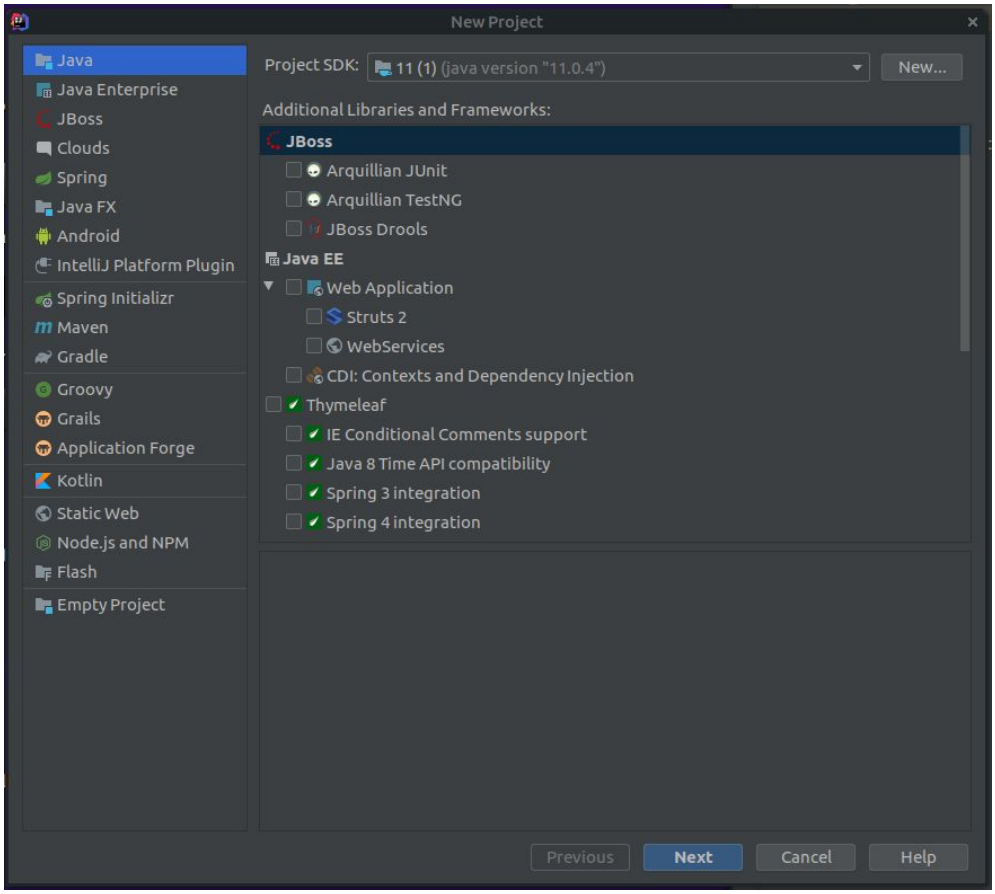
# Setting Up Spelling Corrector Unit Tests

## Downloads

First you will need to visit the Projects section of the current CS 240 website and download the “Files (Zip)” folder as well as the “Test Files (Zip)” folder. Inside this “Test Files (Zip)” you will find 3 dictionary files and two folders labeled “test_files” and “jars”.

