Eclipse and MarieSimulator

Introduction

This tutorial is intended to introduce some of the features of the Eclipse IDE using files from an existing JAVA teaching application – MarieSimulator see
http://computerscience.jbpub.com/ecoa/2e/student_resources.cfm

For the new user, this application also demonstrates Swing GUI features and complicated Data Structures. Therefore, it can act as a point of reference for the features/approach of a realistic application. However, MarieSimulator uses the older Swing GUI package which is being replaced by JavaFX.

You can download the latest version of Eclipse from: http://www.eclipse.org/downloads/packages/ for your architecture (32bit or 64bit and platform). Oxygen (September 2017) is the latest version. For JAVA, you need
Eclipse IDE for Java Developers or Eclipse IDE for Java EE Developers. Google "how to install eclipse on …" for help.

You may also need to research via Google search:

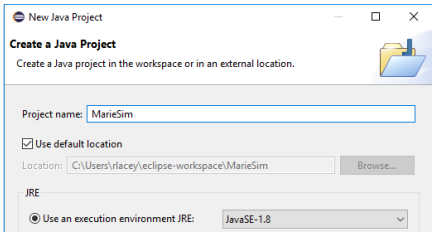
Perspectives	https://www.tutorialspoint.com/eclipse/eclipse_perspectives.htm
Eclipse IDE Views:	
Java Packages	
Oxygen Install:	https://www.youtube.com/watch?v=5cphDv9GDS8
Oxygen IDE Improvements:	https://www.youtube.com/watch?v=Y0-cO9wZj6w
Window Builder:	http://www.eclipse.org/windowbuilder/download.php
Java Packages:	

Create MarieSimulator

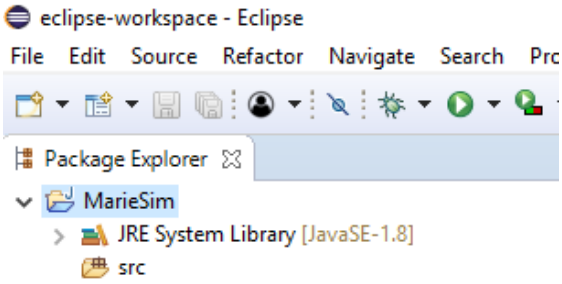
1. Start eclipse

2. File -> New -> Java Project

3. Set the Project Name: MarieSim

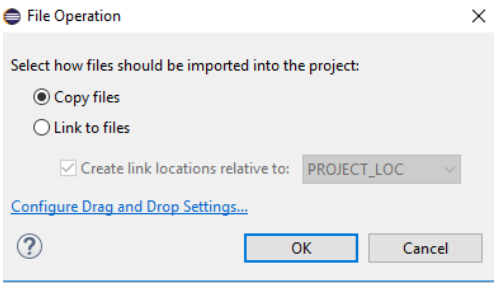


producing:



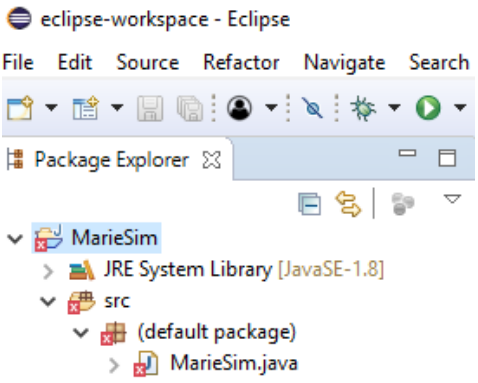
4. Import MarieSim.java

Using a File Explorer find the file MarieSim.java, then drag and drop it into the src folder within Eclipse.

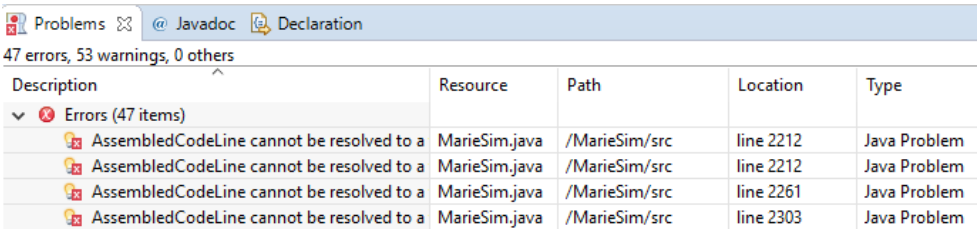


Take the Copy Files option by clicking OK.

Producing:



Notice the error indications, which are detailed in the Problems window.



