IntelliJ Guide

Programming in the Large (CSSE2002)

School of EECS

The University of Queensland

Revision 1.2.0

1 What is an IDE?

An IDE is an Integrated Development Environment. It is used by programmers to improve their efficiency in writing code by integrating commonly used tools into their text editor. A typical IDE will include text editing, syntax highlighting, code completion, and compiling and executing code.

In this course we use the IntelliJ IDEA IDE, developed by JetBrains. This guide goes through the steps for getting started with IntelliJ.

2 Installation

If you are using a lab computer, IntelliJ should already be installed. Otherwise, the steps to install IntelliJ are as follows:

- In a web browser, nagivate to:

  https://www.jetbrains.com/idea/

- Click the `Download` button.
- Click the `Download` button for the Community Edition on the following page.

  Warning: the first `Download` button is for the Ultimate Edition, which is not necessary for this course. Scroll down to find the Community Edition download link.[1]

The IntelliJ installation file will now be downloaded. Follow the usual installation steps based on your operating system.

3 First Project

3.1 Opening the IDE

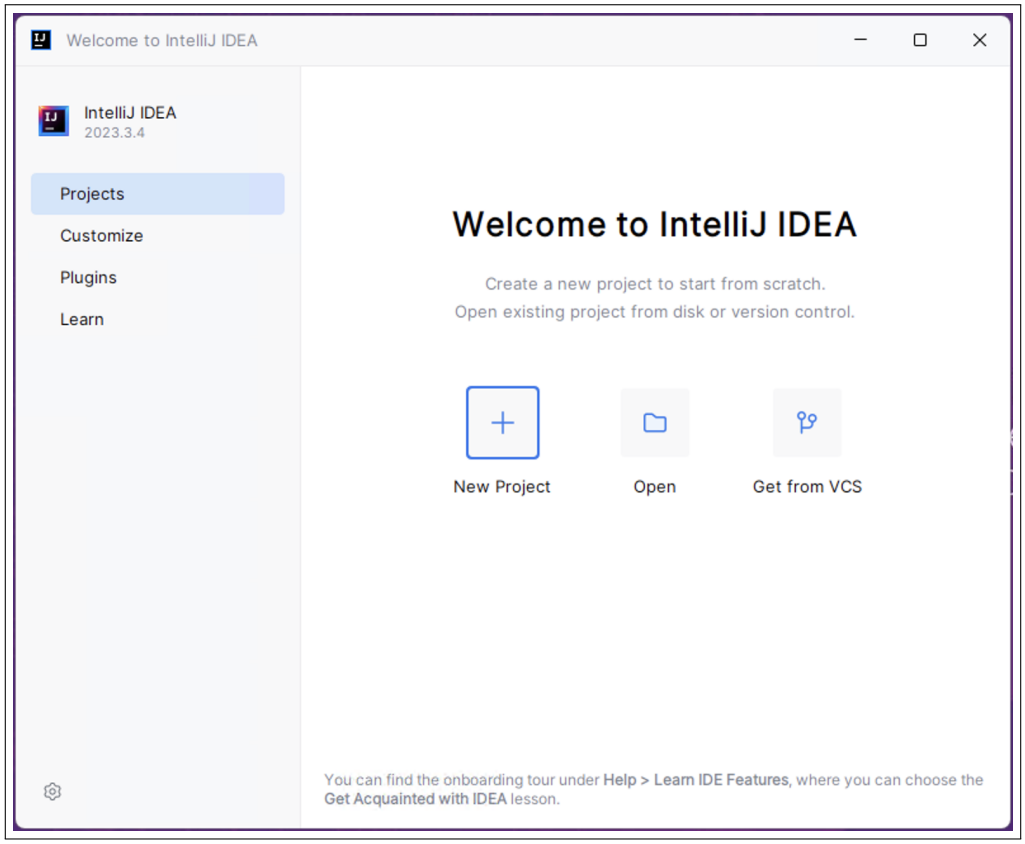
Opening IntelliJ IDEA for the first time will display a number of screens regarding importing existing IntelliJ settings (which can be skipped), agreeing to the JetBrains Privacy Policy, and sending usage statistics. Click through these steps as required.