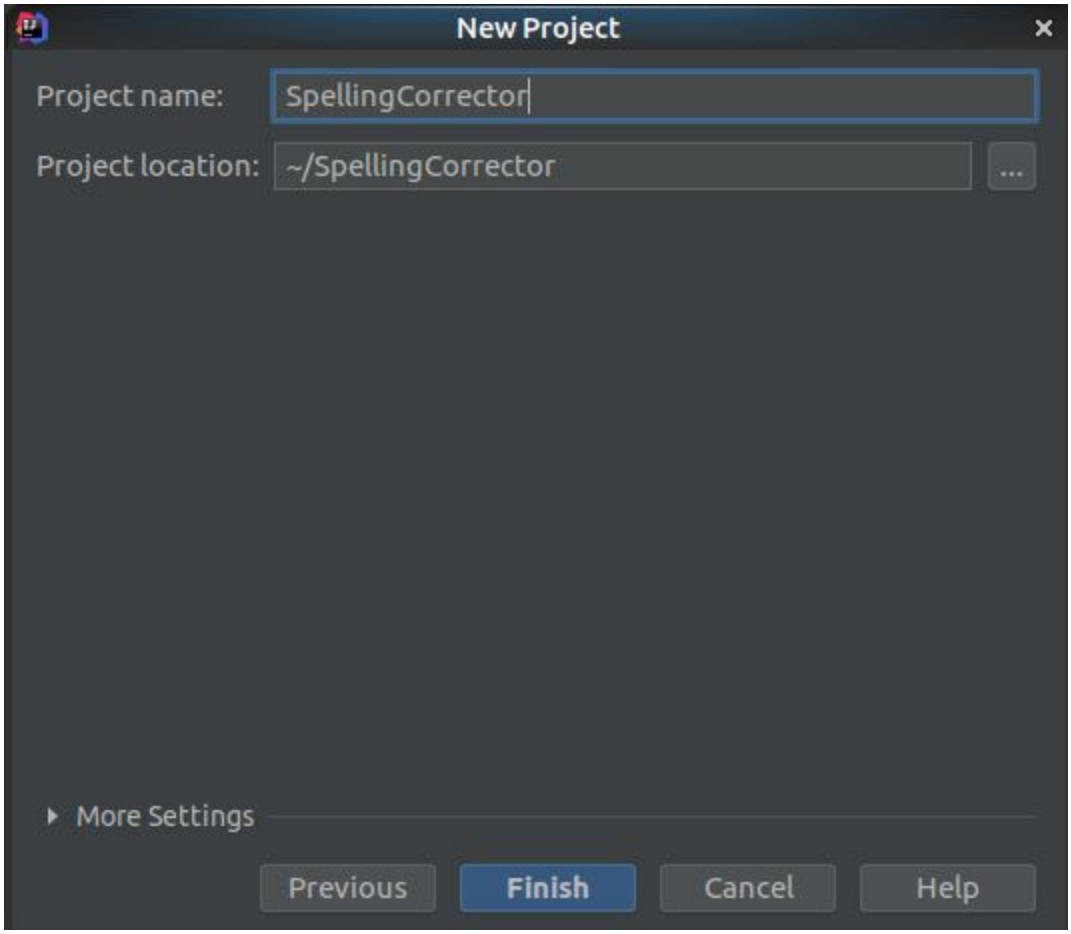
## Create a Project

Open IntelliJ and create a new Project.



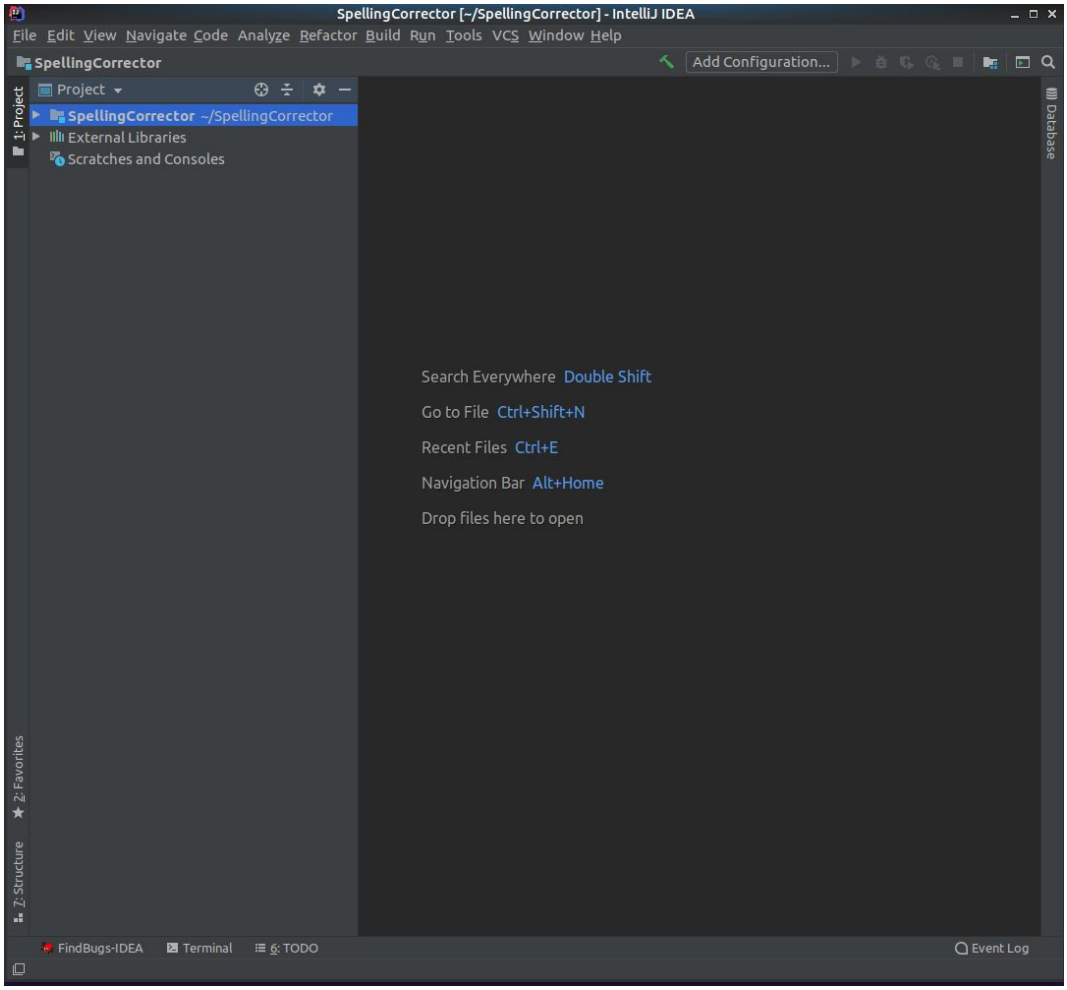

*This tutorial will be setting up as a normal Java project. If you wish to create a Maven or Gradle project you will be responsible for setting it up on your own and getting dependencies working correctly.*

