

## LESSON 1

# GETTING STARTED WITH JAVA AND ECLIPSE

## INTRODUCTION

This lesson will introduce you to the very basics of the Java language, and how to start a new project in Eclipse (the most recent version of which as of this writing is Kepler.) If you have a slightly older version of Eclipse, such as Juno or Indigo the instructions for the development environment should be extremely similar, if not the same in many cases. As far as Java is concerned, the code will be identical.

### 1.1 WHAT IS JAVA? WHAT IS ECLIPSE?

**Java** is a **high-level programming language**. Essentially, you type **source code** to tell the computer what to do. Java is high-level, meaning the **syntax**, that is, the rules and structure of the programming statements, are closer to a **natural language** (such as English) than some **lower-level languages** like C, or assembly, or machine code.

Computers only understand **machine code**, ultimately. This essentially consists of the zeroes and ones (0 and 1) that the computer processes as on and off signals. However, humans are very bad at memorizing long strings of 0s and 1s and therefore it's more helpful if we can use a language closer to our own. Java is one of these high-level languages.

**Eclipse** is one of the most popular **IDEs (integrated development environments)** for Java. It is used extensively by professional Java developers in all different industries and academia. It is also well-suited for mobile developers developing for Android, and appropriate add-ons can be installed to enable Android development with relative ease. Since Eclipse is **open-source**, meaning its own source code is openly available to be modified, many companies have made custom versions of Eclipse that suit their in-house purposes.

Other popular IDEs include **Netbeans, jGRASP, IntelliJ IDEA, and Jdeveloper**.

## IDEs, COMPILERS, LINKERS, AND DEBUGGERS

An IDE typically consists of at least the following components:

- Compiler
- Linker
- Debugger

The reason we call it an integrated development environment is because it consists of a **compiler**, which in Java converts source code to Java **bytecode**, which is sort of like machine code for the **JVM (Java Virtual Machine)**. Java is different from many other languages for many reasons, but most notably that it runs on so many different

platforms. The programmers of the Java language itself have created many different JVMs for different platforms (MacOS, Linux, Windows, etc.)

These JVMs are designed to execute the bytecode on the native system. Many language convert source code **directly** into the machine code of the machine the program is on. But this reduces **portability**, in other words, if you compile a C++ program on Windows 64 bit and try to run the executable on MacOS machine, it will not run (unless there is some sort of emulator or interpretation environment on the host system.)

Truth be told, Java is not considered solely a **compiled language**, like C, C++, etc. It is also not solely an **interpreted language** like JavaScript, PHP, Perl, etc. It is actually considered a **hybrid language** because it has features of both compiled and interpreted languages.

After the code is compiled from Java source code into bytecode, it is run on the Java Virtual Machine either by being interpreted, which is usually much slower during execution, or using a more modern approach, **JIT compilation (Just-in-time compilation)**, which compiles the code right before it runs. So loading the program might be slower, but the execution is much faster than a strictly interpreted bytecode.

Often, other libraries of code are pulled in or **linked** to the source code being compiled. For example, a lot of work has been done in libraries that support graphical user interface components such as text boxes, drop down lists and labels, but your program won't know how to use these unless the code another developer wrote is linked in.

Another feature of an IDE is the **debugger**. A debugger allows you to step through a program more slowly and see the state of memory and execution so you can determine where an error (called an *exception*) or some other problem is occurring. This helps significantly in fixing and optimizing code.



