SYSC 3110 – Lab 1: Discovering IntelliJ

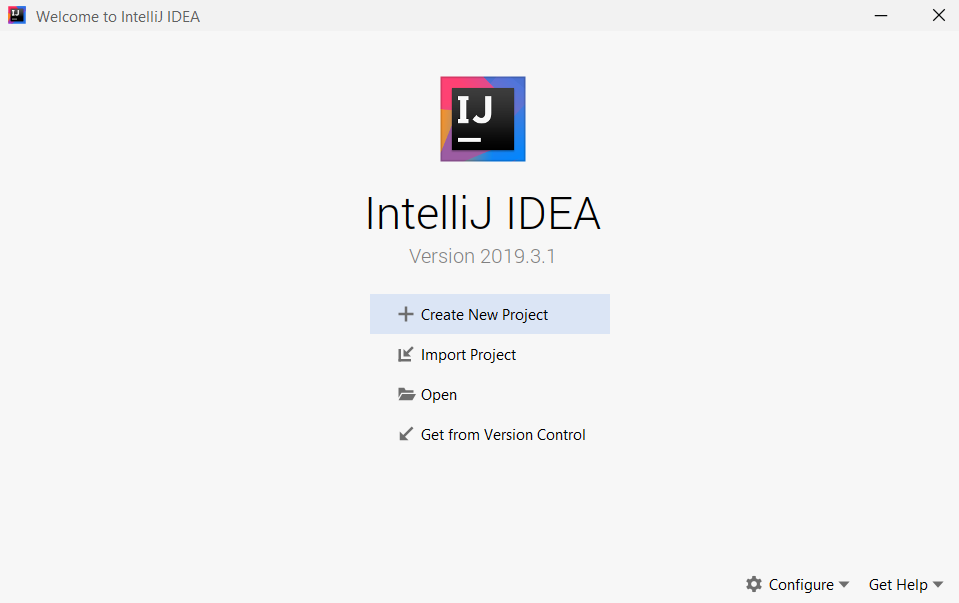
**Read this first: to successfully complete this assignment, read everything carefully, and go through all the steps. You will have to answer some questions along the way. Write those answers down and show them to the TA when you’re done to get marked.**

The goal of this first lab is to familiarize yourself with the IntelliJ environment. We will do this slowly and step-by-step.

IntelliJ is an Integrated Development Environment (IDE) developed by JetBrains. IntelliJ was designed to be easy to use and maximize your productivity so, it offers many kinds of built-in features to save your time and make coding less tedious. We will see some of these features in this lab but if you would like to see all the features visit: <https://www.jetbrains.com/idea/features/>.

Why learn to use IntelliJ? Because a good engineer must be familiar with good tools! Java Development Tools (JDT), a major component of IntelliJ, provides a great code browser, an inline debugger, as well as support for advanced concepts such as refactoring, unit testing and versioning (we will worry about these concepts later in the term…). Knowing how to use an advanced IDE will make you more productive in your development tasks, including the project you will have to work on in this course!

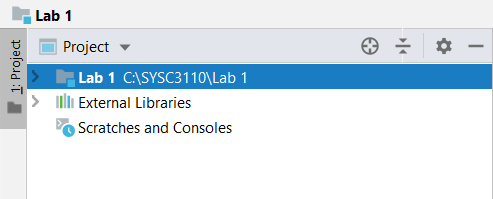
Step 1. Getting Started  
First, [download and install IntelliJ](https://www.jetbrains.com/idea/). When you first run IntelliJ, you will be prompted with a window with four options to get started on a project.



Click on Create New Project and follow the steps to create a new Java project. The first window will prompt you for the version of Java you wish to use; for now use the most recent version and click “Next”. The second window will prompt you for a template, ignore this and click “Next”. The final window will prompt you for a project name and the destination folder where the project will be saved. Name the project Lab1 and be sure to save the project somewhere you can access it because we will be re-using this lab in the future.