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. Click “Next”.
Continue to click “Next” until you reach this window:

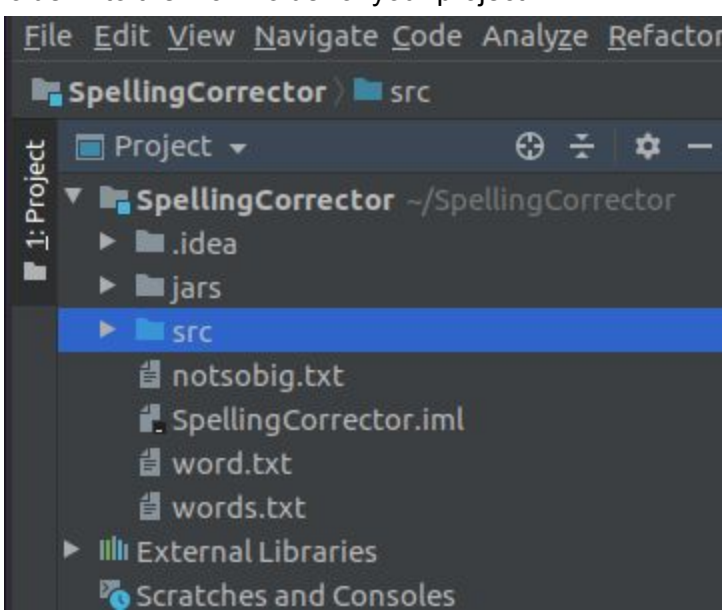



Name your project whatever you want though we suggest using SpellingCorrector. Click “Finish”

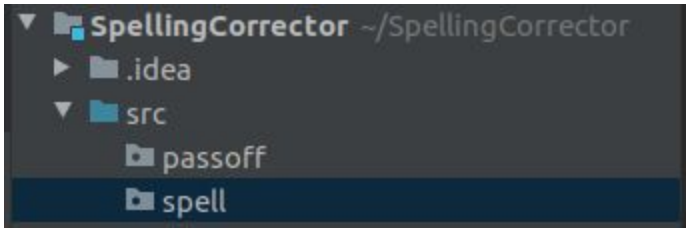
Now your project will build and will open:



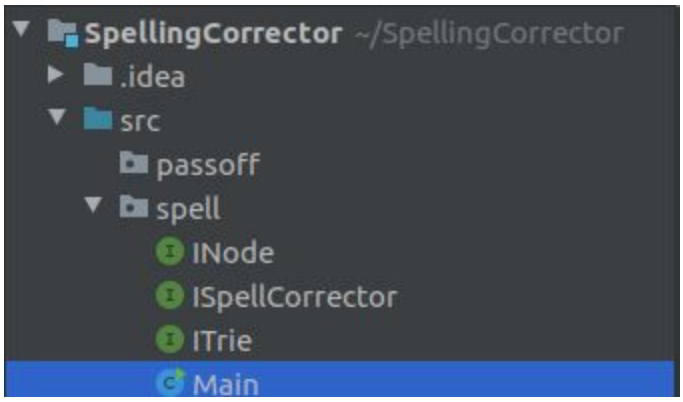

Open the “Test Files (zip)” file you downloaded earlier and move the dictionary files and “jars” folder into the main folder of your project.



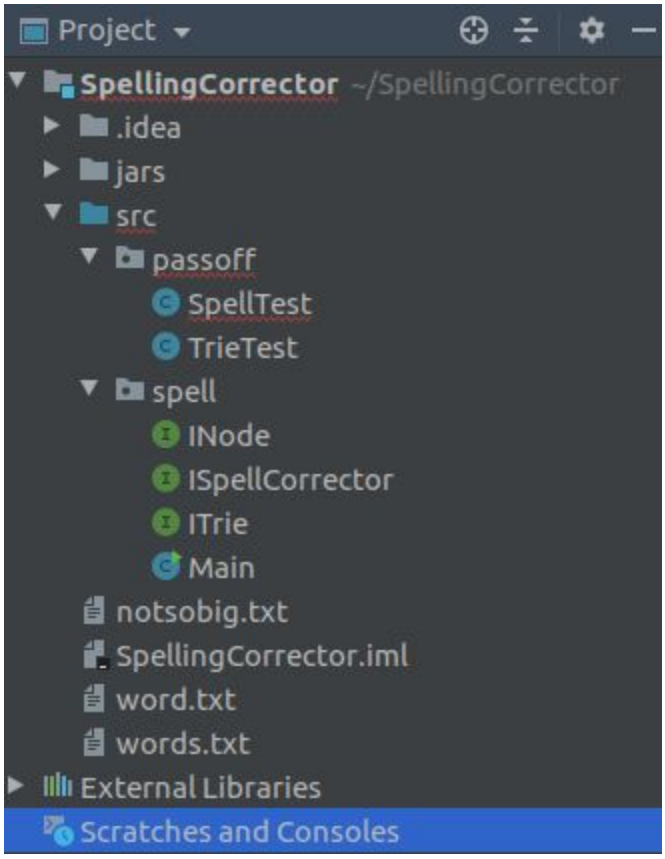
Right click on your src folder and add two new packages (New > Package). One needs to be called passoff, the other needs to be called spell.