## 1.2 GETTING STARTED WITH ECLIPSE

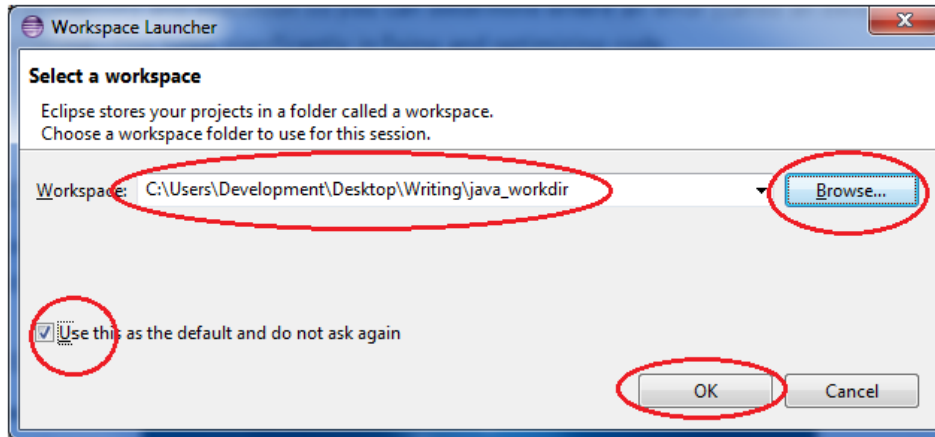
In this section, we'll start a new project in Eclipse. Examples will be in the Kepler version of Eclipse, but should work on most other modern versions with minimal differences.

### 1.2.1 CREATING A NEW PROJECT IN A WORKSPACE

A **workspace** is a dedicated folder on your file system that holds one or more **projects**, which are collections of source code and **resources**, such as images, video, fonts, files, and other non-code data.

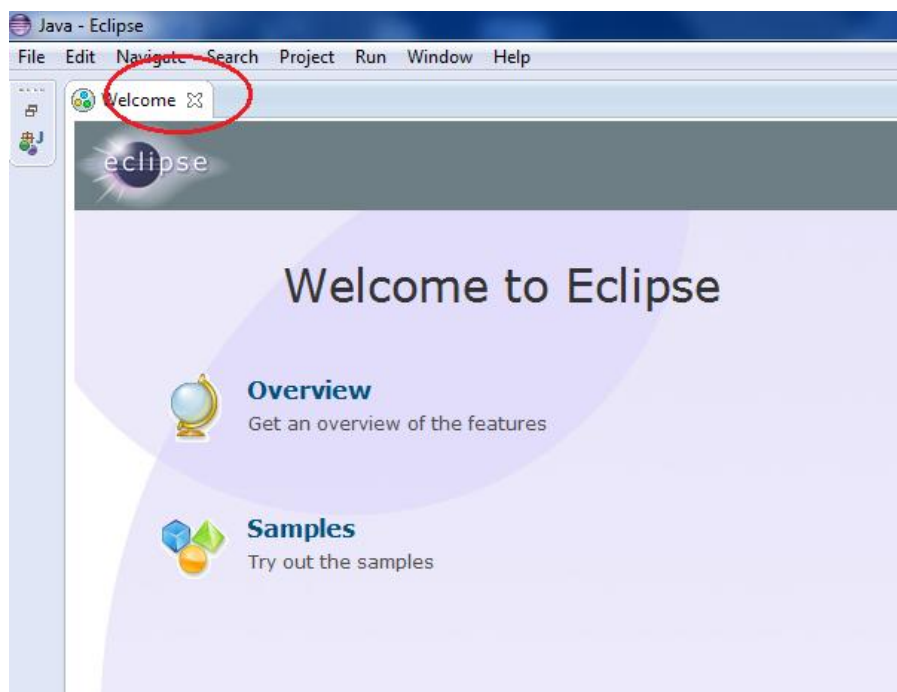


1. Click the Eclipse Icon, which should look like  or .
2. The first time you run Eclipse, it will ask you to select a workspace. You should create a folder somewhere on your computer where you want to keep your Java projects.

