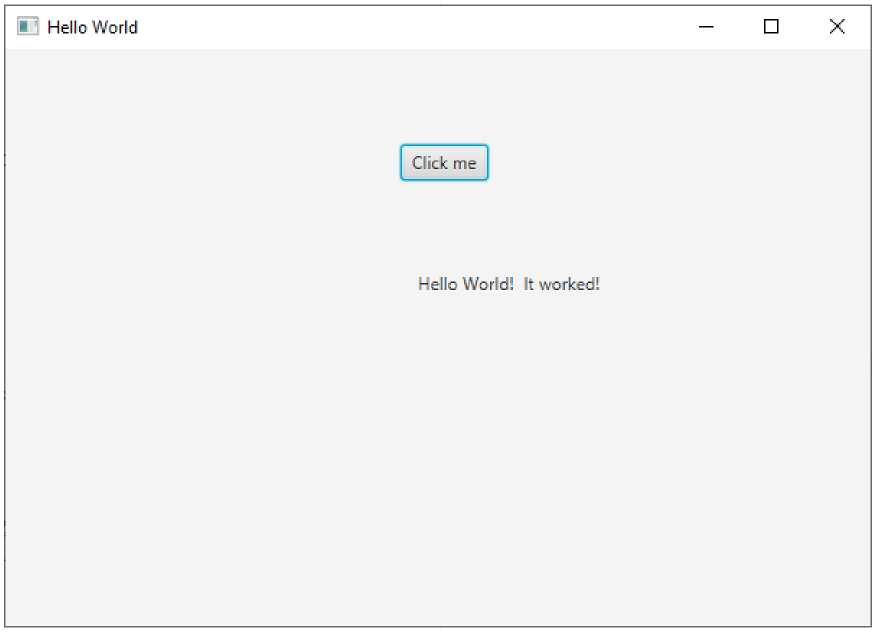
5.5 Editing our code in Apache NetBeans and running our program

1. Back in Apache NetBeans, if you look in the FXML file, you'll notice it has been updated
2. You'll notice it doesn't recognize doSomething for example, which we need to fix
3. If you right-click the FXML file, and go to Make Controller, code for the label's identifier (theLabel) and the doSomething method (both with @FXML annotations above them) will appear in the HelloFXController file
4. In the doSomething action method (event handler), modify the code to change the text of the label when the user clicks the button:

```
@FXML
private void doSomething(ActionEvent event) {
    theLabel.setText("Hello World!  It worked!");
}
```

5. Save the file

6. Run the program again and interact with it
7. You should get the result, as follows:



Hello World! It worked!

Figure 26 The final piece of artwork, running

8. Woohoo! It works!
9. Go take a nap, you earned it!