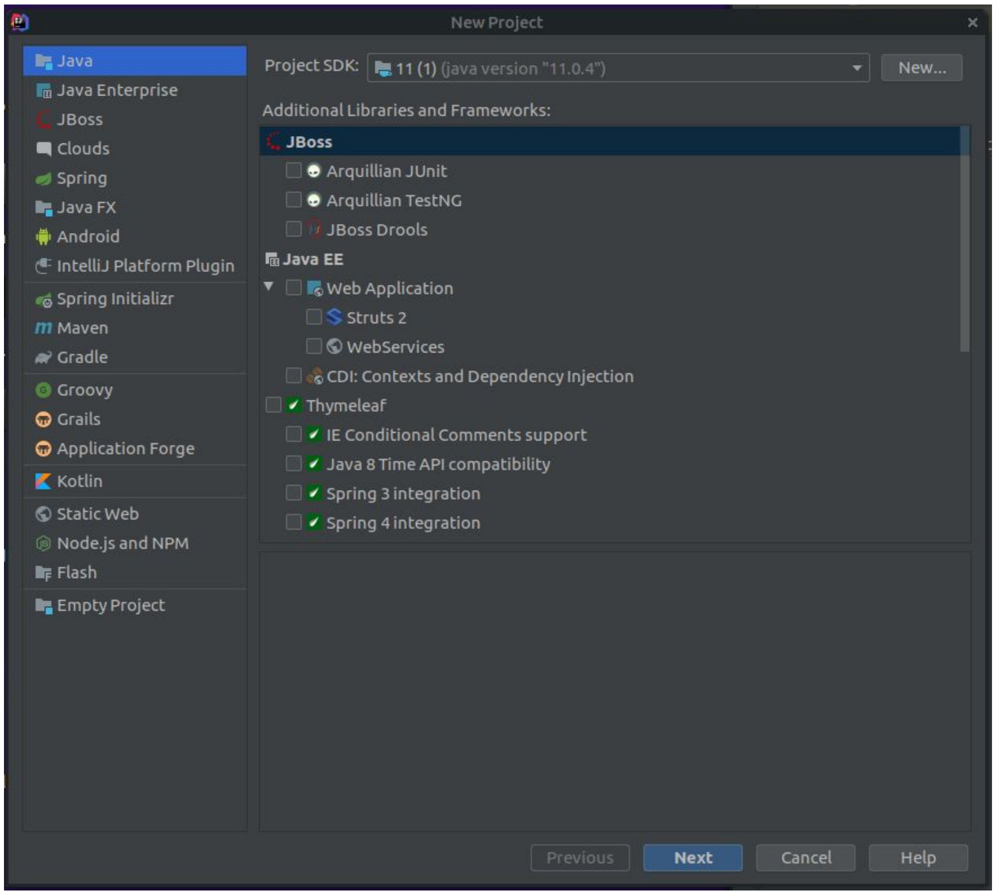
Setting Up Image Editor Unit Tests

Downloads

First you will need to visit the Projects section of the current CS 240 website and download the "Test Files (Zip)" folder. Inside this zip you will find 4 folders: jars, key_images, out_images, source_images, and test_files.

Create a Project

Go ahead and open IntelliJ and create a new Project.



Project SDK: 11 (1) (java version "11.0.4")

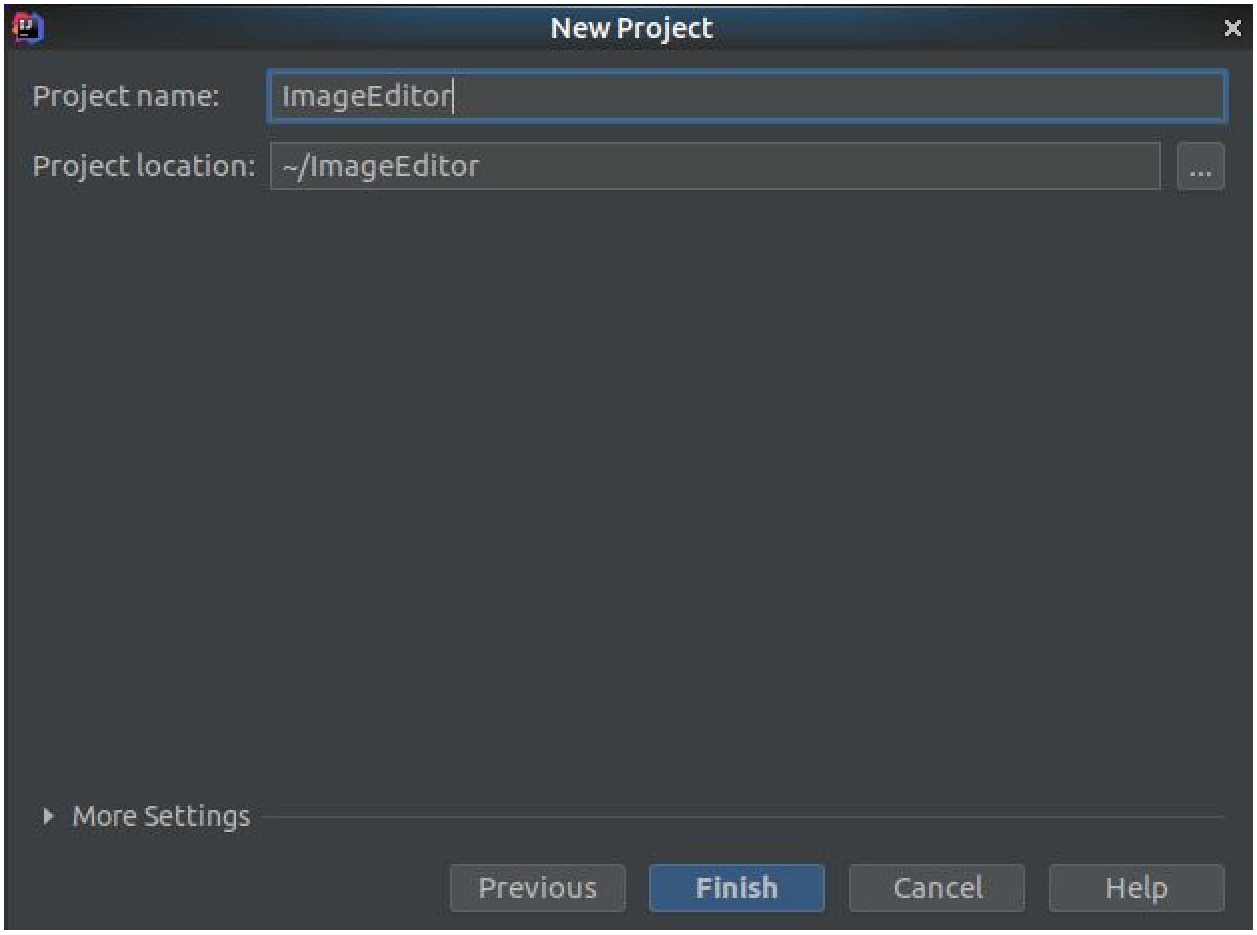
Additional Libraries and Frameworks:

- JBoss
  - [ ] Arquillian JUnit
  - [ ] Arquillian TestNG
  - [ ] JBoss Drools
- Java EE
  - [ ] Web Application
    - [ ] Struts 2
    - [ ] WebServices
  - [ ] CDI: Contexts and Dependency Injection
- [ ] Thymeleaf
  - [ ] IE Conditional Comments support
  - [ ] Java 8 Time API compatibility
  - [ ] Spring 3 integration
  - [ ] Spring 4 integration

This tutorial will be setting up as a normal Java project. If you wish to create a Maven or Gradle project you will be responsible for setting it up on your own and getting dependencies working correctly.

Go ahead and select "Java" for your project type and in "Project SDK" select whatever version of Java you want (You should select whatever is the latest version of Java installed on your

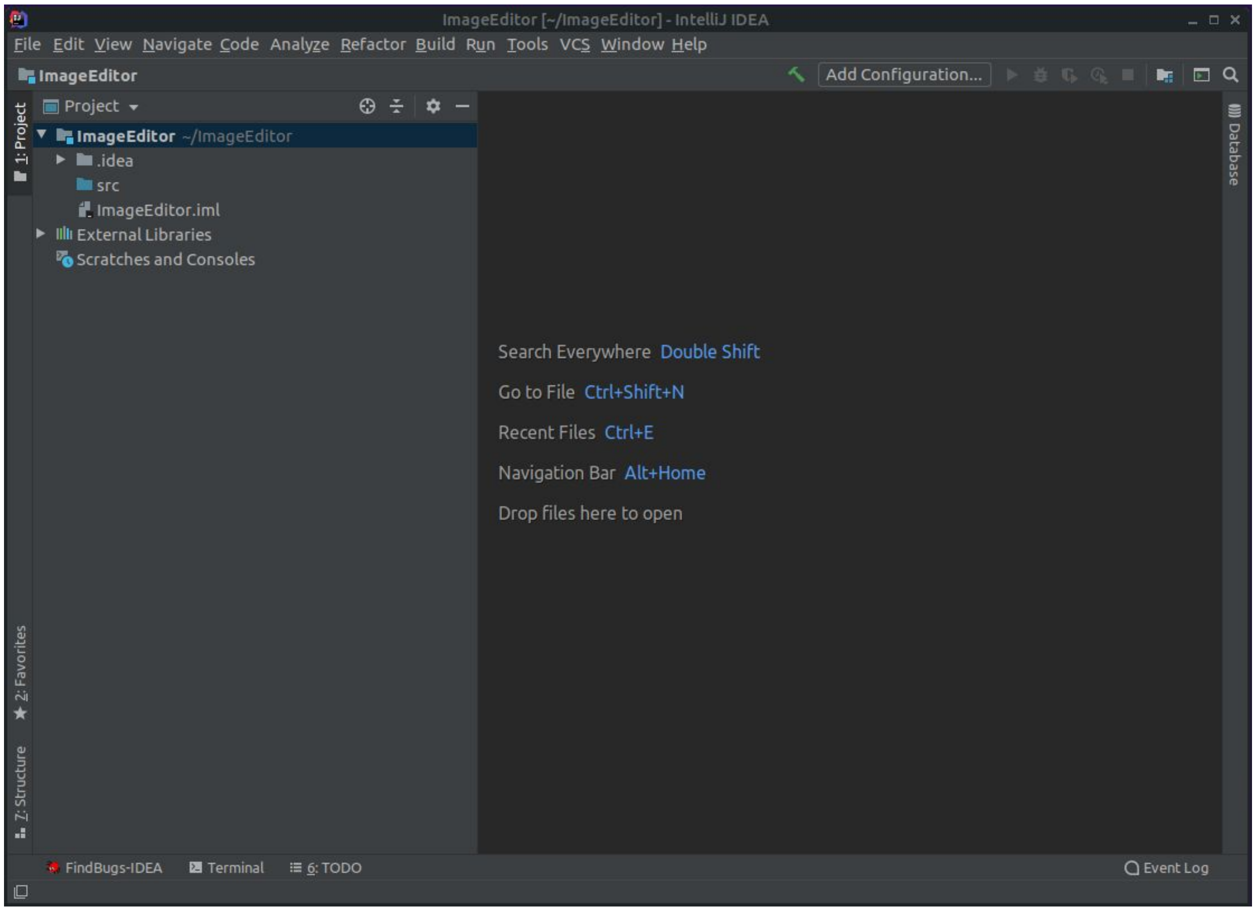
computer). You do not need to select any additional libraries or frameworks. Go ahead and click "Next".
Continue to click "Next" until you reach this window:



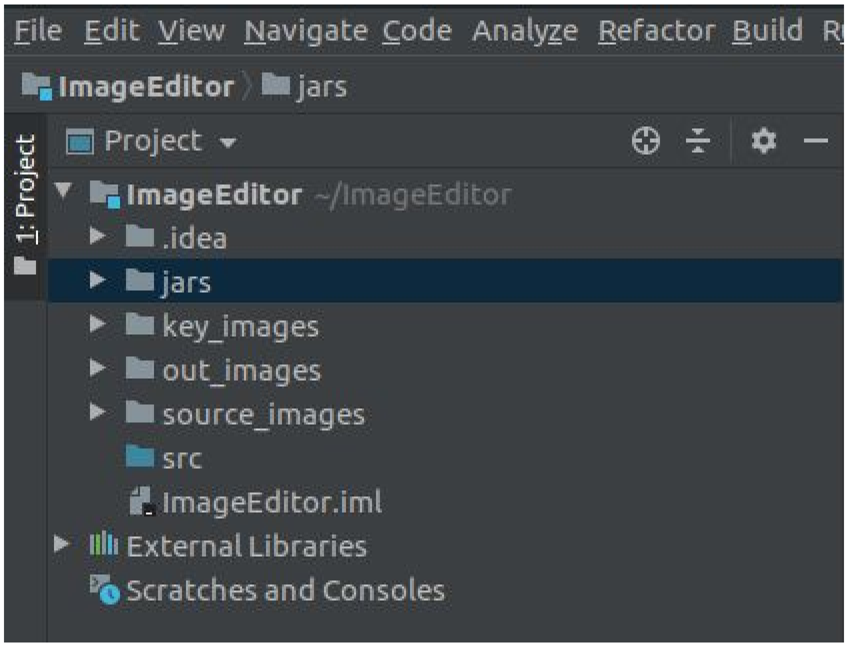
Project name: ImageEditor
Project location: ~/ImageEditor

Go ahead and name your project whatever you want though we suggest using ImageEditor.
Click "Finish"

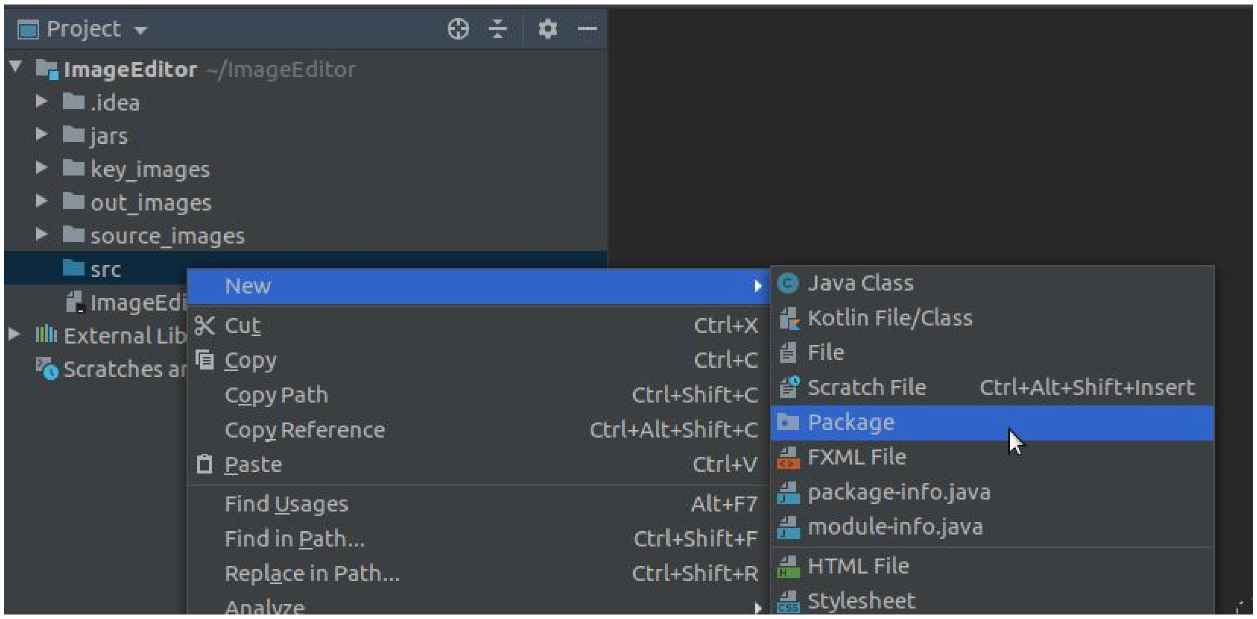
Now your project will build and will open:



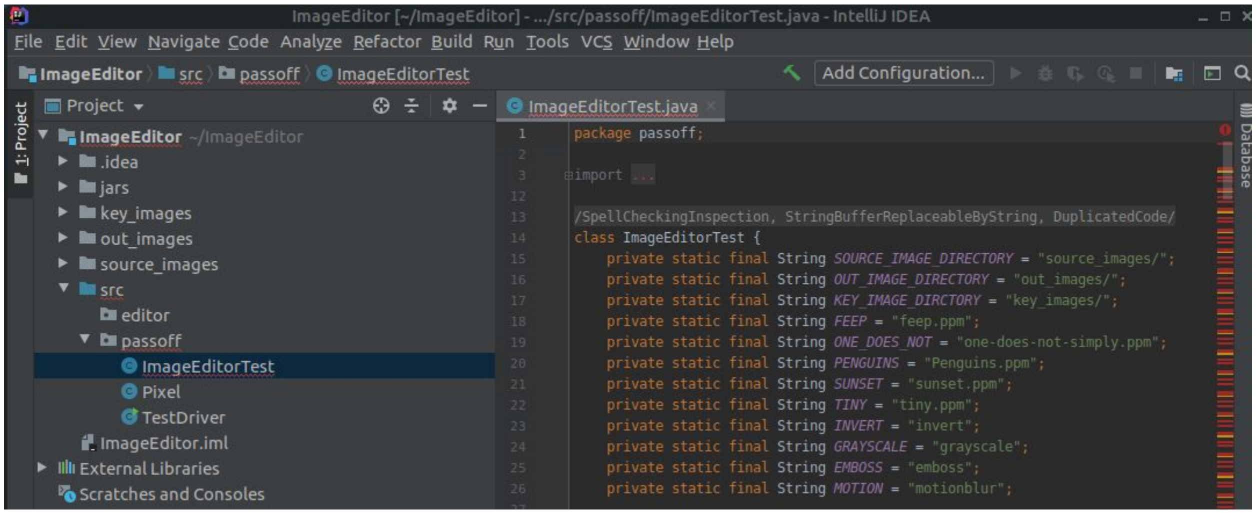
Go ahead and open the zip file you downloaded earlier and move jars, key_images, out_images, and source_images into the main folder of your project.