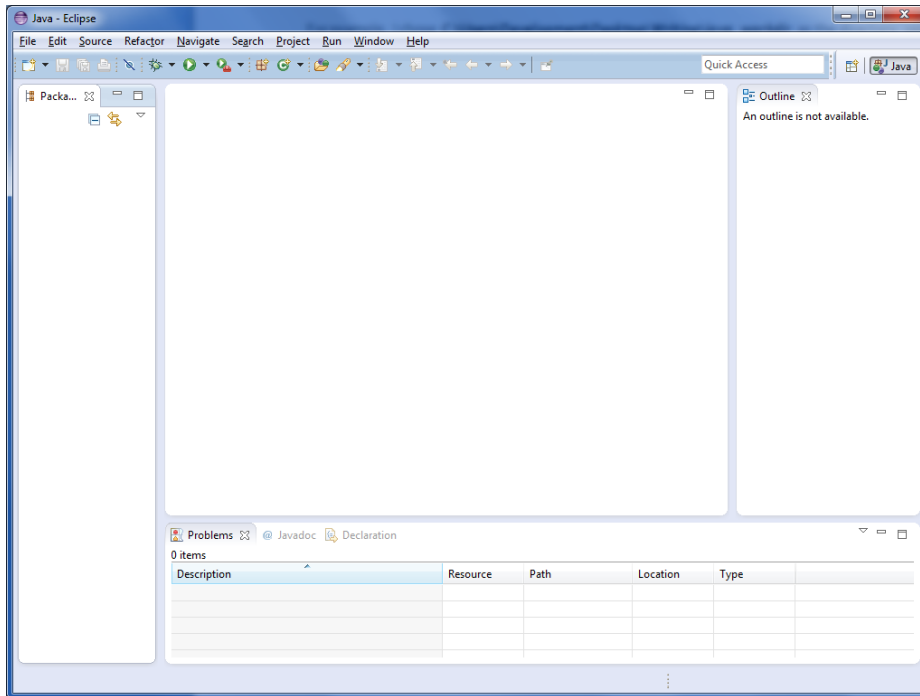


For example, I chose `C:\Users\Development\Desktop\Writing\java_workdir` as the directory, using the **Browse...** button. Also, make sure to check the “Use this as the default and do not ask again” option.

3. Hit the **OK** button
4. Click the X on the default start screen unless you want to read some of the information provided



5. Now you have an empty workspace for your projects.



To the far left, you will see the **Package Explorer**. This will be where your packages will be listed. It is one of many **views** in Eclipse that make your life easier. Down at the bottom is where code problems, known as **syntax errors** will appear, as well as where some of your Java applications will be executed.

The big white area in the middle is the space where your **editor** will go. This is where you actually type the code.

Now that we have an empty workspace, we can write and execute our first program in Java!

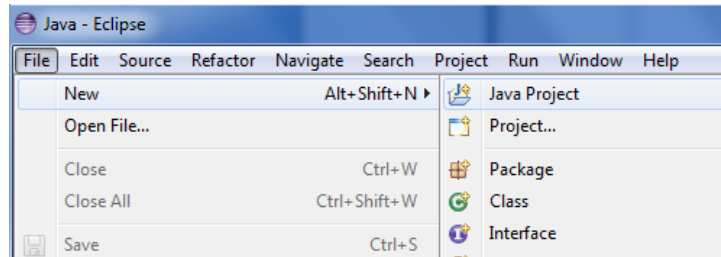
### 1.2.2 CREATING OUR FIRST PROJECT

Each **project** will contain the source code for a program we are working on. The first program (and first few programs, in fact) we will work on will be very simple **console applications**, meaning they don't have a **GUI (graphical user interface)**.

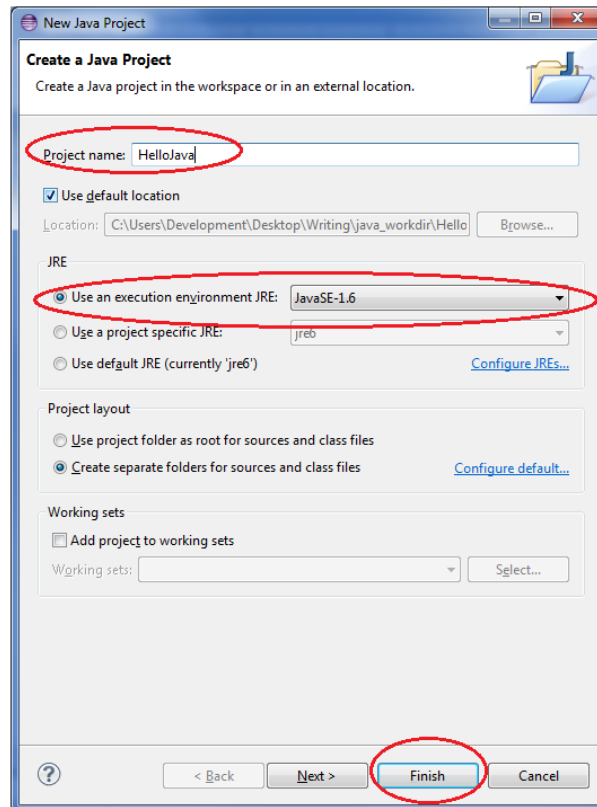
They are just text-based interfaces where the user of the program can read the output, and sometimes type input into the program.