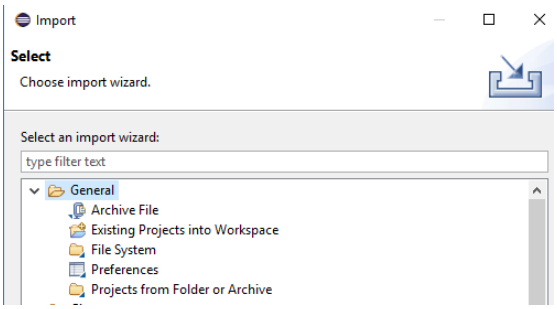
These errors are to be expected, as we have not brought in all the files yet.

Notice the first error is related to a file called AssembledCodeLine.

5. Import AssembledCodeLine.java

Now rather than dragging and dropping, lets do a file import a Wizard.

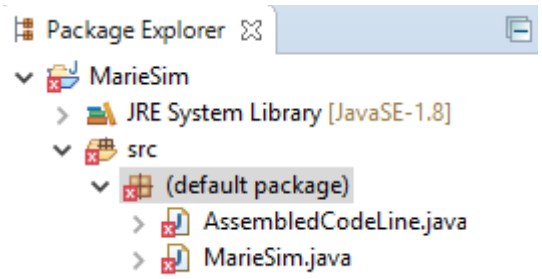
Right-mouse click on the (default package), then select Import from the menu.



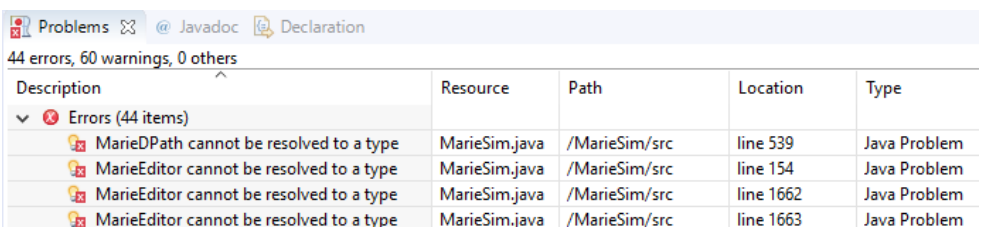
Select File System

Use the browse button and navigate to the folder with the relevant files and tick/select the file AssembledCodeLine.java then click Finish.

The package explorer should appear as follows:



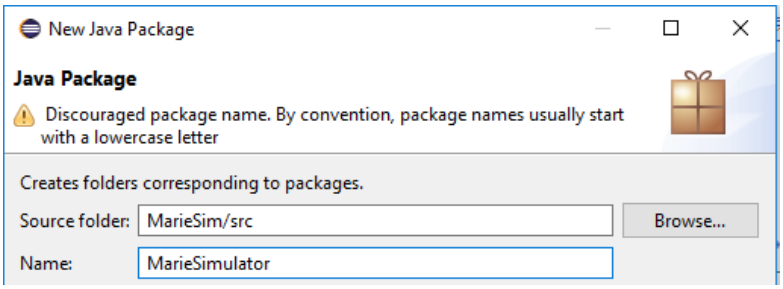
We still have errors, but they are different to before, AssembledCodeLine.java has fixed 4 errors and introduced one new error.



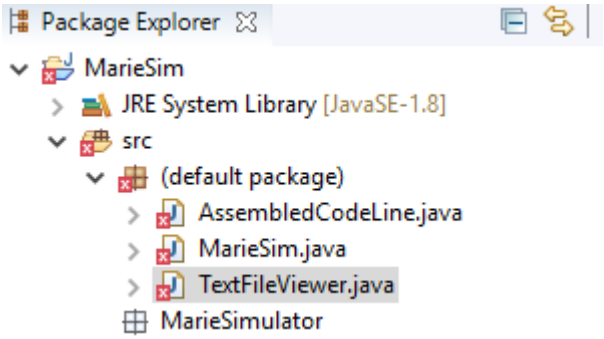
Double-click AssembledCodeLine.java you will see that package MarieSimulator; is an error source, double-click MarieSim.java and it is an error here also.

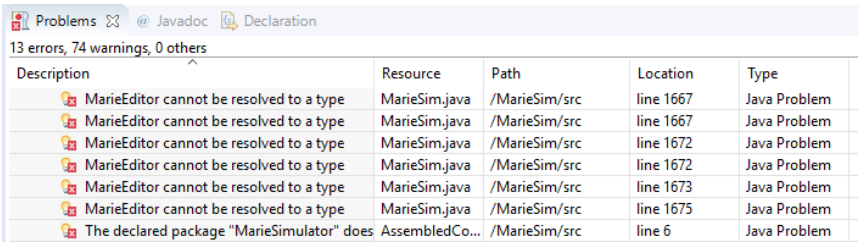
6. Create Package MarieSimulator

Click on src in the package explorer and select New->Package and type MarieSimulator in the name input (ignore the warnings/convention) and click finish.