[1] As a student, you are able to get a student licence for JetBrains products. This allows you to install the Ultimate Edition of IntelliJ at no cost. While the Community Edition will be sufficient for this course, if you are interested in using the Ultimate Edition, you can read about applying for a student licence at:

https://www.jetbrains.com/student/

3.2 Setting Up the Project

After the initial setup screens, the welcome screen should display. This is where you pick what project you would like to open. To create a blank new project, click on `New Project`.



IntelliJ should automatically detect any version of the Java Development Kit (JDK) installed on your computer. In this course, we will be using **JDK 21**.

The new project window asks you to select a name and location for the new project.

On a personal computer, the location can be anywhere you like (although try to choose somewhere sensible so that you remember where to find it at a later stage).

On the lab computers, it is important that you pick a location somewhere on your `H:\` drive (rather than your `C:\` drive) if you want your project to be permanently saved to your account and accessible from other lab computers.

Fill out the fields for your new project:

Name This is the name of the project in which you will store your Java files. Pick a sensible name, initially CSSE2002 will do.

Location Type the path, or use the dropdown folder icon to choose an appropriate folder for your project. Choose an organizational structure that makes sense to you.

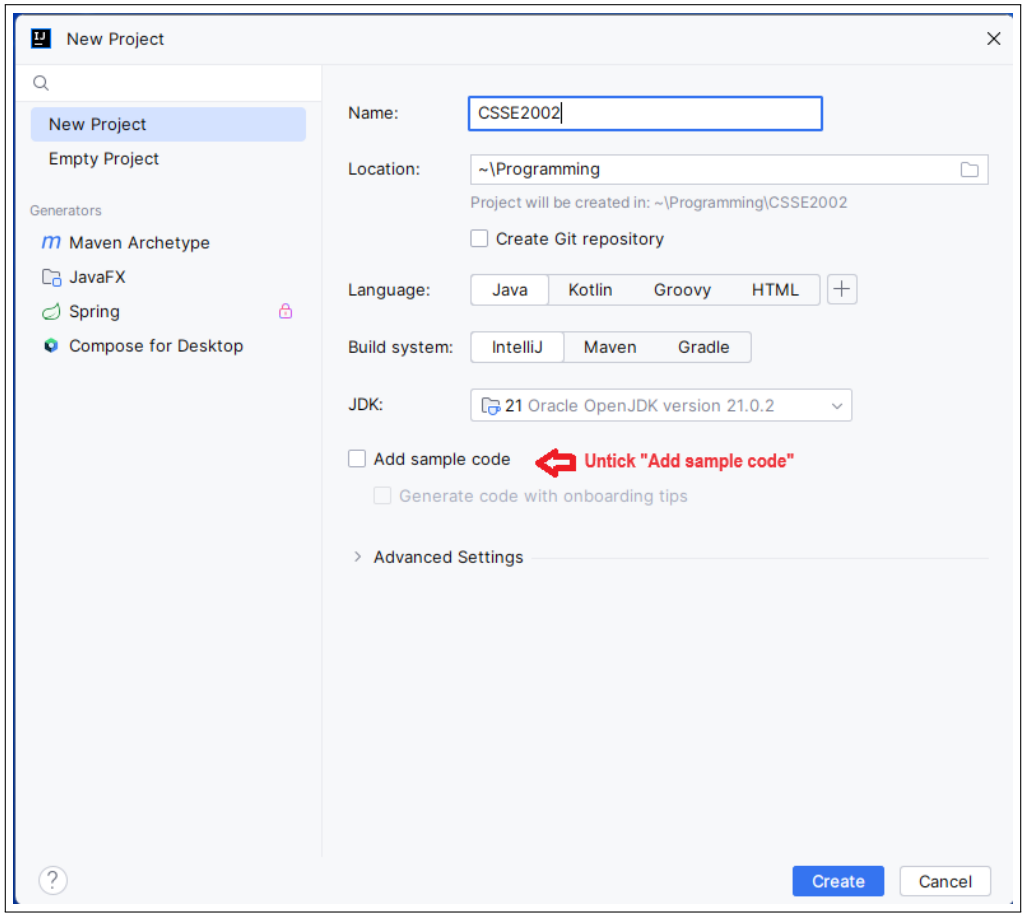
Language Select Java.

