Eclipse for Java

How To Install Eclipse and Get Started with Java Programming (on Windows, macOS and Ubuntu)

TABLE OF CONTENTS (HIDE)

Eclipse (@ www.eclipse.org) is a *free* and *open-source* Java Integrated Development Environment (IDE), originated from IBM inspired by VisualAge (in 2001), and now maintained by Eclipse Foundation. Eclipse is a desktop app written mostly in Java. However, it uses its own Java graphics library known as SWT (Standard Widget Toolkit), instead of Java's Swing/AWT.

Eclipse is popular for Java application development (Java SE and Java EE) and Android apps. It also supports C/C++, PHP, Python, Perl, and other web project developments via extensible plug-ins. Eclipse is cross-platform and runs under Windows, Linux and macOS.

Eclipse Versions

The various versions are:

- Eclipse 1.0 (November 7, 2001): based on an earlier Java IDE called VisualAge from IBM.
- Eclipse 2.0 (June 28, 2002)
- Eclipse 2.1 (March 28, 2003)
- Eclipse 3.0 (June 25, 2004)
- Eclipse 3.1 (June 28, 2005)
- Eclipse 3.2 (June 30, 2006) (Callisto - named after one of the Jupiter's Galilean moons): started annual simultaneous release of all the related Eclipse projects.
- Eclipse 3.3 (June 25, 2007) (Europa - named after another Jupiter's Galilean moons)
- Eclipse 3.4 (June 19, 2008) (Ganymede - named after yet another Jupiter's Galilean moons)
- Eclipse 3.5 (June 12, 2009) (Galileo - named after the great 17th century scientist and astronomer Galileo Galilei)
- Eclipse 3.6 (June 23, 2010) (Helios - named after god of the sun in Greek Mythology)
- Eclipse 3.7 (June 23, 2011) (Indigo)
- Eclipse 4.2 (June 27, 2012) (Juno)
- Eclipse 4.3 (June 2013) (Kepler)
- Eclipse 4.4 (June 2014) (Luna)
- Eclipse 4.5 (June 2015) (Mars)
- Eclipse 4.6 (June 2016) (Neon)

- Eclipse 4.7 (June 2017) (Oxygen)
- Eclipse 4.8 (June 2018) (Photon)
- Eclipse 2018-09 (4.9) (starting quarterly release), Eclipse 2018-12 (4.10)
- Eclipse 2019-03 (4.11), Eclipse 2019-06 (4.12), Eclipse 2019-09 (4.13), Eclipse 2019-12 (4.14)
- Eclipse 2020-03 (4.15), Eclipse 2020-06 (4.16), Eclipse 2020-09 (4.17), Eclipse 2020-12 (4.18)
- Eclipse 2021-03 (4.19), Eclipse 2021-06 (4.20), Eclipse 2021-09 (4.21), Eclipse 2021-12 (4.22)
- Eclipse 2022-03 (4.23), Eclipse 2022-06 (4.24), Eclipse 2022-09 (4.25), Eclipse 2022-12 R (4.26)

1. How to Install Eclipse IDE 202x-xx for Java Developers

1.1 How to Install Eclipse on Windows

Step 0: Install JDK

To use Eclipse for Java programming, you need to first install Java Development Kit (JDK). Read "How to Install JDK for Windows".

Step 1: Download

Download Eclipse from https://www.eclipse.org/downloads/packages/. Choose "**Eclipse IDE for Java Developers**" and "**Windows x86_64**" (e.g., "`eclipse-java-202x-xx-R-win32-x86_64.zip`" - about 313MB) ⇒ Download.

Step 2: Unzip

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

I prefer the zip version, because there is no need to run any installer. Moreover, you can simply delete the entire Eclipse directory when it is no longer needed (without running any un-installer). You are free to move or rename the directory. You can install (unzip) multiple copies of Eclipse in the same machine.

1.2 How to Install Eclipse on macOS

To use Eclipse for Java programming, you need to first install JDK. Read "How to install JDK for macOS".

To install Eclipse:

1. Goto http://www.eclipse.org/downloads/package/. Choose "**Eclipse IDE for Java Developers**" and "**macOS x86_64**" (for Intel processor) or "**macOS AArch64**" (for M1 Arm Processor). To find out which processor your mac has, google "How to tell what processor your mac has". You will receive a DMG file.
2. Double-click the downloaded Disk Image (DMG) file. Follow the screen instructions to install Eclipse. Eclipse will be installed under "`/Applications/eclipse`". (To confirm!)

