# Getting Started with Java

## Overview

In this lab you’ll create a new Java project in Eclipse, and get some practice working in the Eclipse environment. You'll also learn a bit more about the structure of Java applications.

## Roadmap

There are 6 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Creating a Java project
2. Creating a new package
3. Creating a main class
4. Adding code to input and output text messages
5. Importing the solution project
6. Additional suggestions

## Exercise 1: Creating a Java project

In Eclipse, create a new Java project named StudentGettingStarted. (Throughout this course you'll create lots of Java projects, and you'll use the naming convention StudentXxx for each project). We'll also be giving you solution projects named SolutionXxx for each lab – more on this later).

Examine the contents of your new StudentGettingStarted project. Note the following:

* The src folder is where you'll put all your Java code. The folder is empty at the moment.
* The JRE System Library folder contains all the standard classes in the Java library. If you expand this folder, you'll see a bunch of JAR files. A JAR file is a special zip-file format that contains compiled Java code. (When you've finished writing an application, you often bundle it up into JAR files to make it easier to deploy onto the target computer… it's easier to deploy a few JAR files than to deploy 100's of separate .class files!)

## Exercise 2: Creating a new package

As we mentioned briefly during the chapter, a package is a group of related Java classes. In a large application, you might have hundreds of Java classes, and you typically organize them into packages to make things easier to manage.

It's up to you how many packages you have in your application, and what you name them. For example, you might have a package named mycompany.myapp.gui that contains all the Graphical User Interface classes, a package named mycompany.myapp.db that contains all the database-related classes, and a package named mycompany.myapp.biz that contains all the business logic for the application.

With this in mind, create a new package as follows:

* Right-click the src folder, click New, and then click Package.
* In the New Java Package dialog box, type in a package name such as mypackage and then click Finish.

Verify that the src folder now has a package named mypackage. The icon looks like a package you might get from the postman ☺.

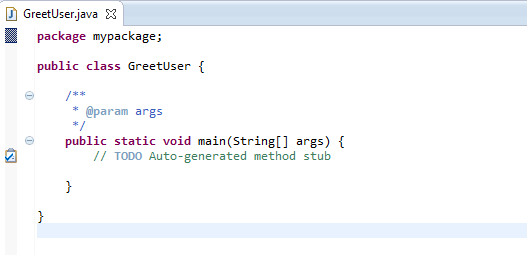
## Exercise 3: Creating a main class

Java is completely object-oriented. This means that all Java code you ever write must live inside a class somewhere. Furthermore, one of your Java classes must have a main() function – this is the starting point where the Java application will start executing.

So create a main class (i.e. a class that has a main() function) as follows:

* Right-click the mypackage node, click New, and then click Class.
* In the New Java Class dialog box, type in a class name such as GreetUser. There's a lot going on in this dialog box – we'll explore some of these details in due course. For now, all you need to do is "check" the option to create a main function. Then click Finish.

This should create GreetUser.java, and it should look like this in Eclipse:



Note that Eclipse uses colour-coding to aid readability.

* The purple words are keywords (e.g. package, public, class, static, and void). These are part of Java syntax and have a fixed meaning to the Java compiler – there are about 50 keywords altogether (e.g. if, else, while, int, long, etc.)
* The black words represent names we've chosen ourselves. For example, our package is called mypackage, our class is called GreetUser, and we have a function named main.
* The light-blue and green bits are comments, which describe what your code does (the compiler doesn't really take any notice of comments!). Java has several ways to write comments, as you can see in the screenshot. /\*\* … \*/ is called a JavaDoc comment, and it allows you to embed documentation in-line with your code. You can also use // for one-line comments, typically deep inside the code to explain a particularly tricky bit of syntax.

## Exercise 4: Adding code to input and output text messages

Add a statement inside main() to display a greeting message to the user. You can use the System.out.println() function to do this. (Notice that every time you hit the "." key, Eclipse pops up a little box showing you available options. This IntelliSense is a great help for developers, it means you don't have to remember all the details of the Java library).

Save your file. Eclipse will compile it immediately and highlight any syntax errors. Assuming the syntax is OK, right-click inside the Java file and select Run As | Java Application. You should see a message displayed in the Console window at the bottom of Eclipse.

Now you're going to add some code to ask the user to enter their name… To do this, you can make use of the System.in object (System.in represents the keyboard, in much the same way as System.out represents the output console). Unfortunately, it's surprisingly tricky to get keyboard input using System.in. The best approach is to user a standard Java helper class named Scanner. This is the syntax you need:

Scanner s = new Scanner(System.in);

There's a lot going on here, and we're not ready to explain all the syntax just yet. In a nutshell, what we're doing is creating a Scanner object to read input from the System.in device (which represents the keyboard).

If you save the file at the moment, you'll notice that Eclipse complains about the syntax. It puts red squiggles underneath the Scanner class name. Why is this? It's all to do with the way the Java class library is organized... There are thousands of standard classes in Java, and these classes are organized into packages for manageability. For example:

* The String and System classes are located in a special standard Java package named java.lang. The classes in this package are so commonplace that every Java application automatically has access to the java.lang package.
* The Scanner class is located in a different package named java.util. If you want to access any classes from java.util, you must "import" them into your code. Eclipse actually makes this easy to do… Put your mouse cursor on the Scanner class name, and Eclipse will pop up a little box showing some ways to fix the error. Click on the option to import 'Scanner' (java.util) and the error will disappear, thanks to the following statement that Eclipse adds near the top of your code:

import java.util.Scanner;

Now that you've got Scanner working, you can use it to get input from the user. Add this code:

## System.out.print("What is your name? ");

## String name = s.nextLine();

## System.out.println("Hello, " + name);

## s.close();

Run the application. It should ask you for your name, and then display it back to you ☺

## Exercise 5: Importing the solution project

During this course there will be many Java projects that you'll need to import into Eclipse, to see the demo code accompanying the chapters and to see the solution code for labs. For example, there's a solution project for this lab ☺.

Open Windows Explorer and go to the C:\OcjaDev folder. There are 3 sub-folders:

* Demos  
  Contains Eclipse projects that show demo-code for each chapter.
* Student  
  Contains Eclipse projects that contain starter code (where applicable) for some of the labs.
* Solutions  
  Contains Eclipse projects that contain the solution code for each lab. For example, there's a folder named SolutionGettingStarted that contains the solution code for this lab.

To import a Java project into Eclipse (so you can view and run the code), follow these steps:

* Invoke the File | Import menu command.
* In the dialog box, expand General, click Existing Projects into Workspace and then click Next.
* In the next dialog box, specify the following root directory to import:

C:\OcjaDev\Solutions\SolutionGettingStarted

* Then click Finish.

Eclipse should now show the solution project. Feel free to take a look at the code to see how it compares to the code you wrote in your project!

**Exercise 6 (If time permits): Additional suggestions**

* Ask the user to enter their first name and last name separately, and then display their full name back to them.
* Set some breakpoints in your code and run the application through the debugger in Eclipse. Spend some time getting used to the various Windows displayed in the Debug perspective in Eclipse.