Build system Select IntelliJ.

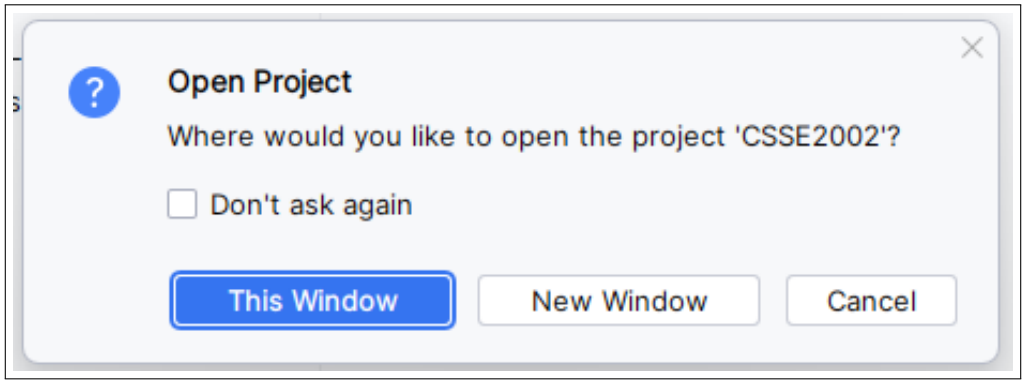
JDK Use the drop down button to select the JDK 21 version (you may have a more up to date subversion, but it should start with 21). If you do not see JDK 21, go to § *Select JDK*.

* **Untick the `Add sample code` button**

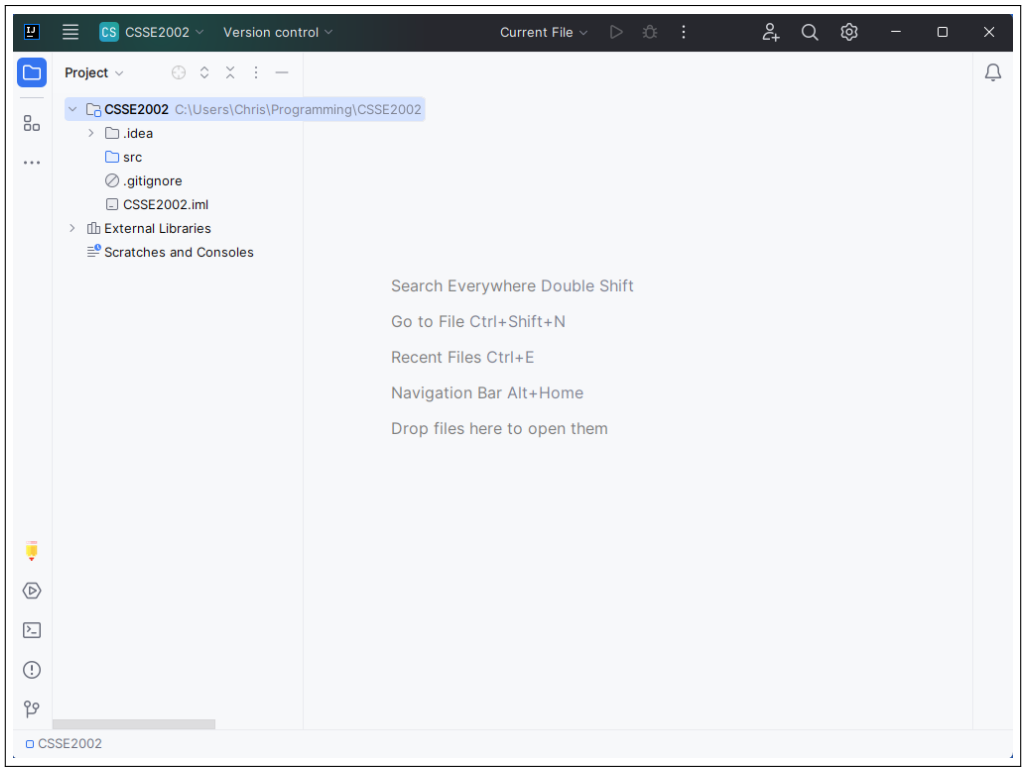
Once these fields are selected or completed, select the `Create` button to create your new project.