1.3 How to Install Eclipse on Ubuntu Linux

Eclipse comes with many flavors (See "Eclipse Packages" @ https://www.eclipse.org/downloads/compare.php):

- To use Eclipse for Java programming, choose "Eclipse IDE for Java Developers" (JavaSE) or "Eclipse IDE for Java EE Developers" (JavaEE). You need to first install JDK. Read "How to install JDK on Ubuntu".
- To use Eclipse for PHP programming, choose "Eclipse IDE for PHP Developers".
- To use Eclipse for C/C++ programming, choose "Eclipse IDE for C/C++ Developers".

Nonetheless, you can install any package, and then add more features when needed.

To install Eclipse (e.g, for Java Programming):

1. Download Eclipse from http://www.eclipse.org/downloads/. Under "Get Eclipse IDE 202x-xx" ⇒ Click the link "Download Packages" (instead of pushing the button "Download x86_64"). Choose "Eclipse IDE for Java Developers" for Java SE program development; or "Eclipse IDE for Java EE Developers" for developing webapps ⇒ Linux x86_64. You will receive a tarball (e.g., "`eclipse-java-202x-xx-R-linux-gtk-x86_64.tar.gz`") in the "`~/Downloads`" folder.
2. We shall install Eclipse under `/usr/local`.

```
// Unzip the tarball into /usr/local
$ cd /usr/local
$ sudo tar xzvf ~/Downloads/eclipse-java-202x-xx-R-linux-gtk-x86_64.tar.gz
      // Extract the downloaded package
      // x: extract, z: for unzipping gz, v: verbose, f: filename
      // Extract into /usr/local/eclipse
      // You can also unzip in "File Explorer" by double-clicking the tarball.

// Set up a symlink in /usr/bin (which is in the PATH)
$ cd /usr/bin
$ sudo ln -s /usr/local/eclipse/eclipse
      // Make a symlink in /usr/bin for the eclipse executable
$ ls -ld /usr/bin/eclipse
lrwxrwxrwx 1 root root 26 Aug 30 11:53 /usr/bin/eclipse -> /usr/local/eclipse/eclipse
$ which eclipse
/usr/bin/eclipse
```

To run Eclipse, open the "`/usr/local/eclipse`" folder and click on the "Eclipse" icon; or start a "Terminal", enter "eclipse".

Lock Eclipse on Launcher

Simply start Eclipse. Right-click the Eclipse icon ⇒ "Lock to Launcher" or "Add to Favourite".

(For older version - If the above don't work) Create a `/usr/share/applications/eclipse.desktop` file with the following contents:

```
[Desktop Entry]
Name=Eclipse
Type=Application
Exec=eclipse
Terminal=false
Icon=/usr/local/eclipse/icon.xpm
Comment=Integrated Development Environment
NoDisplay=false
Categories=Development;IDE;
Name[en]=Eclipse
```

Start Eclipse, right-click on the Eclipse icon on launcher ⇒ "Lock to launcher".

2. 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., the directory (or folder) that you would like to save your files (e.g., `c:\myProject\eclipse_workspace` for Windows) ⇒ Launch.
3. If the "Welcome" screen shows up, close it by clicking the "close" 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.
    a. In "Project name", enter "`FirstProject`".
    b. Check "Use default location".
    c. In "JRE", select "Use default JRE 'jre' and workspace compiler preferences".
    d. In "Project Layout", check "Use project folder as root for sources and class files".
    e. In "Module", **UNCHECK** "Create module-info.java" file.

    Push "Finish" button.
3. IF "Create module-info.java" dialog appears, Click "Don't Create" (This will not appear if you do step 2(e)).

Step 2: Write a Hello-world Java Program (or "Java Class")

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.
   a. In "Source folder", keep the "FirstProject".
   b. In "Package", leave it **EMPTY**. Delete the content if it is not empty.
   c. In "Name", enter "`Hello`".
   d. Check "`public static void main(String[] args)`".
   e. Don't change the rest.

   Push "Finish" button.
3. 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 pane (the bottom pane).