Right click on your src folder and add two new packages. One needs to be called passoff, the other can be named whatever you want.



Move the files from test_files into passoff


ImageEditor [~/ImageEditor] - .../src/passoff/ImageEditorTest.java - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ImageEditor src passoff ImageEditorTest
Add Configuration...
Project
ImageEditor ~/ImageEditor
.idea
jars
key_images
out_images
source_images
src
editor
passoff
ImageEditorTest
Pixel
TestDriver
ImageEditor.iml
External Libraries
Scratches and Consoles
ImageEditorTest.java
1: Project
Database

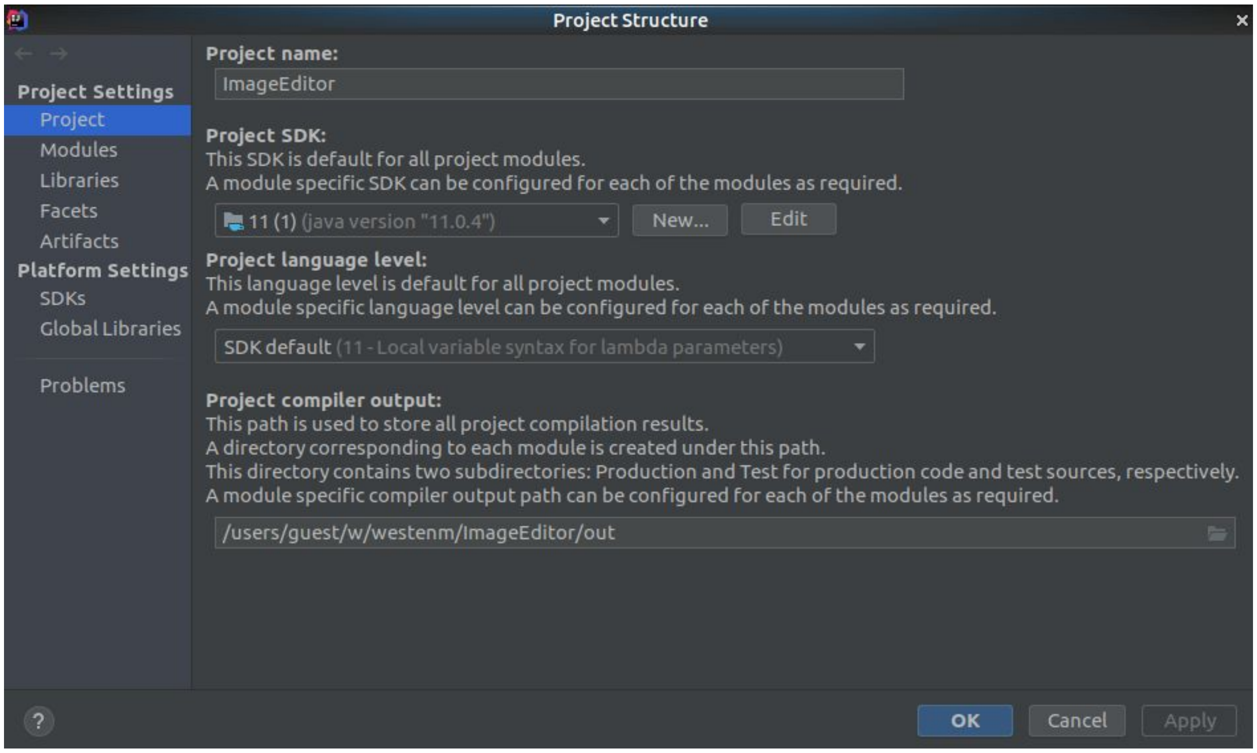
```java
package passoff;

import ...

/SpellCheckingInspection, StringBufferReplaceableByString, DuplicatedCode/
class ImageEditorTest {
    private static final String SOURCE_IMAGE_DIRECTORY = "source_images/";
    private static final String OUT_IMAGE_DIRECTORY = "out_images/";
    private static final String KEY_IMAGE_DIRCTORY = "key_images/";
    private static final String FEEP = "feep.ppm";
    private static final String ONE_DOES_NOT = "one-does-not-simply.ppm";
    private static final String PENGUINS = "Penguins.ppm";
    private static final String SUNSET = "sunset.ppm";
    private static final String TINY = "tiny.ppm";
    private static final String INVERT = "invert";
    private static final String GRAYSCALE = "grayscale";
    private static final String EMBOSS = "emboss";
    private static final String MOTION = "motionblur";
```

You'll see a lot of red errors, but that is okay for now.

Dependencies

The last step we need to do is add dependencies to your project. Since these tests run on JUnit 5 you need to have the files for JUnit 5 so that IntelliJ can access them and use them. The folder that is labeled "jars" is where all this code is stored as .jar files. So we will add dependencies to your project for these files so that your project knows where to go in order to access the code.

Select File > Project Structure (or use Ctrl+Alt+Shift+S) and you'll see this screen