If you see the below popup, select the first `This Window` button.



You will then be taken to an empty CSSE2002 project that should look like this:



At this stage you can ignore these folders and files within CSSE2002:

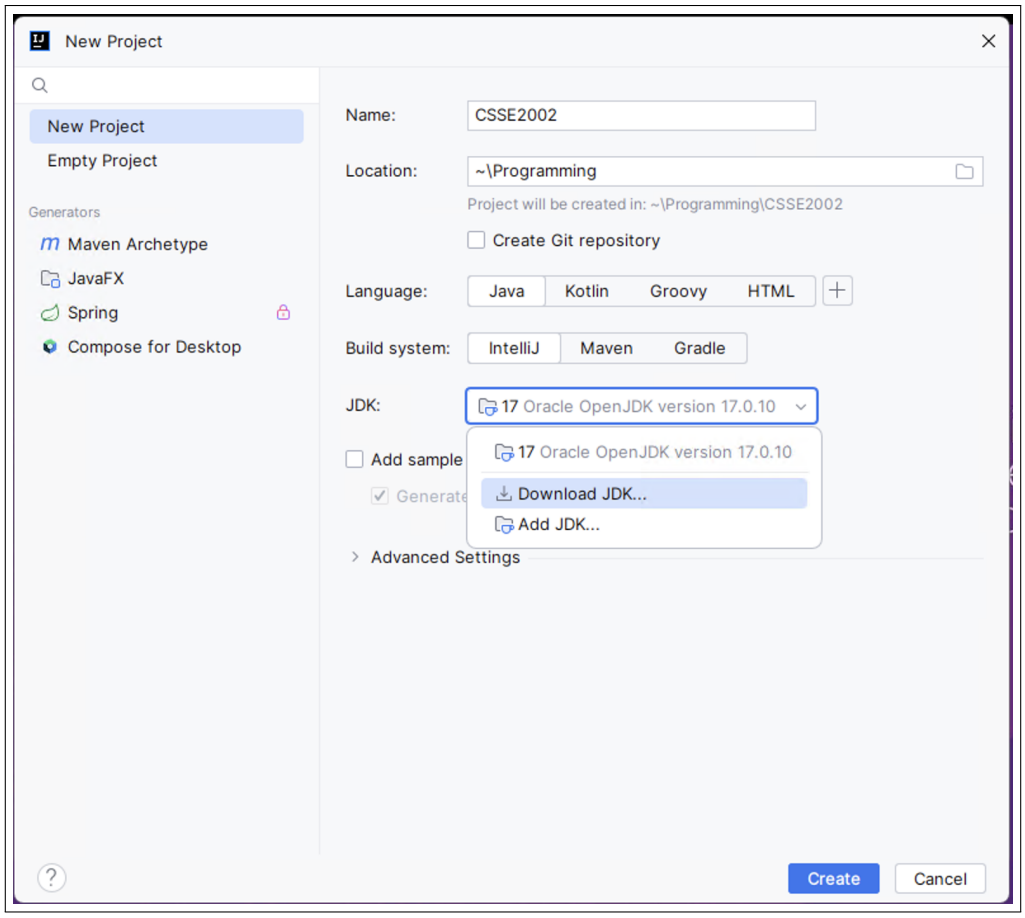
```
.idea (and its contents)
.gitignore
CSSE2002.iml
```

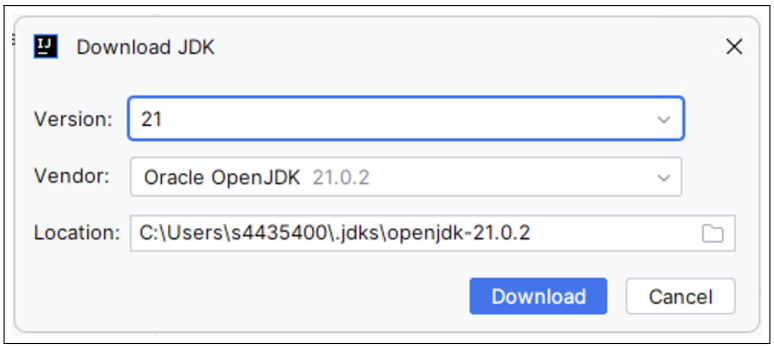
The subfolder we are interested in is the src folder, which will automatically be coloured blue.
If your src folder is not blue, please read "View Multiple Projects at the same time" below.