Step 2. Getting Familiar with the Environment

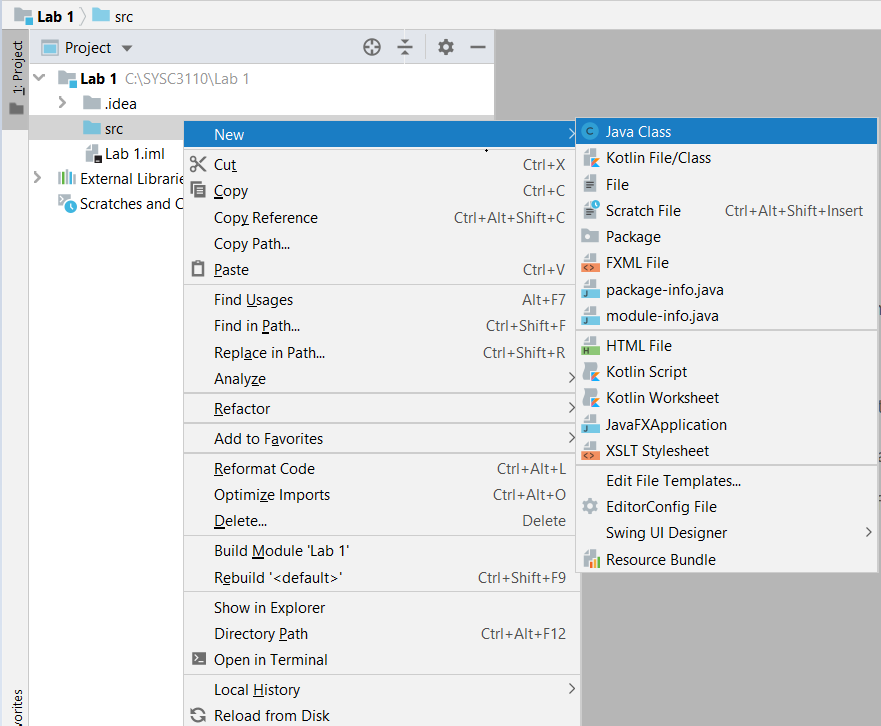
After you have created your project, take some time to mouse over the various menu items (File, Edit, Navigate…) at the top of the window to discover what each of them offers. On the left sidebar you should see the Project tool window; if you do not, click View->Tool Windows->Project or press alt+1 to make it appear.



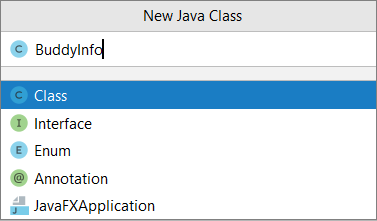
There is a “>” sign next to the folder (here called “Lab 1”) where you created the project, showing that there are some items inside. Click on it. That’s the path to the default directories and files that are used with any Java project. Write down the names of these folders and files as the first deliverable. Click the “>” sign next to External Libraries and then click the “>” sign beside the java version number. This is the path to the external java libraries that are included into your project by default. Write down three of these libraries as your second deliverable.

Step 3. Creating a New Class

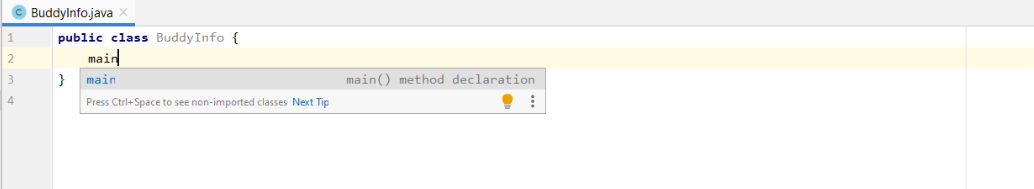
To create a new class, right-click the src folder in the project tool window and select Java class.



Name the class BuddyInfo and press enter.



Congratulations, you have made your first class in IntelliJ! Next, we will add a main method. An easy way to do this is to use the smart-completion feature. Simply type “main” and then press enter and IntelliJ will complete the signature and insert the main method for you.



Step 5. Compiling and Running

Now let’s make our code display a simple “Hello World!” for now. Type (don’t copy-paste!):

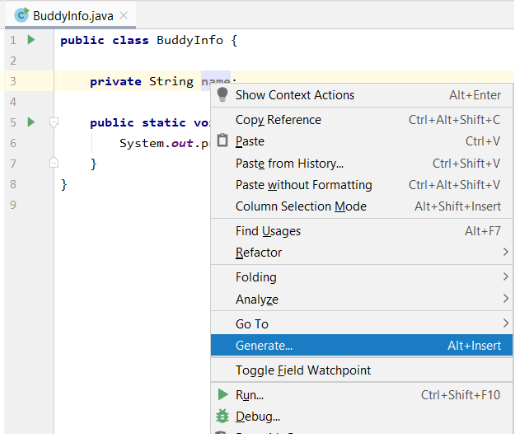
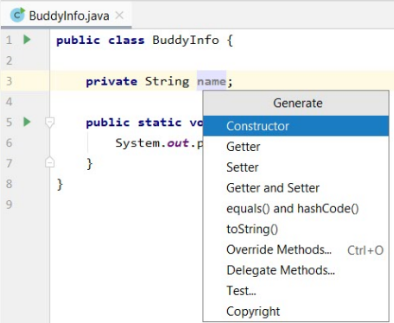
**System.out.println(“Hello world!”);**