Project Structure

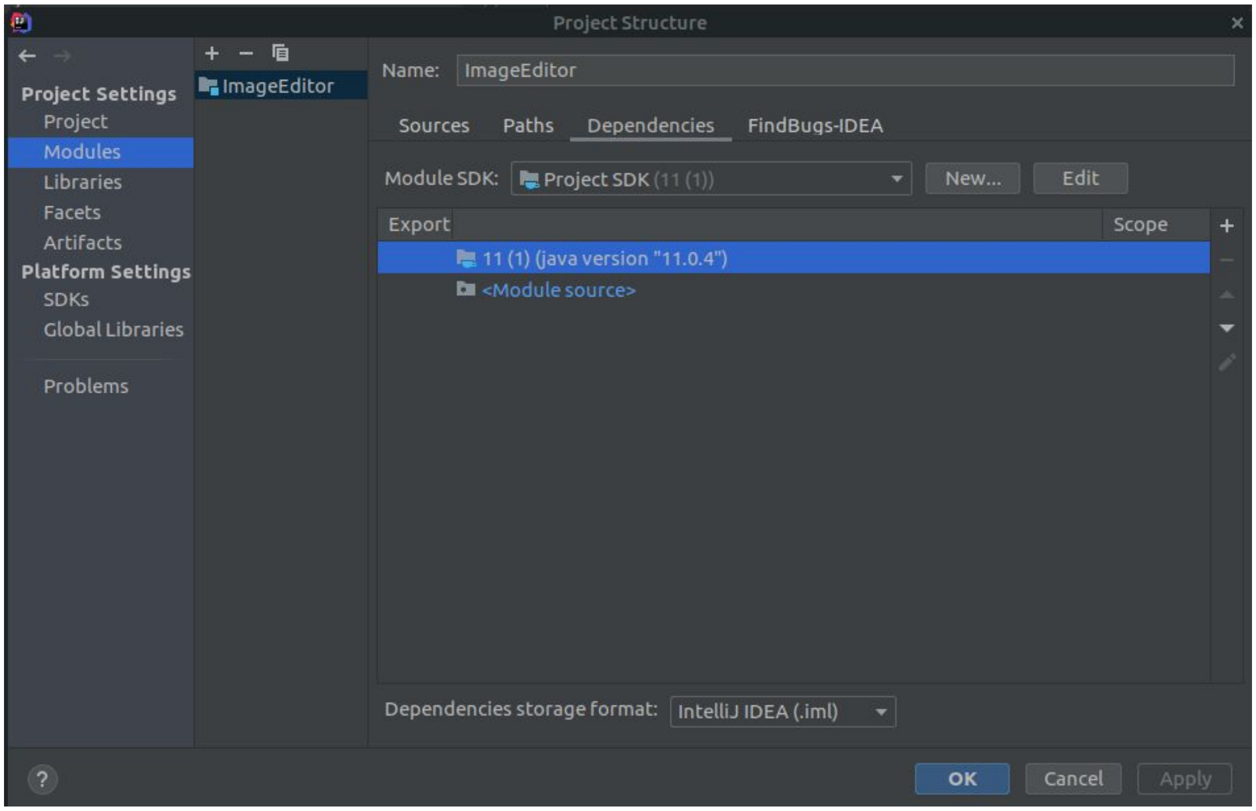
Project name: ImageEditor

Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.
11 (1) (java version "11.0.4")

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.
SDK default (11 - Local variable syntax for lambda parameters)

Project compiler output:
This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.
/users/guest/w/westenm/ImageEditor/out

Double-click Modules and click the tab labeled Dependencies


Project Structure

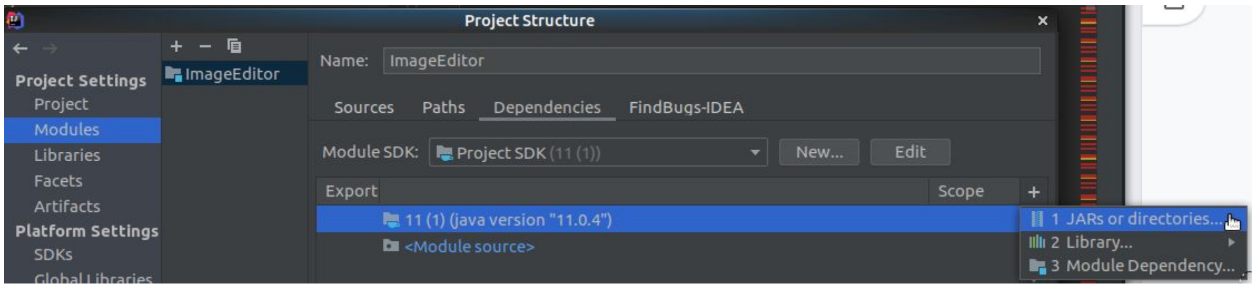
Name: ImageEditor

Module SDK: Project SDK (11 (1))

Export	Scope
11 (1) (java version "11.0.4")	
<Module source>	

Dependencies storage format: IntelliJ IDEA (.iml)

Now click the little “+” icon to the far right of the window and select “JARs or directories”



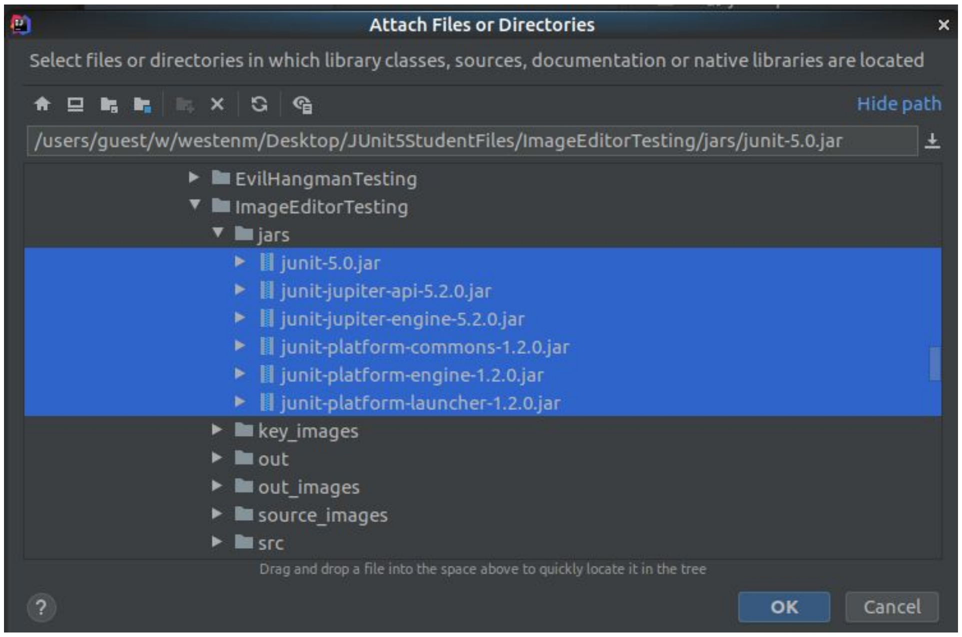
Name: ImageEditor

Module SDK: Project SDK (11 (1))

11 (1) (java version "11.0.4")

<Module source>

Navigate to the jars folder in your project and select all of the jar files then click “OK”