Take all the files you downloaded from “Files (Zip)” and move them into the spell folder




Move the files from test_files into passoff

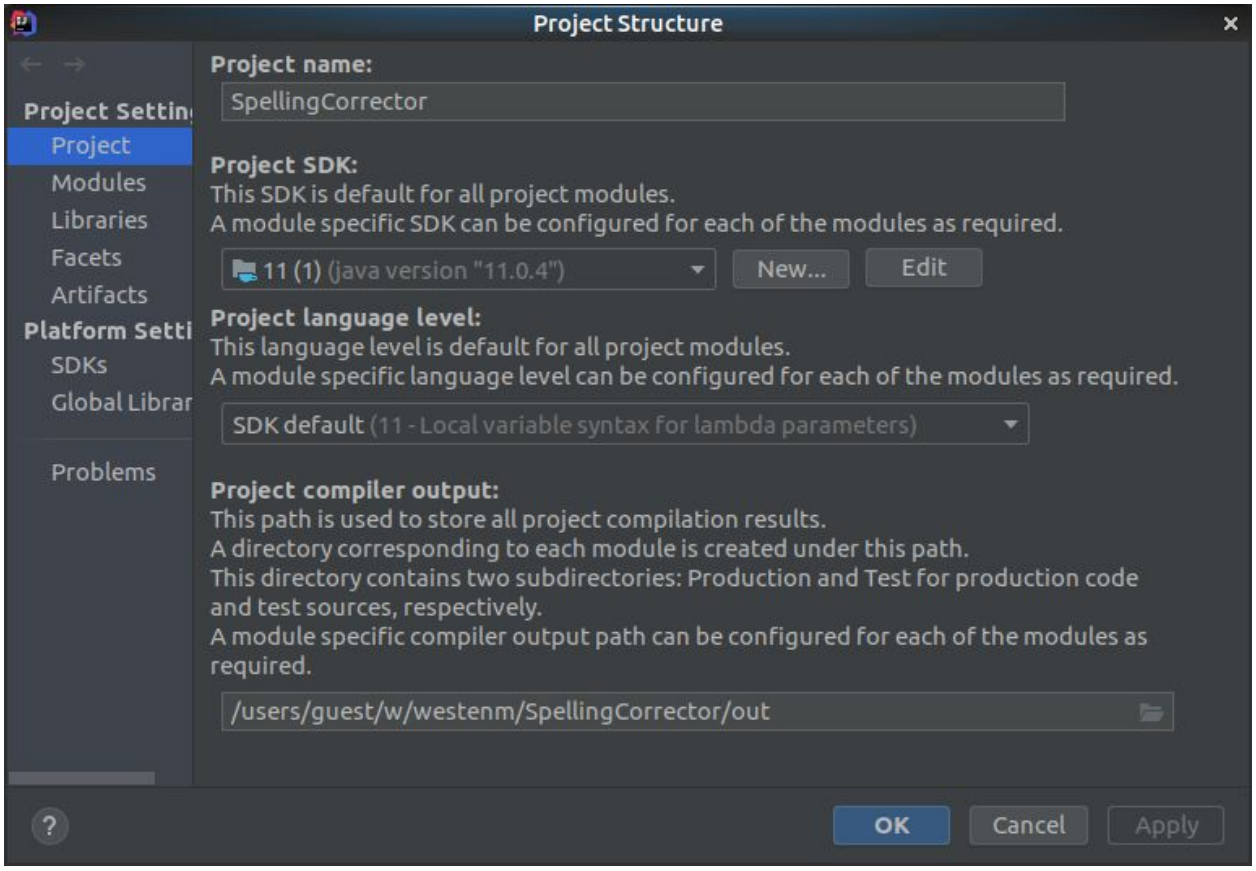



You’ll see a lot of red errors, but that is okay for now.