Your Java packages, classes and other objects will be created within this src folder (see § *Creating Packages*).

3.3 Select JDK

If you do not see JDK 21 in your dropdown, click the `Download JDK...` button. Then from the popup window, find version 21 and press `Download` to download and install JDK 21.



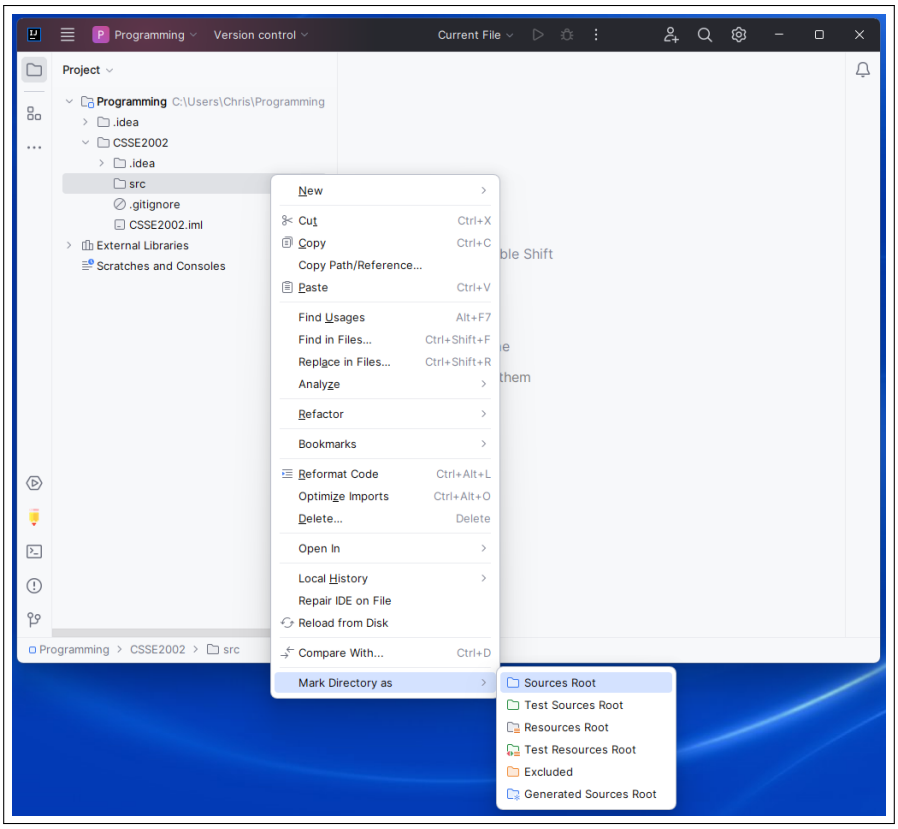


3.4 VIEW MULTIPLE PROJECTS AT THE SAME TIME

If the src folder within the CSSE2002 project is not blue (like in the image below), this means IntelliJ is pointing to a parent folder (in this case Programming, rather than CSSE2002).

If you wish to stay in the parent folder in order to access other folders outside the current CSSE2002 project, you can do so. However you will also have to mark the CSSE2002 src folder as a "Sources Root folder" in order to create Java Classes and other objects in that src folder.

To mark the src folder as a blue Sources Root folder, **right click** on the **src** folder, select `Mark Directory As` and then select `Sources Root`. The src folder will now appear blue.



The following instructions will continue as if IntelliJ is pointing directly to the CSSE2002 project, but you can also follow the same instructions from a parent folder.

3.5 CODE ORGANISATION

Several new item types can be created from within the **src** folder.

One of those types is the *package*. A package is a folder within the src folder that stores one or more pieces of Java code. You can have multiple packages in the src folder (just like having multiple subfolders on your computer), and each package can contain subpackages as well as Java code files. The types of Java code files you will commonly use are classes, instances and enums.