inside the main() method. As you type, you will notice that IntelliJ provides code completion, by offering you a choice of methods you want to invoke. This is available in most modern IDEs of course. Also, IntelliJ might notice possible syntax problems, and it will highlight these problems with red font. IntelliJ can help you out with these by suggesting to import a package, create a new interface, and so on. If at this point you don’t have a compilation error (but why should you?), you can run the program. To do this, click the “Run” menu item, select “Run BuddyInfo” (You can obtain the same result by clicking on the run arrow in the toolbar or the run icon directly beside the main method and the class containing the main method). The console should appear, displaying “Hello world!” Copy-paste the code that appears in the BuddyInfo.java pane as the third deliverable.

Step 6. Further Programming and Exploration

The BuddyInfo class is supposed to contain information about a buddy of yours like, their name, address and phone number.

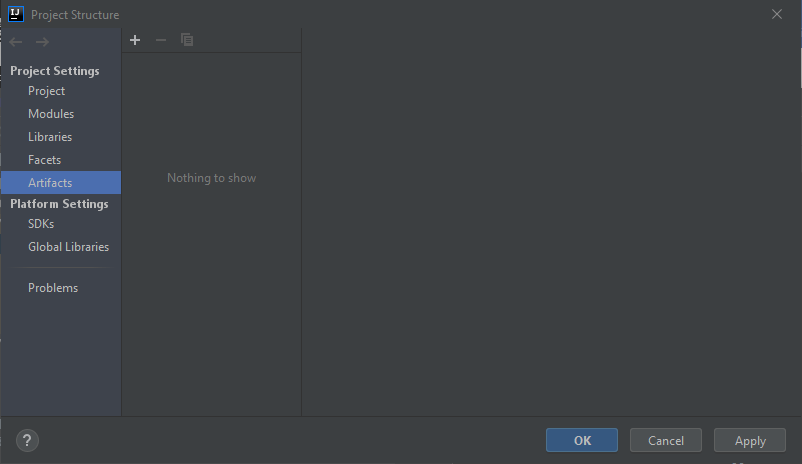
1. Create the instance variables (remember those from SYSC2004?) corresponding to those attributes.
2. Right-click on one of the instance variables and select Generate, which will display a menu of generation options (alternatively, you could highlight the field and a lightbulb will appear suggesting options including generate or, you can highlight the field and use the Generate hotkey Alt + Insert).

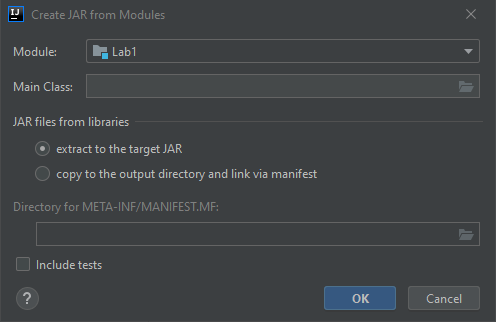
1. Generate the getter method using the Generation menu (beware though, generated code is not always adequate for your needs; it often is merely a starting point).
2. Provide constructors for your class: one that takes as parameters the values you need to correctly instantiate your class, and a default constructor that uses default values for those (but avoid code duplication!). The Generate menu may be helpful here as well!
3. Modify the main() method so that it instantiates a buddyInfo object and sets at least its name to a name of your choice, for example “Homer”, and then displays “Hello” followed by the name that you have chosen using the corresponding getter.