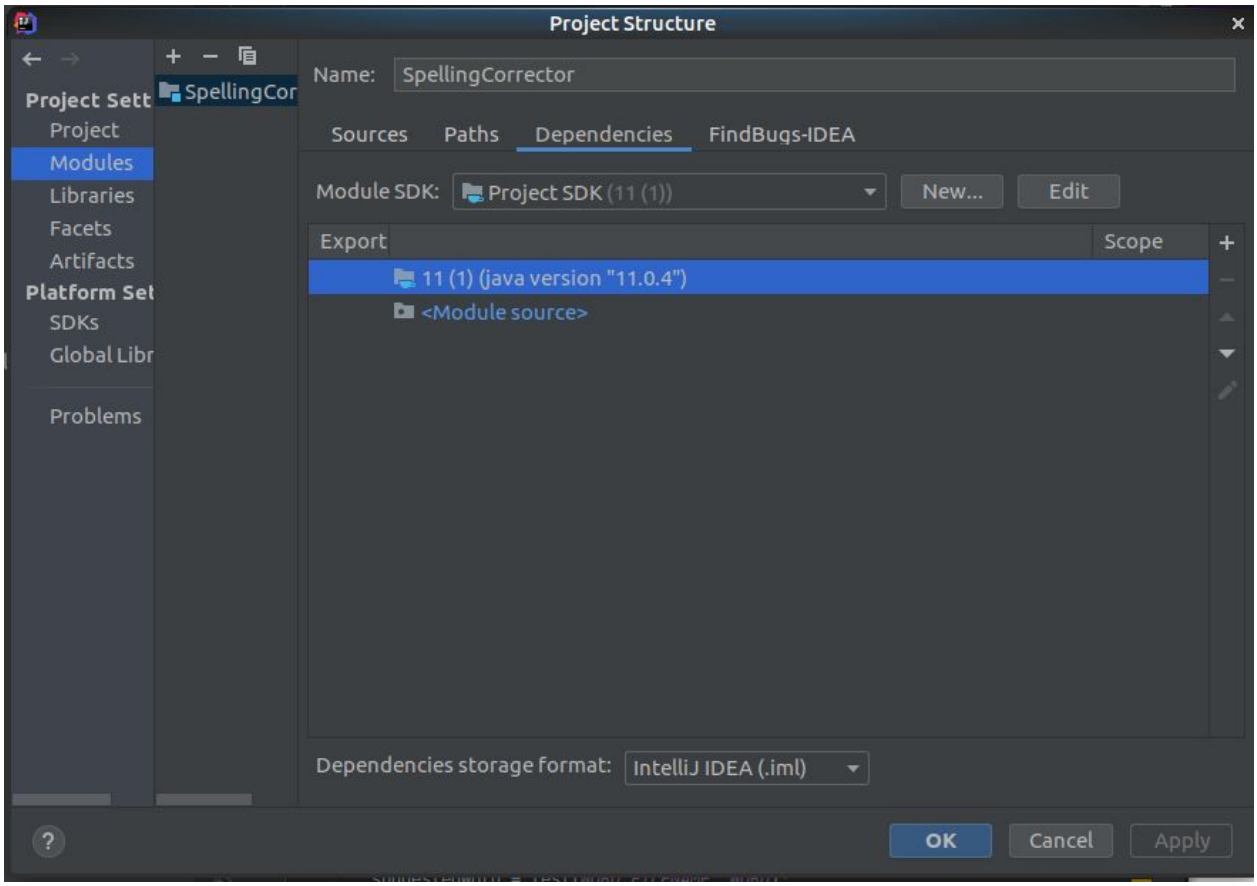
# Dependencies

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.

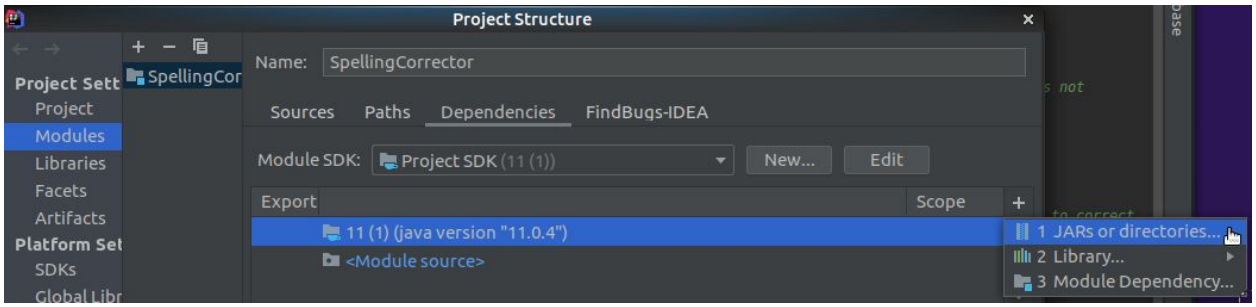
Select File > Project Structure (or use Ctrl+Alt+Shift+S) and you’ll see this screen



