To get started with java

1. If not already done download and install the jdk.

<https://www.youtube.com/playlist?list=PLFE2CE09D83EE3E28>

See tutorial 1

2. Test this by running a sample program.

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world!");

}

}

Javac Hello.java

Java Hello

See tutorial 2 <https://www.youtube.com/playlist?list=PLFE2CE09D83EE3E28>

3. Install Eclipse 4.8 (Photon) for Java Developers

##### see tutorial 3.

##### Step 1: Download

Download Eclipse from [https://www.eclipse.org/downloads](https://www.eclipse.org/downloads/). Under "Get Eclipse Photon" ⇒ Click "Download Packages". For beginners, choose the 4th(?) entry "Eclipse IDE for Java Developers" and "Windows 64-bit" (e.g., "eclipse-java-photon-R-win32-x86\_64.zip" 195MB) ⇒ Download. (In later years you will probably need other versions of eclipse, remember you can have several versions on the same machine.)

##### Step 2: Unzip

To install Eclipse, simply unzip the downloaded file into a directory of your choice (e.g., "c:\myproject").

There is no need to run any installer. You can move or rename the directory. You can install (unzip) multiple copies of Eclipse in the same machine or even on a pen drive.

### Writing your First Java Program in Eclipse

##### Step 0: Launch Eclipse

1. Launch Eclipse by running "eclipse.exe" from the Eclipse installed directory.
2. Choose an appropriate directory for your workspace, i.e., where you would like to save your files (e.g., H:\myproject\eclipse for Windows).
3. If the "Welcome" screen shows up, close it by clicking the "cross" button next to the "Welcome" title.

##### Step 1: Create a new Java Project

For each Java application, you need to create a project to keep all the source files, classes and relevant resources.

To create a new Java project:

1. Choose "File" menu ⇒ "New" ⇒ "Java project" (or "File" ⇒ "New" ⇒ "Project" ⇒ "Java project").
2. The "New Java Project" dialog pops up.
   1. In "Project name", enter "FirstProject".
   2. Check "Use default location".
   3. In "JRE", select "Use default JRE (currently 'JDK10.0.x')". But make sure that your JDK is 1.8 and above.
   4. In "Project Layout", check "Use project folder as root for sources and class files".

Push "Next" button.

1. Uncheck "Create module-info.java file" box (if it is checked) ⇒ Finish.

##### Step 2: Write a Hello-world Java Program

1. In the "Package Explorer" (left pane) ⇒ Right-click on "FirstProject" (or use the "File" menu) ⇒ New ⇒ Class.
2. The "New Java Class" dialog pops up.
   1. In "Source folder", keep the "FirstProject".
   2. In "Package", delete the content if it is not empty.
   3. In "Name", enter "Hello".
   4. Check "public static void main(String[] args)".
   5. Don't change the rest.

Push "Finish" button.

1. The source file "Hello.java" opens on the editor panel (the center pane). Enter the following codes:

public class Hello {

public static void main(String[] args) {

System.out.println("Hello, world!");

}

}

##### Step 3: Compile & Execute the Java Program

1. There is no need to compile the Java source file in Eclipse explicitly. It is because Eclipse performs the so-called incremental compilation, i.e., the Java statement is compiled as and when it is entered.
2. To run the program, right-click anywhere on the source file "Hello.java" (or choose "Run" menu) ⇒ Run As ⇒ Java Application.
3. The output "Hello, world!" appears on the Console panel (the bottom pane).

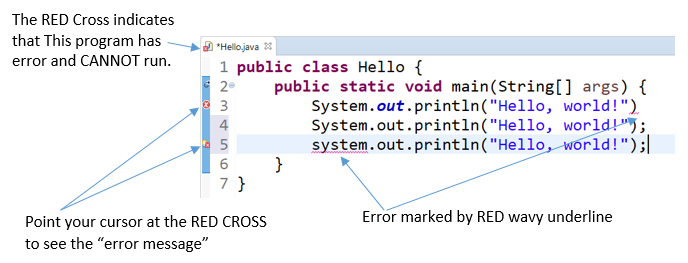
NOTES:

* You should create a NEW Java project for EACH of your Java application.
* Nonetheless, Eclipse allows you to keep more than one programs in a project, which is handy for writing toy programs (such as your tutorial exercises). To run a particular program, open and right-click on the source file ⇒ Run As ⇒ Java Application.
* Clicking the "Run" button (with a "Play" icon) runs the recently-run program (based on the previous configuration). Try clicking on the "down-arrow" besides the "Run" button.

#### Correcting Syntax Errors

Eclipse performs incremented compilation, as and when a source "line" is entered. It marked a source line having syntax error with a RED CROSS. Place your cursor at the RED CROSS to view the error message.

You CANNOT RUN the program if there is any syntax error (marked by a RED CROSS before the filename). Correct all the syntax errors; and RUN the program.



HINTS: In some cases, Eclipse shows a ORANGE LIGHT-BULB (for HINTS) next to the ERROR RED-CROSS (Line 5 in the above diagram). You can click on the LIGHT-BULB to get a list of HINTS to resolve this particular error, which may or may not work!

SYNTAX WARNING: marked by a orange triangular exclaimation sign. Unlike errors, warnings may or may not cause problems. Try to fix these warnings as well. But you can RUN your program with warnings.