Step 7: SUPER IMPORTANT!!! Packaging and exporting your code

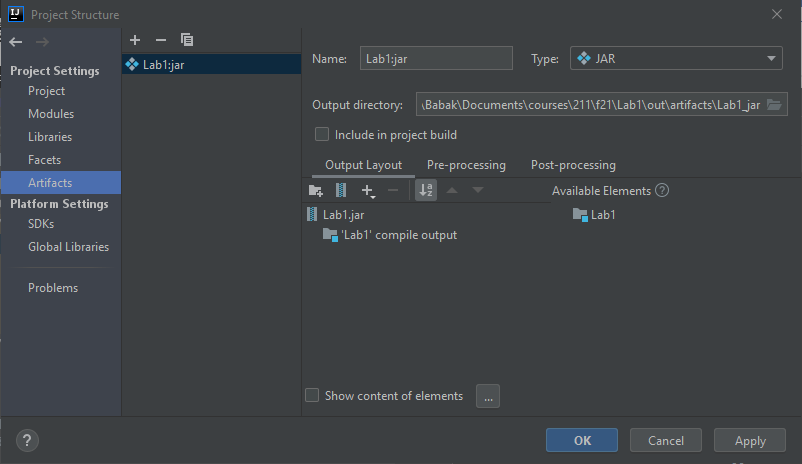
1. What you wrote in the previous section was the *source code* of class BuddyInfo. It resides in a file called BuddyInfo.java. The IDE compiled that file behind the scenes and created a file called BuddyInfo.class. That’s the file (which we also sometimes call “the executable”) that it ran! Let’s go find these files. If you right-click on “BuddyInfo” in the Project Pane, you’ll find a menu item called “Open in” then select “Explorer” (or similar if you are using an OS that is not Windows). Which file does Explorer show?(BuddyInfo.java) Open it with an editor other than your IDE. What do you see? (the code I wrote) From there, try finding the BuddyInfo.class file (you’ll need to go up from the “src” folder and see what other folders were generated for you by the IDE) . Write down the full path of where you found BuddyInfo.class. Now open this file with an editor other than IntelliJ. What do you see? That’s the Java bytecode that was generated as a result of the compilation process, and is not (usually) for human consumption.
2. So now we know that there are multiple files we may need to deliver to our client. For our assignments and projects in this course, you are always expected to show BOTH your source code and your executable! This is so that the TAs can evaluate the quality of your code (by looking at the source code), run your code (by running your executable) and make sure that your executable was indeed produced by compiling your source code. In the Java world, the best way to package all of these deliverables is to put them in a file called a JAR file. A JAR file is really not much more than a ZIP file, which may additionally contain some instruction in case someone wants to run the code by simply double-clicking on the JAR file instead of having to extract and run the executable. To create a JAR file containing your deliverables for this lab:
   1. File -> Project Structure -> Artifacts



* 1. there, click the “+” followed by “JAR” then “From modules with dependencies”. A new dialog will pop up, like so:



* 1. For “Main Class”, select “BuddyInfo”. This is to indicate that this is the class that will get executed (among potentially many candidate class files that have a “main” method) when a user simply double-clicks on the resulting JAR file. click OK.
  2. SUPER IMPORTANT: you are NOT done! Now you also need to tell IntelliJ to include the source files in the JAR file as well, which is NOT something it does by default! To do this, click the “+” under “Output Layout” in the window below, and select “Module sources” and click “OK”:



* 1. Now let’s verify that this worked! Build -> Build Artifacts will bring up a pop up with the name of the JAR it is going to create, and select “build” from that menu. The status bar at the bottom of the IDE will tell you that it “built” that JAR file for you. You can find this JAR file in the Project Pane, in out->artifacts->folder containing the JAR file. Right-click on the JAR file and select “Show in Explorer”.
  2. So now you know how to access this JAR file, so you can upload it to Brightspace! But let’s check that the JAR file indeed contains the files you intended to include. There is a command line tool that comes with the Java JDK called, unsurprisingly, jar. But here I’ll show you how this file is really nothing but a Zip file. Create a copy of your JAR file, rename the extension of the copy to “.zip”, and now you can just view what’s inside it using Explorer.

Deliverables

1. What is the name of the default folders and files that appear in the Java project folder?

.idea folder, src folder, and Lab 1.iml

1. Name three default Java libraries included in a Java project by IntelliJ.

java.compiler, java.desktop, java.management

1. Copy-paste or screenshot of the code that appears in the BuddyInfo.java pane (Hello World implementation)

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

1. The path where the file BuddyInfo.class lives

Within the folders as follows: out > production > Lab 1 > BuddyInfo.class

1. The JAR file containing your completed BuddyInfo class which has a main method that instantiates a buddyInfo object and displays “Hello” followed by the name of the buddyInfo object. This JAR file should contain BOTH the source file and the executable.

Yes, it contains both BuddyInfo.class (executable) and Buddyinfo.java (source file).