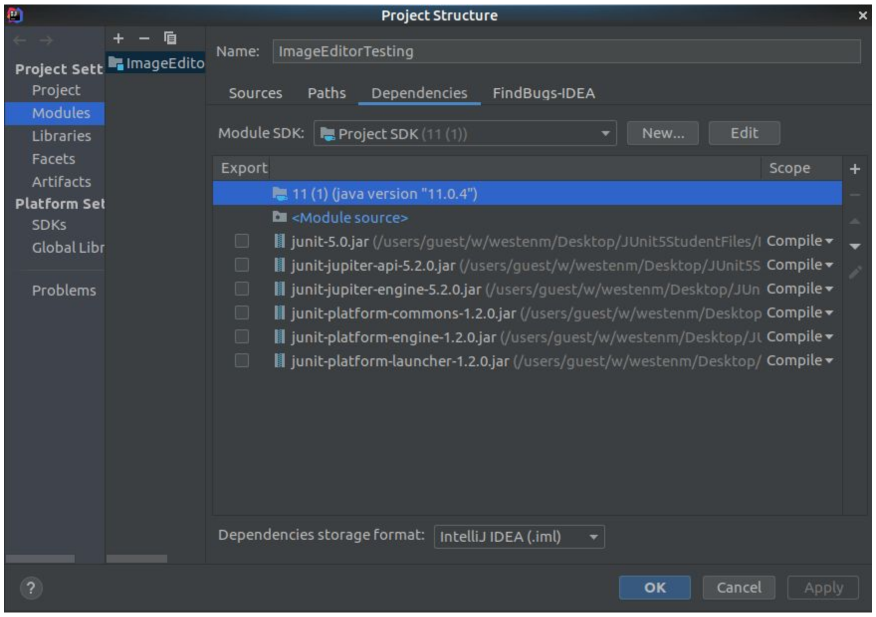
Select files or directories in which library classes, sources, documentation or native libraries are located

/users/guest/w/westenm/Desktop/JUnit5StudentFiles/ImageEditorTesting/jars/junit-5.0.jar

```
EvilHangmanTesting
ImageEditorTesting
  jars
    junit-5.0.jar
    junit-jupiter-api-5.2.0.jar
    junit-jupiter-engine-5.2.0.jar
    junit-platform-commons-1.2.0.jar
    junit-platform-engine-1.2.0.jar
    junit-platform-launcher-1.2.0.jar
  key_images
  out
  out_images
  source_images
  src
```

Drag and drop a file into the space above to quickly locate it in the tree

These will be added to the dependencies list. Go ahead and click “OK”



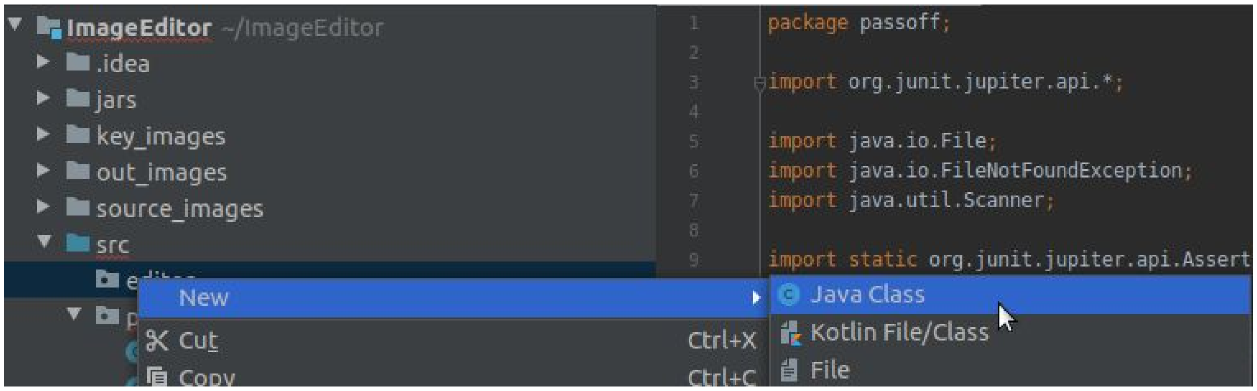
Name: ImageEditorTesting

Module SDK: Project SDK (11 (1))

Export		Scope
	11 (1) (java version "11.0.4")	
	<Module source>	
☐	junit-5.0.jar (/users/guest/w/westenm/Desktop/JUnit5StudentFiles/I	Compile
☐	junit-jupiter-api-5.2.0.jar (/users/guest/w/westenm/Desktop/JUnit5S	Compile
☐	junit-jupiter-engine-5.2.0.jar (/users/guest/w/westenm/Desktop/JUn	Compile
☐	junit-platform-commons-1.2.0.jar (/users/guest/w/westenm/Desktop	Compile
☐	junit-platform-engine-1.2.0.jar (/users/guest/w/westenm/Desktop/Jt	Compile
☐	junit-platform-launcher-1.2.0.jar (/users/guest/w/westenm/Desktop/	Compile

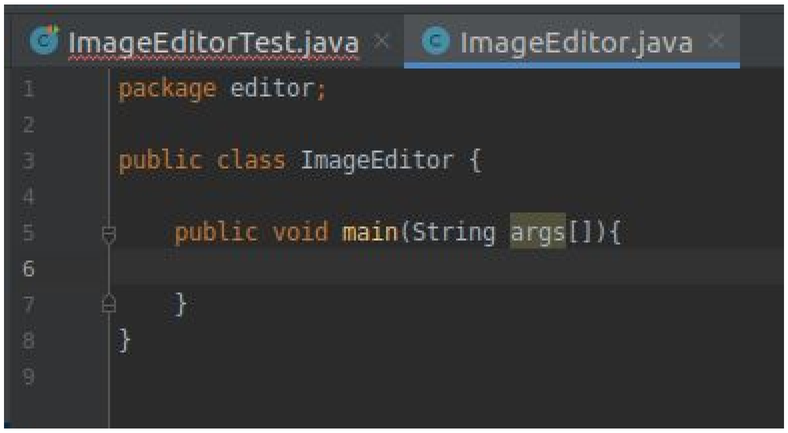
Dependencies storage format: IntelliJ IDEA (.iml)