To create a new *package* within the *src* folder:

1. **Right click** on the **src** folder, select `New` and then select `Package`.
2. Enter a name for your package using the correct naming convention:

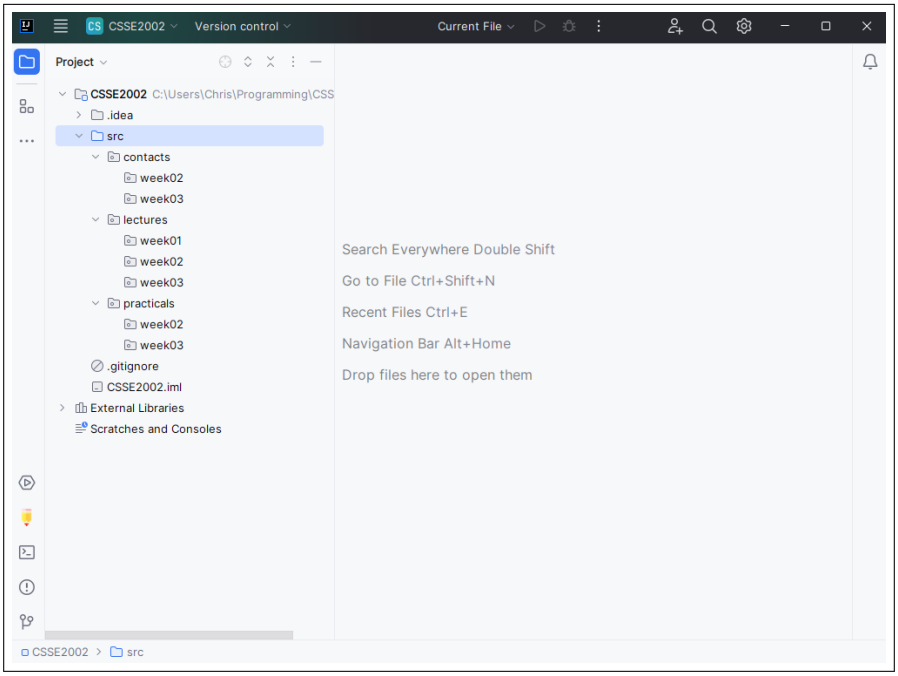
   "Package names should all be lowercase, with individual words directly concatenated together (without things such as underscores).

   ```
   package example.packages.packageName; // incorrect
   package example.packages.package_name; // incorrect
   package example.packages.packagename; // correct
   ```

   See the §3.1 of the STYLE GUIDE.

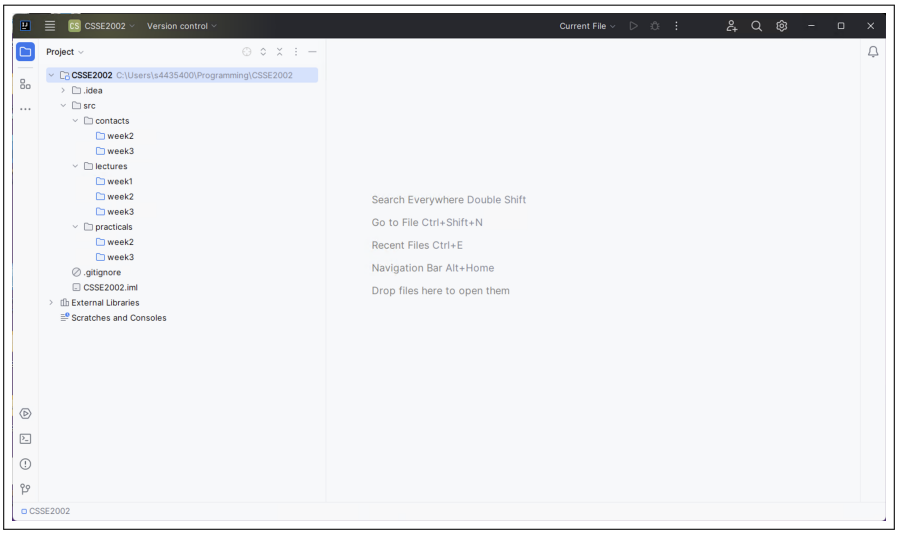
It is useful to establish an organised file structure for your code samples so that you can easily create and find code samples. A lot of code samples will be given to you as part of the lecture content, and you will also write a lot of code in this course.