### Debugging Programs in Eclipse

Able to use a graphics debugger to debug program is crucial in programming. It could save you countless hours guessing on what went wrong.

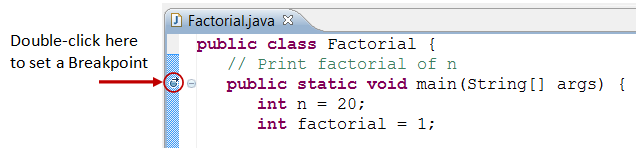
##### Step 0: Write a Java Program

The following program computes and prints the factorial of n (=1\*2\*3\*...\*n). The program, however, has a logical error and produce a wrong answer for n=20 ("The Factorial of 20 is -2102132736" - a negative number?!).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\*\* Compute the Factorial of n, where n=20.  \* n! = 1\*2\*3\*...\*n  \*/  public class Factorial {  public static void main(String[] args) {  int n = 20; // To compute factorial of n  int factorial = 1; // Init the product to 1    int i = 1;  while (i <= n) {  factorial = factorial \* i;  i++;  }  System.out.println("The Factorial of " + n + " is " + factorial);  }  } |

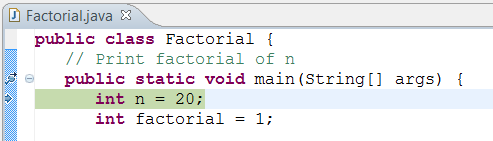
Let's use the graphic debugger to debug the program.

##### Step 1: Set an Initial Breakpoint



A breakpoint suspends program execution for you to examine the internal states (e.g., value of variables) of the program. Before starting the debugger, you need to set at least one breakpoint to suspend the execution inside the program. Set a breakpoint at main() method by double-clicking on the left-margin of the line containing main(). A blue circle appears in the left-margin indicating a breakpoint is set at that line.

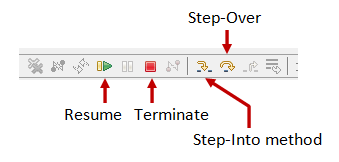
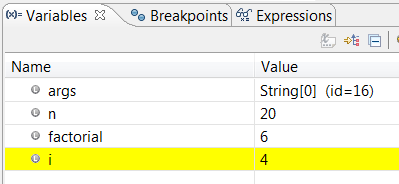
##### Step 2: Start Debugger



Right click anywhere on the source code (or from the "Run" menu) ⇒ "Debug As" ⇒ "Java Application" ⇒ choose "Yes" to switch into "Debug" perspective (A perspective is a particular arrangement of panels to suits a certain development task such as editing or debugging). The program begins execution but suspends its operation at the breakpoint, i.e., the main() method.

As illustrated in the following diagram, the highlighted line (also pointed to by a blue arrow) indicates the statement to be executed in the next step.

##### Step 3: Step-Over and Watch the Variables and Outputs



Click the "Step Over" button (or select "Step Over" from "Run" menu) to single-step thru your program. At each of the step, examine the value of the variables (in the "Variable" panel) and the outputs produced by your program (in the "Console" Panel), if any. You can also place your cursor at any variable to inspect the content of the variable.

Single-stepping thru the program and watching the values of internal variables and the outputs produced is the ultimate mean in debugging programs - because it is exactly how the computer runs your program!

##### Step 4: Breakpoint, Run-To-Line, Resume and Terminate

As mentioned, a breakpoint suspends program execution and let you examine the internal states of the program. To set a breakpoint on a particular statement, double-click the left-margin of that line (or select "Toggle Breakpoint" from "Run" menu).

"Resume" continues the program execution, up to the next breakpoint, or till the end of the program.

"Single-step" thru a loop with a large count is time-consuming. You could set a breakpoint at the statement immediately outside the loop (e.g., Line 11 of the above program), and issue "Resume" to complete the loop.

Alternatively, you can place the cursor on a particular statement, and issue "Run-To-Line" from the "Run" menu to continue execution up to the line.

"Terminate" ends the debugging session. Always terminate your current debugging session using "Terminate" or "Resume" till the end of the program.

##### Step 5: Switching Back to Java perspective

Click the "Java" perspective icon on the upper-right corner to switch back to the "Java" perspective for further programming (or "Window" menu ⇒ Open Perspective ⇒ Java).

Important: I can't stress more that mastering the use of debugger is crucial in programming. Explore the features provided by the debuggers.

##### Other Debugger's Features

Modify the Value of a Variable: You can modify the value of a variable by entering a new value in the "Variable" panel. This is handy for temporarily modifying the behavior of a program, without changing the source code.

Step-Into and Step-Return: To debug a method, you need to use "Step-Into" to step into the first statement of the method. ("Step-Over" runs the function in a single step without stepping through the statements within the function.) You could use "Step-Return" to return back to the caller, anywhere within the method. Alternatively, you could set a breakpoint inside a method.

### Read the Eclipse Documentation

At a minimum, you SHOULD browse through Eclipse's "Workbench User Guide" and "Java Development User Guide" - accessible via the Eclipse's "Welcome" page or "Help" menu. This will save you many agonizing hours trying to figure out how to do somethings later.

Source and a great reference : <http://www3.ntu.edu.sg/home/ehchua/programming/howto/eclipsejava_howto.html>