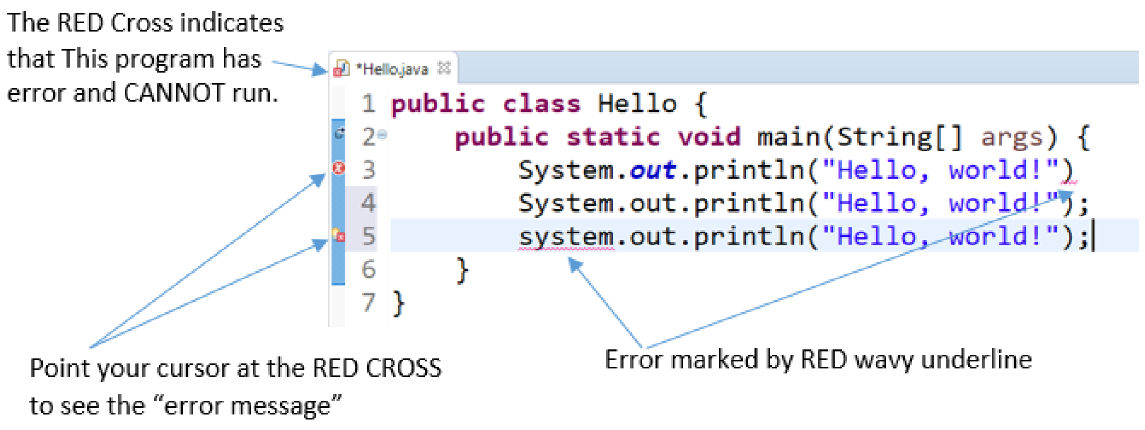
NOTES:

- You should create a NEW Java "project" for EACH of your Java application.
- Nonetheless, Eclipse allows you to keep more than one programs (classes) in a project, which is handy for writing toy programs (such as your tutorial exercises - you can keep many exercises in one project). 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.

2.1 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.


The RED Cross indicates that This program has error and CANNOT run.
*Hello.java
Point your cursor at the RED CROSS to see the “error message”
Error marked by RED wavy underline

```
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello, world!")
4         System.out.println("Hello, world!");
5         system.out.println("Hello, world!");
6     }
7 }
```

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.

2.2 JDK's Javadoc

You can read the Javadoc of a method, by placing the mouse cursor over the method.

3. 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.

4. 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

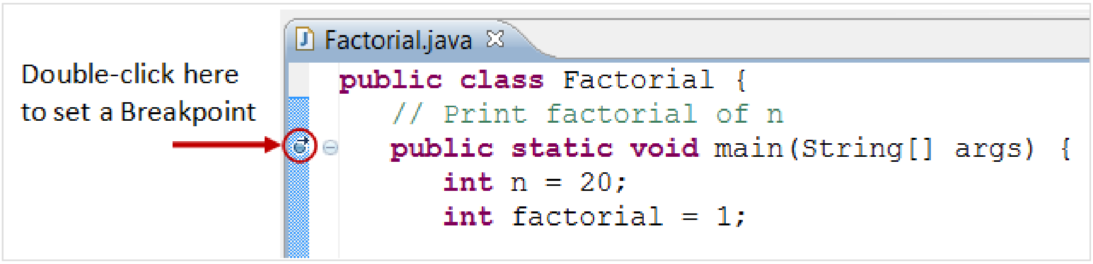
The following program computes and prints the factorial of *n* (=`1*2*3*...*`*n*). The program, however, has a logical error and produce a wrong answer for *n*=`20` ("`The Factorial of 20 is -2102132736`" - a negative number?!).

```
/** 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



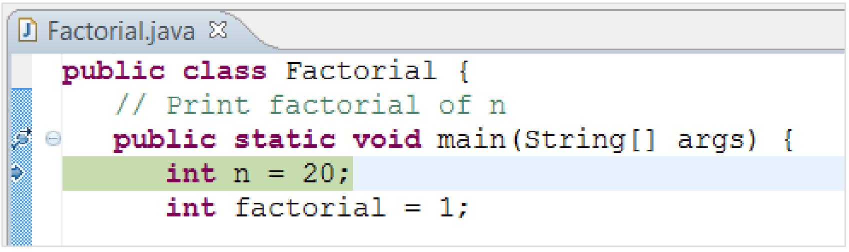
```
public class Factorial {
   // Print factorial of n
   public static void main(String[] args) {
      int n = 20;
      int factorial = 1;
```

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.



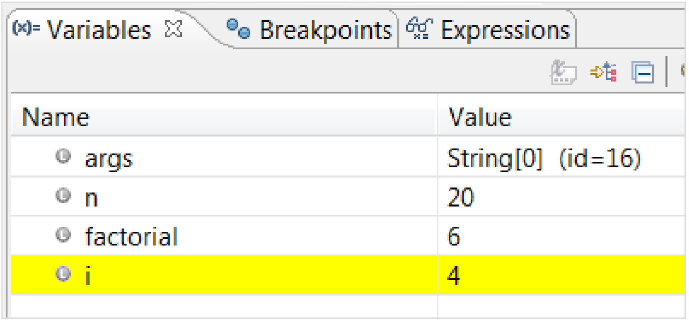
Factorial.java