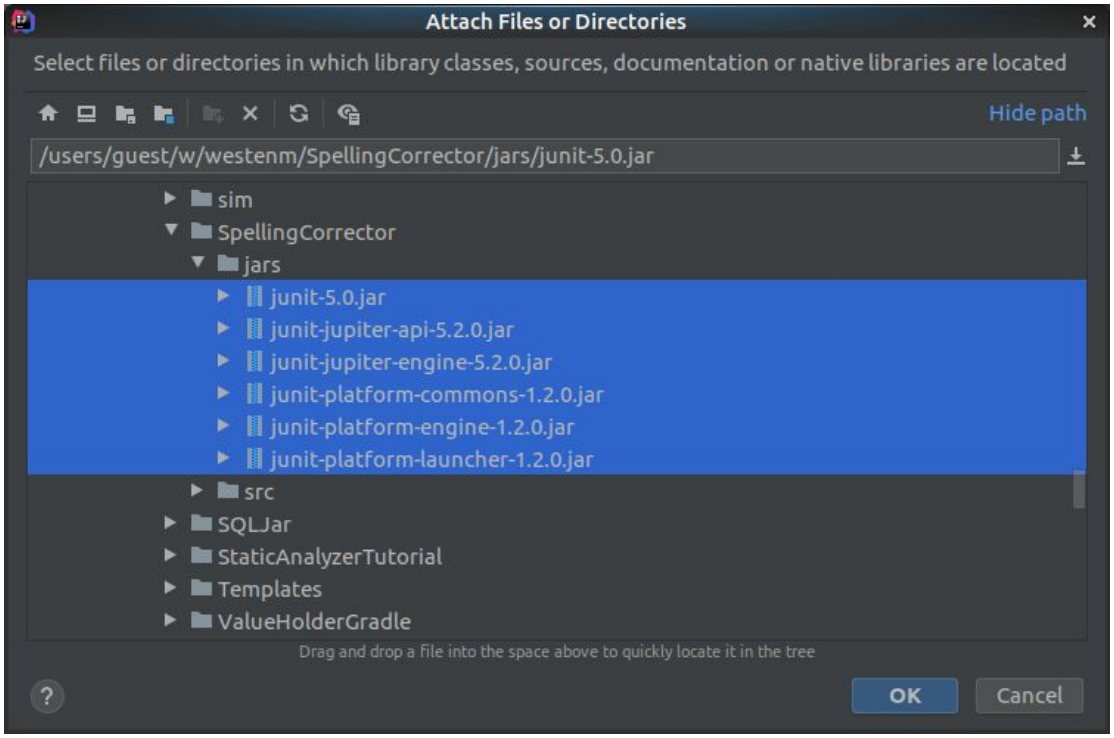
Double-click “Modules” and click the tab labeled “Dependencies”



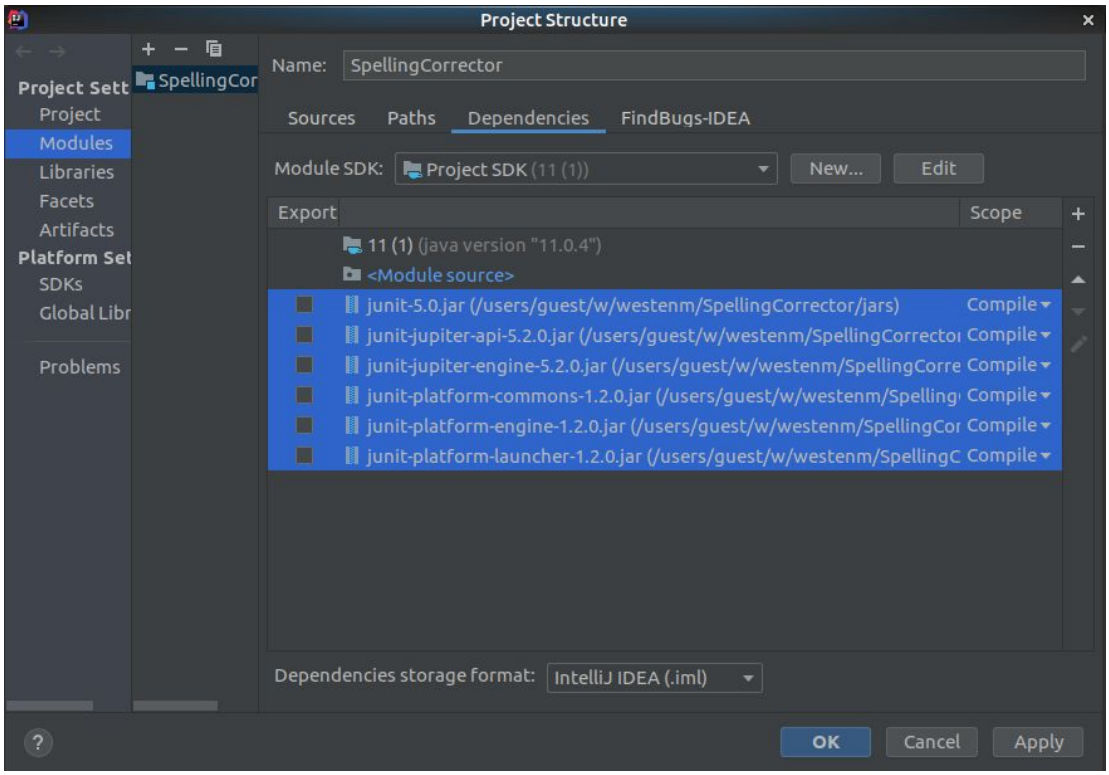

Now click the little “+” icon to the far right of the window and select “JARs or directories”