So first, we must create a project, and then we write the code.

1. Go to File → New →

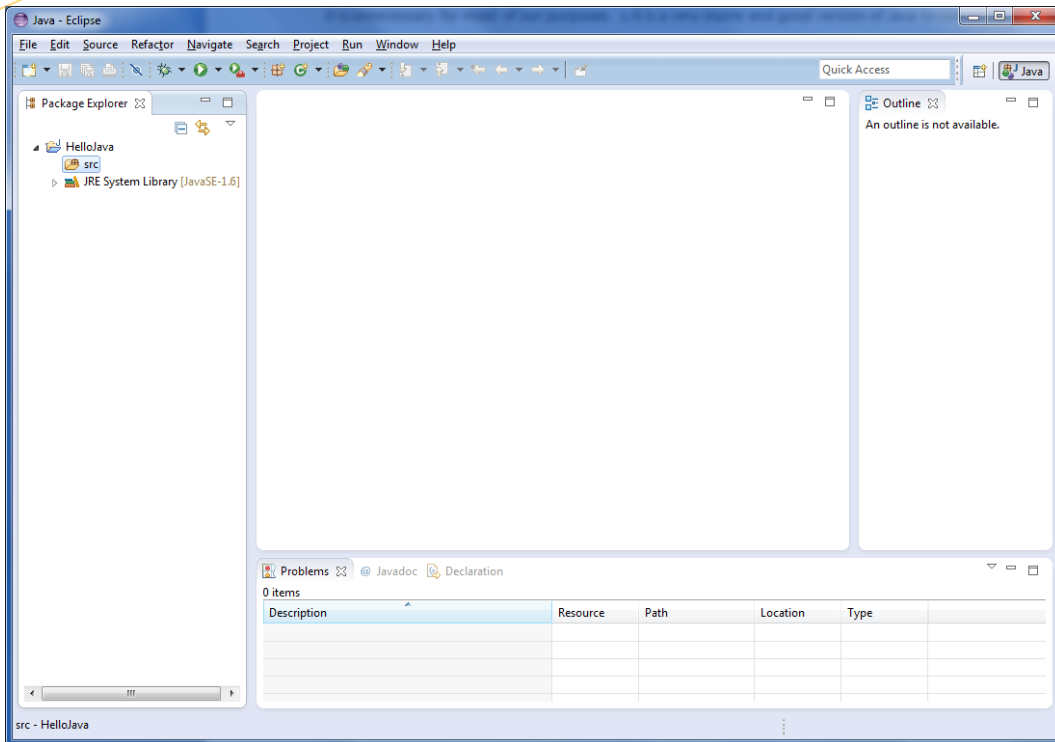


2. Type a useful name for the project, and ensure that it is using a newer version of Java such as version 1.6. That should be the version that is default with the installation of Kepler. You could use 1.7, but to be quite honest it is unnecessary for most of our purposes. 1.6 is a very stable and good version of Java to use.

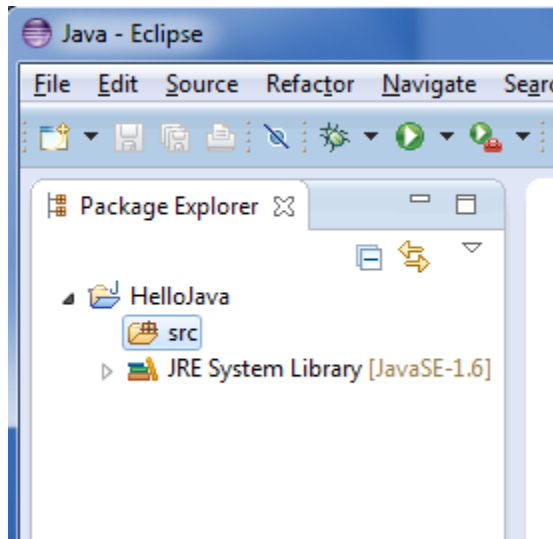


3. Click the **Finish** button

You should now click the little triangle next to the HelloJava icon in the Package Explorer and you should see how there is a src (source) folder and an additional listing of the JRE System Library for Java 1.6. This is so our program has access to all the methods, constants, variables, and other features of the Java 1.6 runtime environment.