All of the red errors should disappear except for the ones related to ImageEditor.main(command);
This is because you have not written your code yet. In your other package create a new Java class and call it ImageEditor



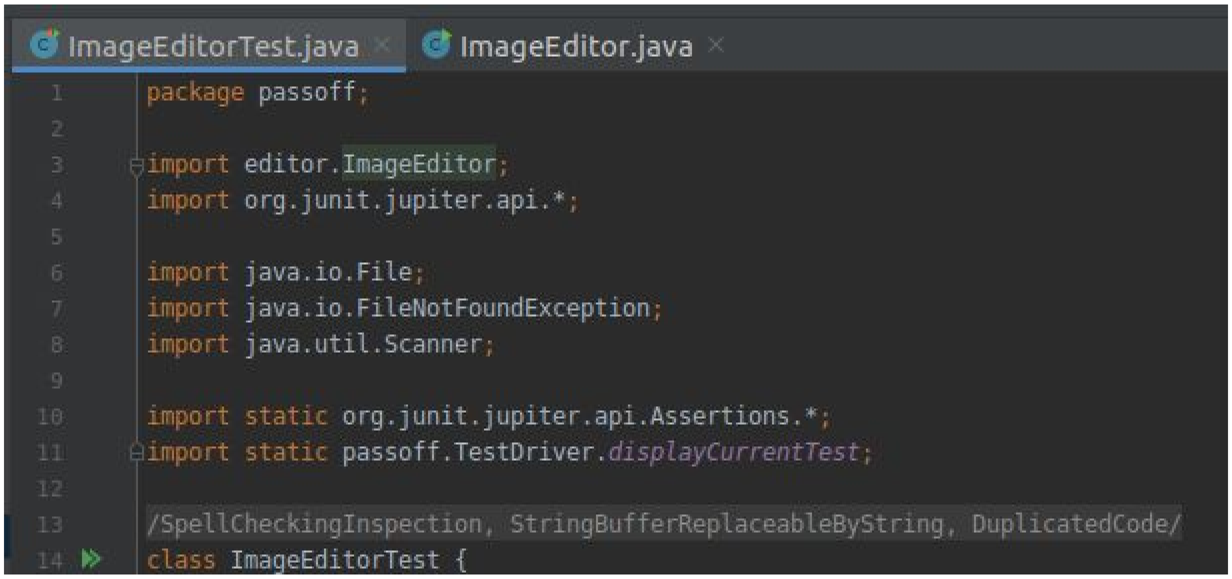
```java
package passoff;

import org.junit.jupiter.api.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import static org.junit.jupiter.api.Assert
```

Give it a standard java main function which will look like this.



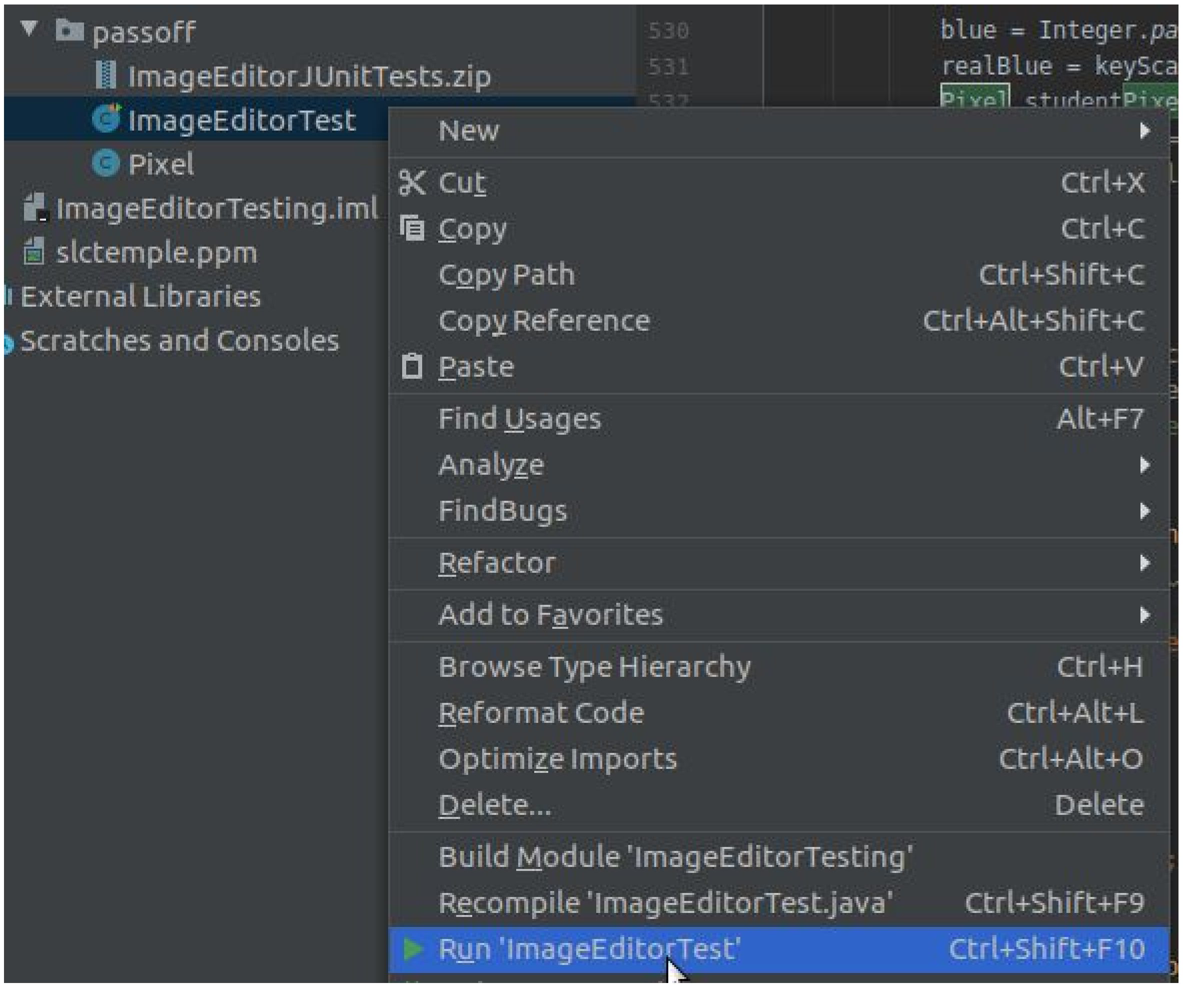
```java
package editor;

public class ImageEditor {

    public void main(String args[]){

    }
}
```