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

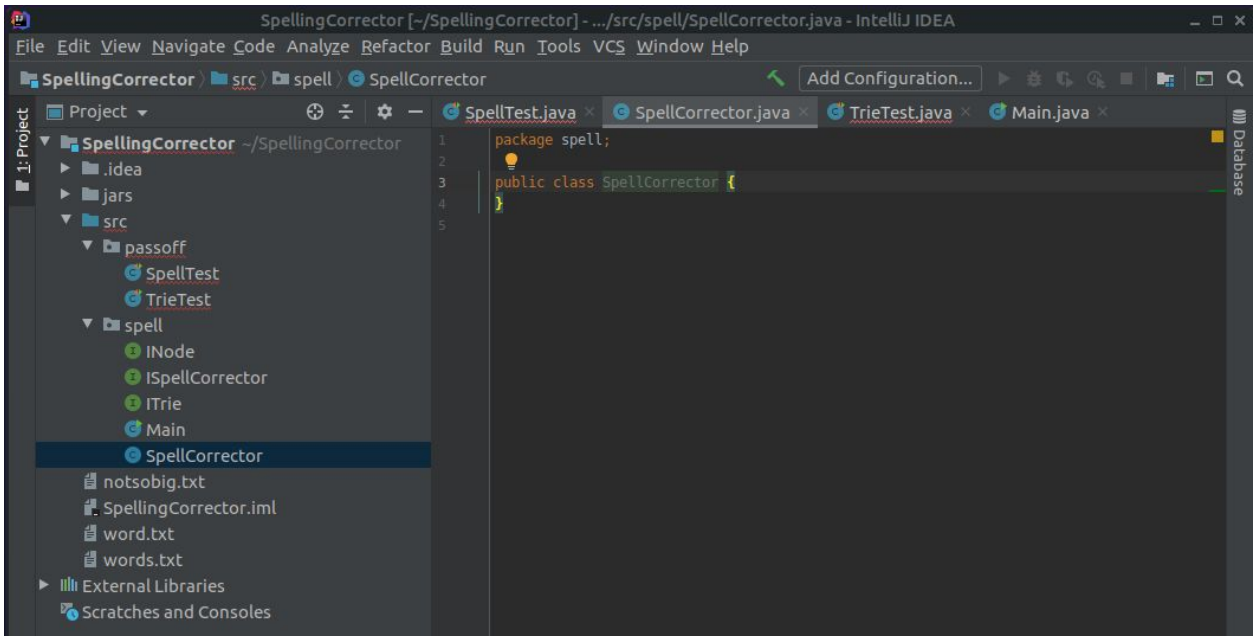



These will be added to the dependencies list. Click “OK”

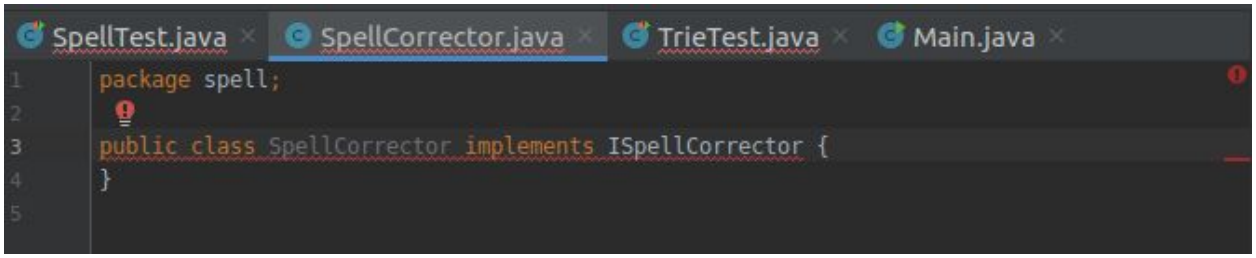


All of the errors should be gone now except for one related to *new SpellCorrector();* in SpellTest and two related to *new Trie();* in TrieTest. We'll fix those now by creating these classes.