Now drag and drop TextFileViewer.java into the (default package) and the errors should reduce significantly.



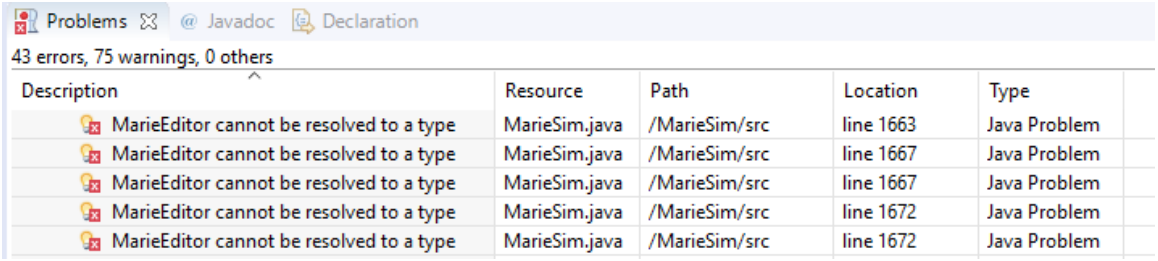


All three files are part of the MarieSimulator package (later), which is currently empty so the resources it provides are still missing in these files and generating errors.

7. Import MarieEditor.java

Now drag and drop MarieEditor.java into the package MarieSimulator.

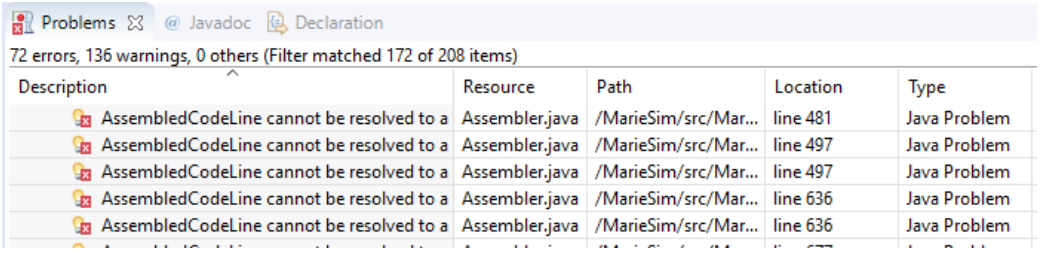
The error count goes up.



8. Import Assembler.java and MarieDPath.java

Now drag and drop both Assembler.java and MarieDPath.java into into the package MarieSimulator.

The error count increases again.



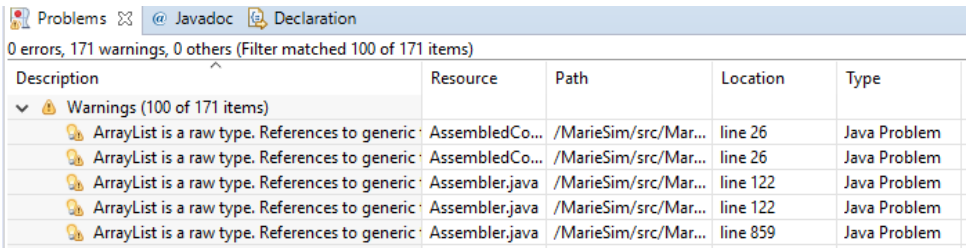
9. Move files to MarieSimulator package

The errors are still present as two of the files we placed in the (default package) should actually be in the package MarieSimulator. So lets move them. Drag and drop from (default package) into MarieSimulator the files: AssembledCodeLine.java and TextFileViewer.java

10. Move MarieSim.java to MarieSimulator package

There are still errors, so drag and drop MarieSim.java into the package MarieSimulator.

Now all we have are warnings as opposed to errors.



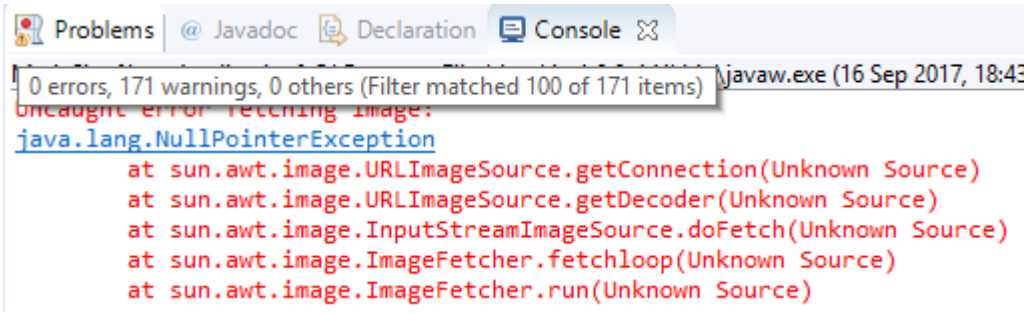
11. Run /Launch Application

Now try and run the application (there a number of ways to do this).