Zoomed in this looks like the following:



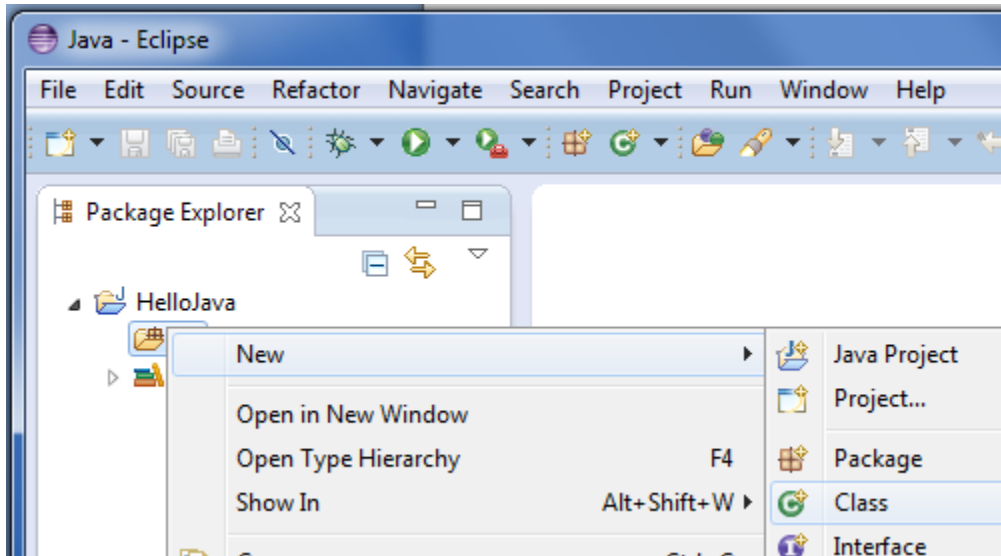
Notice there are currently no source files under the src directory. We are going to add one next.

### 1.2.3 ADDING A SOURCE FILE AND WRITING OUR FIRST PROGRAM: HELLO, JAVA!

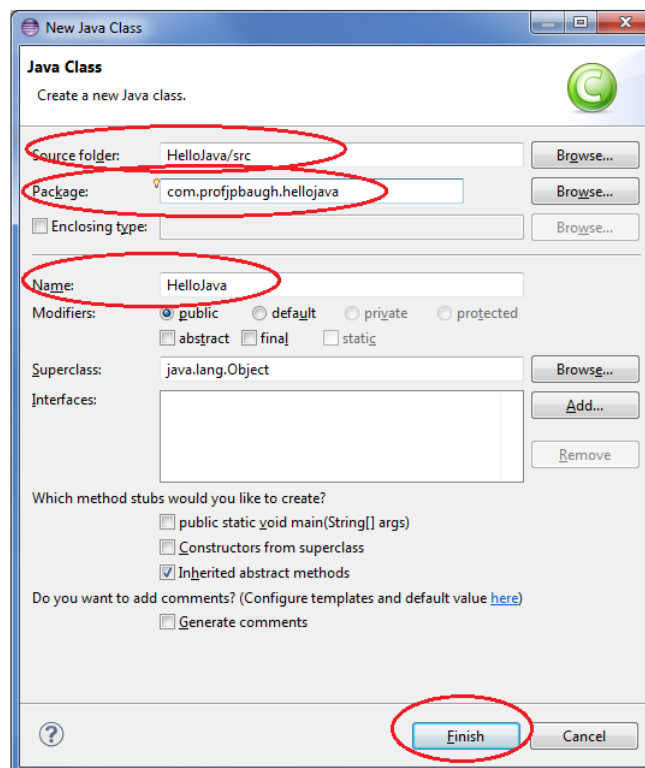
In this section, we'll add the source file and write a little bit of code.