There are two good ways to organise your project. One is to have a top level source root, called `src` and nest all relevant code in packages. This is demonstrated below.



This approach has the advantage the code can be re-used between different packages, e.g. you can use your lecture classes in a contact. However, it will create nested packages that may not be desirable.

The second approach is demonstrated below, this approach requires unmarking `src` as a sources root and marking the individual weeks as source roots.

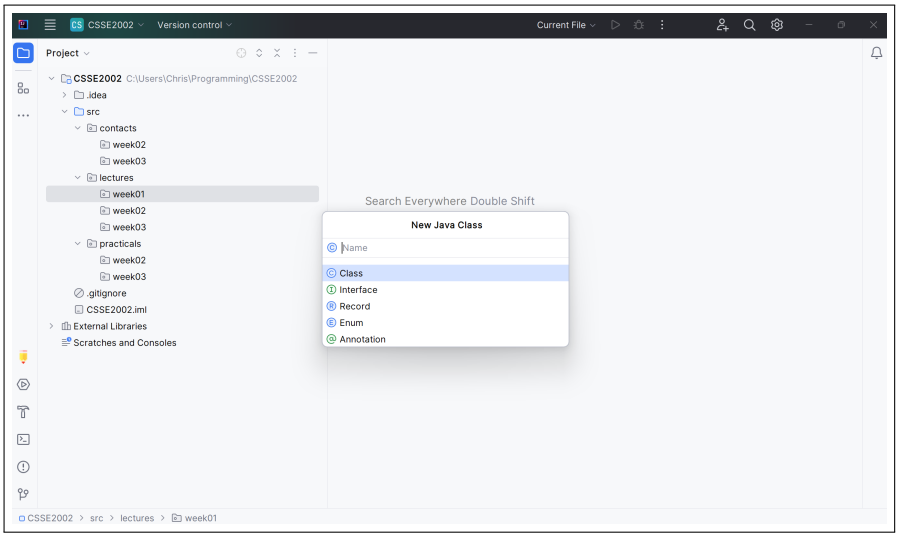


Pick an organizational structure that makes sense to you.

3.6 New Class

In order to create a new class:

1. **Right click** on the package or source root, e.g. **lectures.week1**.
2. Select `New`.
3. Select `Java Class`.
4. You will see the following popup, which has two sections:
   - The bottom section allows you to select whether you are creating a new Class, Interface, Record, Enum or Annotation, this tells IntelliJ what stub to populate in the file. For this example, we will create a new Class (which IntelliJ highlights by default).
   - The top section is where you can name your Class/Interface/Record/Enum/Annotation.



5. Keep the default selection as `Class`.
6. In the `Name` section, type the name of your new class, using the UpperCamelCase convention:

   "Files and classes in Java should be named using `UpperCamelCase`. This involves capitalising the first letter of every word in a file/class name. A correctly named file would be as followes

   ```
   MyExample.java
   ```

   with the top-level class contained within this file named the same way.

   ```
   public class MyExample {
       ...
   }
   ```

   See the §3.2 of the Style Guide.

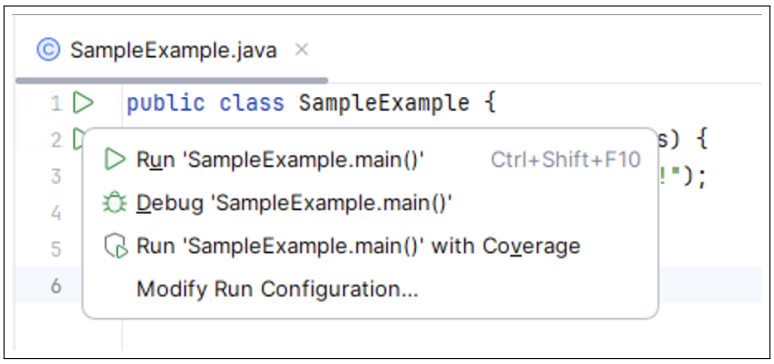
7. Press `Enter` to create your new class. You can now write code inside this Class.