Now make sure that in your ImageEditorTest class it is importing from the correct location

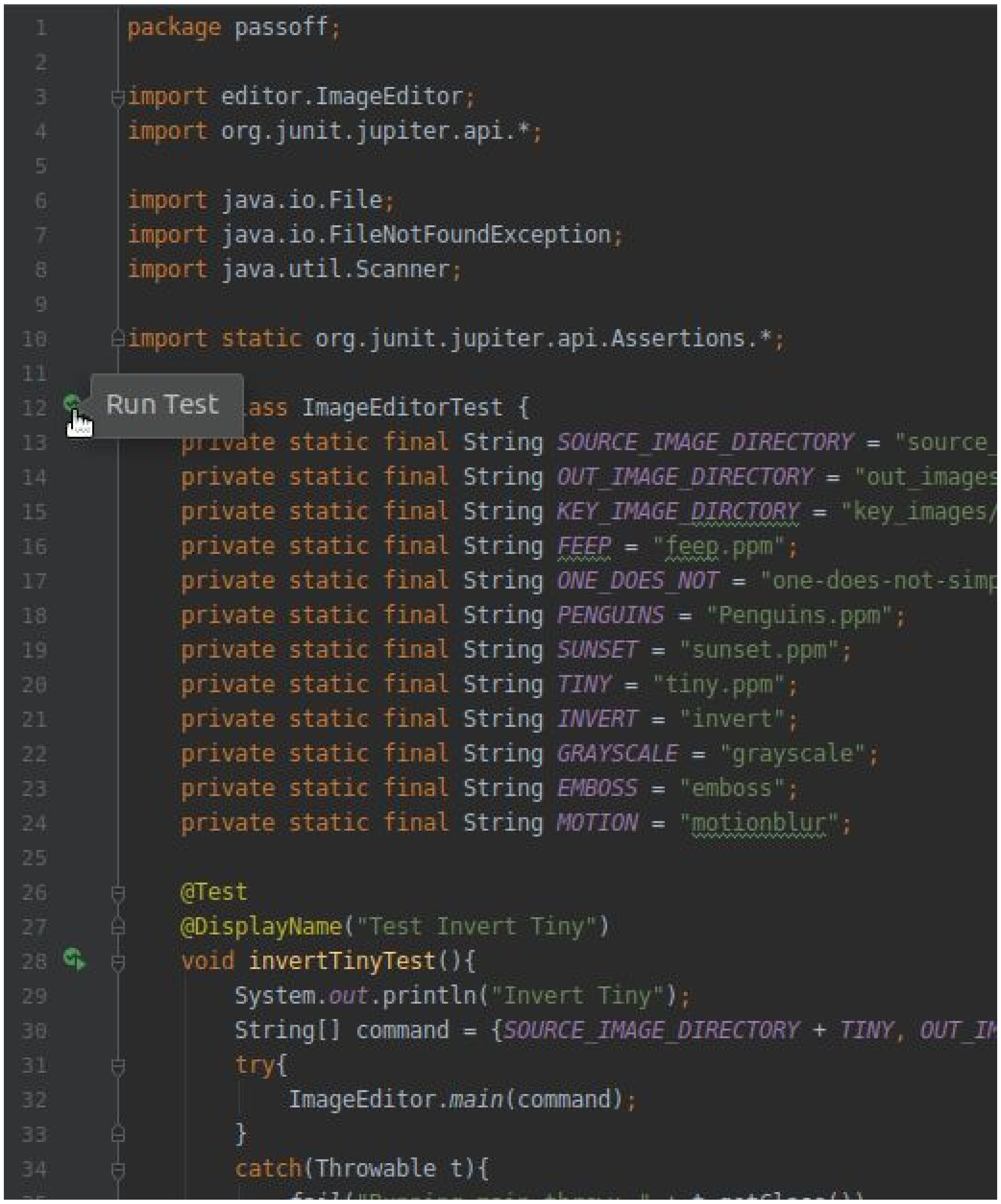


```java
package passoff;

import editor.ImageEditor;
import org.junit.jupiter.api.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import static org.junit.jupiter.api.Assertions.*;
import static passoff.TestDriver.displayCurrentTest;

/SpellCheckingInspection, StringBufferReplaceableByString, DuplicatedCode/
class ImageEditorTest {
```

All of the errors should be gone. Now go ahead and right click on ImageEditorTest and select “Run ImageEditorTest” or click on the green arrow next to the class declaration in the file itself



```
blue = Integer.pa
realBlue = keySca
Pixel studentPixe
```



```
package passoff;

import editor.ImageEditor;
import org.junit.jupiter.api.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import static org.junit.jupiter.api.Assertions.*;

        ass ImageEditorTest {
    private static final String SOURCE_IMAGE_DIRECTORY = "source_
    private static final String OUT_IMAGE_DIRECTORY = "out_images
    private static final String KEY_IMAGE_DIRCTORY = "key_images/
    private static final String FEEP = "feep.ppm";
    private static final String ONE_DOES_NOT = "one-does-not-simp
    private static final String PENGUINS = "Penguins.ppm";
    private static final String SUNSET = "sunset.ppm";
    private static final String TINY = "tiny.ppm";
    private static final String INVERT = "invert";
    private static final String GRAYSCALE = "grayscale";
    private static final String EMBOSS = "emboss";
    private static final String MOTION = "motionblur";

    @Test
    @DisplayName("Test Invert Tiny")
    void invertTinyTest(){
        System.out.println("Invert Tiny");
        String[] command = {SOURCE_IMAGE_DIRECTORY + TINY, OUT_I
        try{
            ImageEditor.main(command);
        }
        catch(Throwable t){
            fail("Running main threw " + t.getClass())
```

This will run the tests. The tests will obviously fail because you have not written your code yet, but now you have the tests all ready to go so you can test as you code.