1. **Right-click** the `src` folder under Package Explorer, and go to **New→Class**

Alternatively, you can go to **File→New→Class**



2. Give the class the name HelloJava



3. Verify that the Source folder is the src folder of our HelloJava project
4. Give your project a **package name**. The package name is somewhat arbitrary, but typically, developers use **reverse URL notation**. It's kind of like a website, but in reverse. It doesn't really matter if you have a website at the location you specify. However, since I do own **profjpbaugh.com**, I use the notation **com.profjpbaugh.PROJECTNAME**, in this case, the project name is **hellojava**.

So, in total, my package name is **com.profjpbaugh.hellojava**

5. Click the **Finish** button

You should now see the code in the editor in the middle for HelloJava.java:

```
package com.profjpbaugh.hellojava;  
  
public class HelloJava {  
  
}
```

6. Add code so that the code looks like the following:

```
package com.profjpbaugh.hellojava;  
  
public class HelloJava {  
  
    public static void main(String[] args)  
    {  
        System.out.println("Hello, Java!");  
    }  
}
```

The **method**, `main` is a special method known as the **entry point** to a Java application. We will learn how to write other methods later on, but this one is essential. When you run your application, this is the method that is **called**, or **invoked**, first. Thus, anything that happens in `main` is what your code will do.

We call another method called `System.out.println` which is for printing a **string of characters**, such as a name or a sentence or even a full paragraph, and then an **endline character**, which ensures the **insertion point** goes to the line under what was just printed. This is important when we have multiple lines of input. Not surprisingly, **println** stands for "print line."

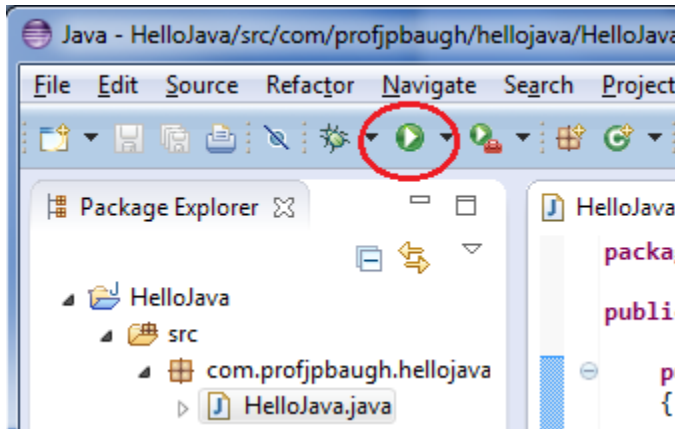
Notice the syntax. Inside the parentheses, we write a string of text enclosed in double quotes, `" "`. Also, notice the semicolon at the end, `;`, which indicates the end of a statement in Java.

Now we're ready to run our application!

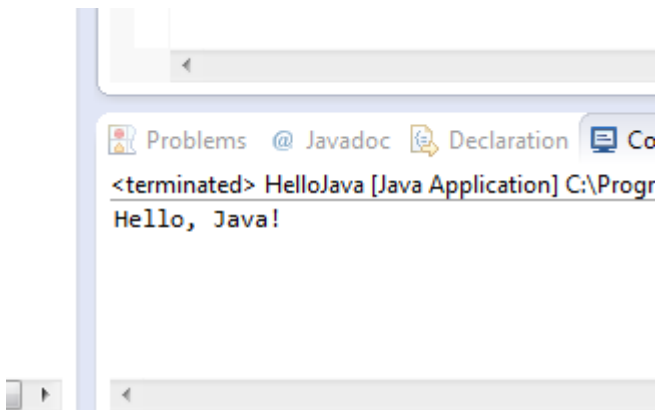


### 1.2.4 RUNNING AN APPLICATION IN ECLIPSE

1. Typically, you can just click the **Run Button** near the top of the workbench



2. View the bottom of the workbench:



And that's it! It shows the output from our program and the end of our first lesson!!

## EXERCISES

1. Make sure you know how to create a project from beginning to finish, giving it a unique name, adding a source code (Class) to the project with a unique package name, as well as running the application.
2. Write a program that prints out two lines saying "Hello, Java!" and "I love Java" on two separate lines

[10]

3. Change the string that is printed out to your name, and say, "Hello my name is John and I'm 30", replacing John with your name, and 30 with your age.