3.7 Compiling and Running

In order to run your code, you will need a `main` method (this is the entry point of Java code). Add a basic `main` method which prints out "Hello, World!" as shown below to your class in order to complete the following steps.

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

Once you've added a `main` method, you will notice that green play buttons (triangles) appear to the left of your code (an area called the gutter). There are many ways to compile and run your code in IntelliJ, however pressing one of these green play buttons is one of the quickest. Pressing this button, and then pressing `Run 'SampleExample.main()'` should compile and run your code, and you should see the output of your `main` method printed to the console (the window which should appear below your code).



Frequency Asked Questions

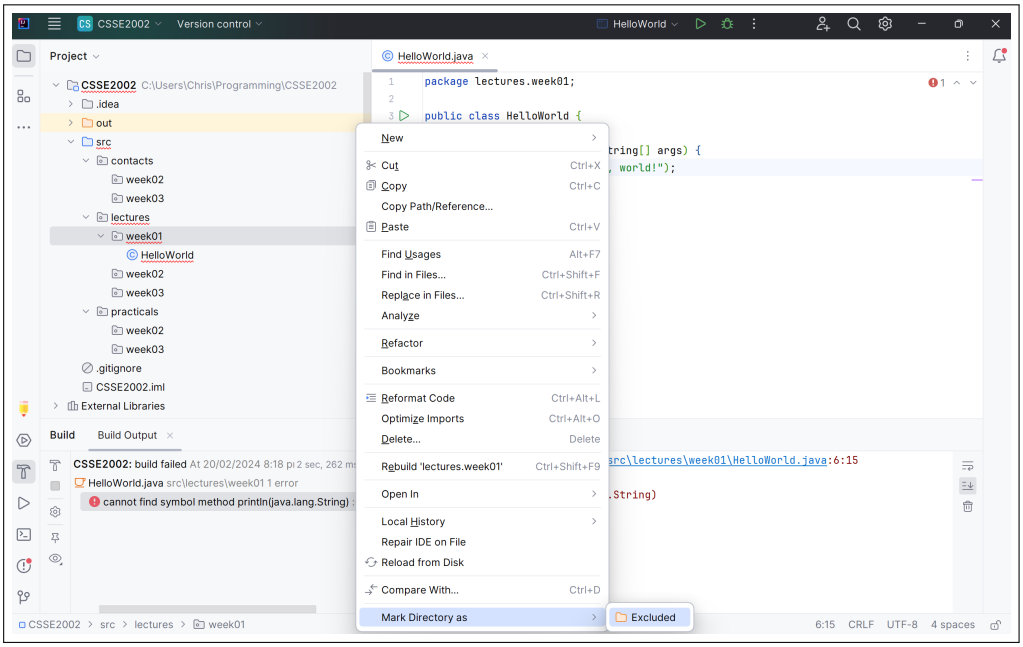
Q.1 IntelliJ refuses to compile and run a class because there are errors in other classes. What do I do?

Errors and incomplete code might exist in classes that you are not currently working on. Some of the lecture code examples may contain errors for teaching purposes, and you are not expected to immediately fix the errors when you download the code from Blackboard into IntelliJ. However in IntelliJ, having any code with errors (even if it is unrelated) can prevent you from creating, building and running code in other packages.

Solution The easiest solution is to isolate the class and its package from the remainder of the code in your CSSE2002 project.

In the example below, the `HelloWorld` class within `lectures.week01` contains a syntax error, which is throwing an error and preventing that class from being compiled and run. The `week01` package in which the class lives can be excluded with the following steps.

1. **Right click** on the package that contains the error class (in this case, `week01`).
2. Select `Mark Directory as` and then select `Excluded`.
   - The icon for that package will be changed to orange.
   - Code in other packages is not affected.
   - The excluded package and its contents will no longer interfere with other code in your project.



Reverse a Directory Exclusion When you wish to work on the code in the excluded directory, the exclusion can be reversed, so that you can modify, compile and run the classes in that directory.

1. Right-click on the excluded directory
2. Select `Mark Directory As`, and then select `Cancel Exclusion`.

The directory will no longer be excluded, and it will no longer be orange. Continue amending the classes within as required.

Acknowledgements This guide was written by Emily Bennett, Brae Webb, Richard Thomas, and Christina Russo.