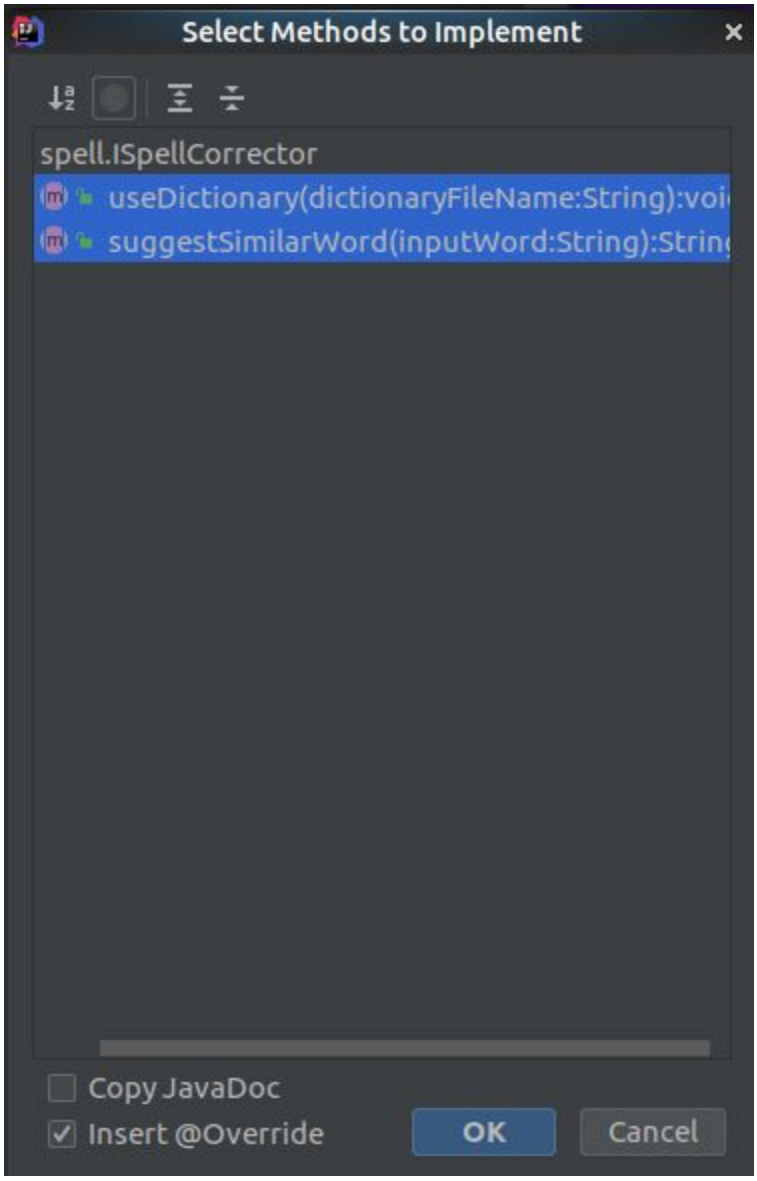
Right click on your spell package and select New > Java Class and name it SpellCorrector.




Now we'll make it implement the interface file that we have. After *public class SpellCorrector* write *implements ISpellCorrector*



You'll see an error, click on the red lightbulb or click Alt+Enter to pull up suggestions. One of the suggestions will be "Implement Methods" click this and select all the suggested methods



IntelliJ will automatically generate the needed code

```java
package spell;

import java.io.IOException;

public class SpellCorrector implements ISpellCorrector {
    @Override
    public void useDictionary(String dictionaryFileName) throws IOException {

    }

    @Override
    public String suggestSimilarWord(String inputWord) {
        return null;
    }
}
```

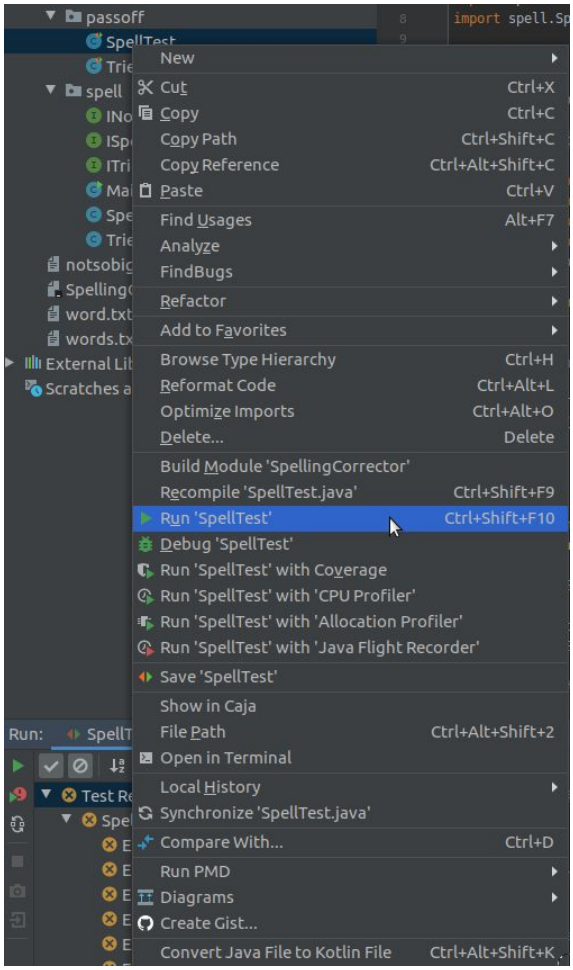
The error in SpellTest will now be gone

Now lets do the same thing for Trie Test with ITrie.

Create a new Java class and call it Trie. Type in *implements ITrie* and have IntelliJ autogenerate the needed methods.

```java
package spell;

public class Trie implements ITrie {
    @Override
    public void add(String word) {

    }

    @Override
    public INode find(String word) {
        return null;
    }

    @Override
    public int getWordCount() {
        return 0;
    }

    @Override
    public int getNodeCount() {
        return 0;
    }
}
```

All of the errors will now be gone in TrieTest and SpellTest. Now right click on SpellTest or TrieTest and select “Run Test” or click on the green arrow next to the class declaration in the file itself.




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.