Right-mouse click the file MarieSim.java and select Run As->Java Application.

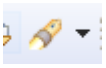
Note: MarieSim.java contains the main function. If you select a run option that does not clearly indicate where the main function is located, you will be prompted to identify the location class.

This produces the following error in the Console window. The application is trying to run but is missing a resource. Errors like this can sit in code until encountered at some point in application (leading to the "but it worked earlier" etc.)

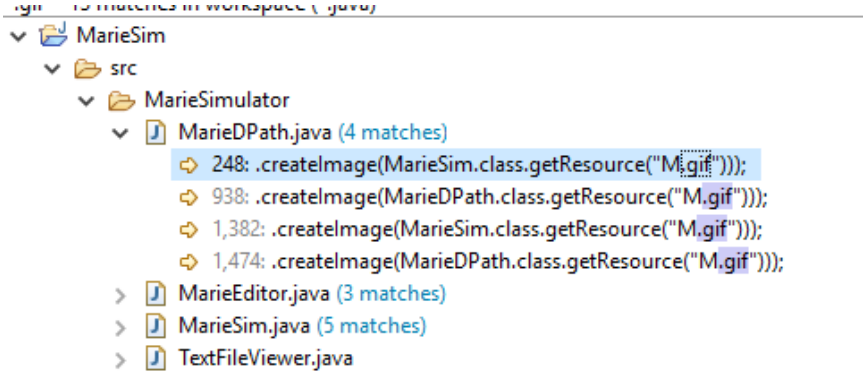


The error states: `Uncaught error fetching image:`

So, somewhere in the code is a reference to an image that cannot be found.

Use the search feature from the Eclipse toolbar  and take the file search option.

Search for ".gif" (we know the error cause).



This is indicating a file M.gif is required in the application, but we have not provided it.

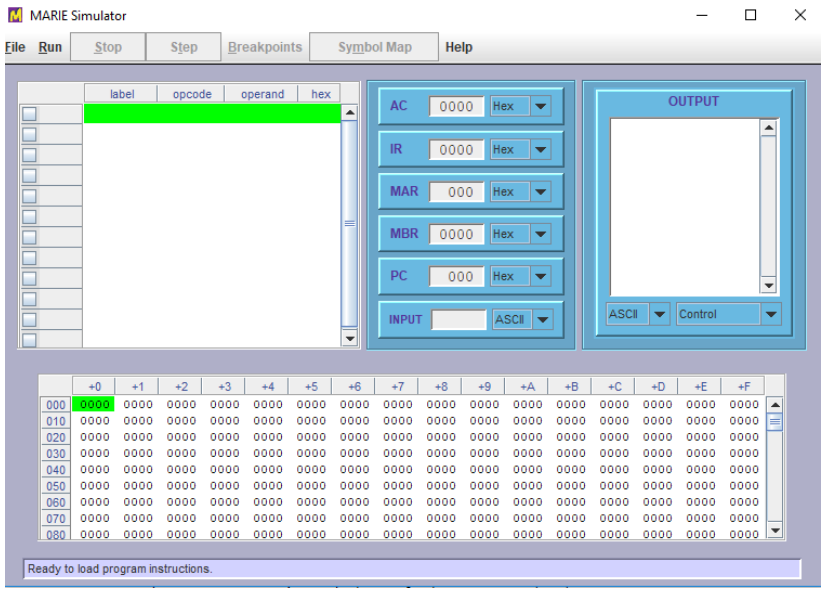
12. Add resource M.gif

Find this file and drag and drop (or import) in the MarieSimulator package.

13. Run

Again, Right-mouse click the file MarieSim.java and select Run As->Java Application.

This should produce a running application.



You have now seen some of the steps to create a package, move files, resolve issues, search and run an application.

While this application may use dated GUI features it can act a reference source for the goals of a larger application.

This is a simulator for a simple CPU architecture to run its own assembly language.

There is a sample file in the HelloWorld folder to be assembled and loaded to run.

Obviously we are re-engineering this project, if you were the original creator you may not have encountered the deliberate mistakes made here to show-off features of the Eclipse IDE. This MARIE is available

Rename the (Default package) to MarieSimulator and place all the java files into this package.