```
public class Factorial {
   // Print factorial of n
   public static void main(String[] args) {
      int n = 20;
      int factorial = 1;
```

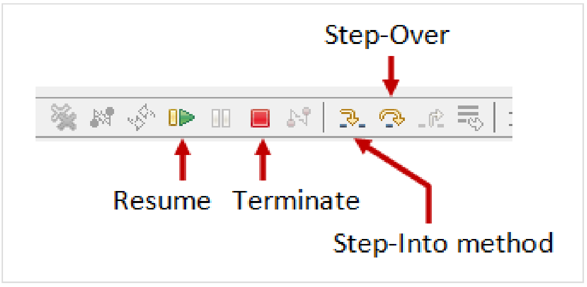
Step 3: Step-Over and Watch the Variables and Outputs

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!



Name	Value
args	String[0] (id=16)
n	20
factorial	6
i	4



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's stress more that mastering the use of debugger is crucial in programming. Explore the features provided by the debuggers.

Other Debugger's Features

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.

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.

5. Tips & Tricks

5.1 General Usages (for all Programming Tasks)

These are the features that I find to be most useful in Eclipse:

1. **Maximizing Window (Double-Clicking):** You can double-click on the "header" of any panel to *maximize* that particular panel, and double-click again to *restore* it back. This feature is particularly useful for writing source code in full panel.
2. **Shorthand Templates (sysout, for,...):** You can type "`sysout`" followed by a ctrl+space (or alt-/) as a shorthand for typing "`System.out.println()`".
   The default shortcut key (ctrl-space or alt-/) depends on the system. Check your system's shortcut key setting in "Edit" ⇒ "Content Assist" ⇒ "Default". Take note that many of you use ctrl+space to switch between input languages. You need to reconfigure either your language switching hot-key or Eclipse.
   Similarly, you can type "for" followed by ctrl-space (or alt-/) to get a for-loop.
   You can create your own shorthand in "Window" menu ⇒ "Preferences" ⇒ "Java" ⇒ "Editor" ⇒ "Templates". (Alternatively, in "Window" ⇒ "Preferences" ⇒ type "template" as filter text and choose "Java" ⇒ "Editor" ⇒ "Templates".)
   You can change your key settings in "Window" menu ⇒ "Preferences" ⇒ "General" ⇒ "Key" ⇒ choose "Command", "Content Assist". (Alternatively, in "Window" ⇒ "Preferences" ⇒ type "key" as filter text and choose "General" ⇒ "Key".)
3. **Intelli-Sense (ctrl-space):** You can use ctrl-space to activate the "intelli-sense" (or content assist). That is, Eclipse will offer you the choices, while you are typing.
4. **Source Formatting (ctrl-shift-f):** Right-click on the source. Choose "Source" ⇒ "Format" to let Eclipse to layout your source codes with the proper indentation.
5. **Source Toggle Comment (ctrl-/):** To comment/uncomment a block of codes, choose "Source" ⇒ "Toggle Comment".
6. **Hints for Correcting Syntax Error:** If there is a syntax error on a statement, a red mark will show up on the left-margin on that statement. You could click on the "light bulb" to display the error message, and also select from the available hints for correcting that syntax error.
7. **Refactor (or Rename) (alt-shift-r):** You can rename a variable, method, class, package or even the project easily in Eclipse. Select and right-click on the entity to be renamed ⇒ "Refactor" ⇒ "Rename". Eclipse can rename all the occurrences of the entity.
8. **Line Numbers:** To show the line numbers, choose "Window" menu ⇒ "Preferences" ⇒ "General" ⇒ "Editors" ⇒ "Text Editors" ⇒ Check the "Show Line Numbers" Box. You can also configure many editor options, such as the number of spaces for tab. Alternatively, you can right-click on the left-margin, and check "Show Line Numbers".
9. **Error Message Hyperlink:** Click on an error message will hyperlink to the corresponding source statement.
10. **Changing Font Type and Size:** From "Window" menu ⇒ "Preferences" ⇒ "General" ⇒ "Appearance" ⇒ "Colors and Fonts" ⇒ expand "Java" ⇒ "Java Editor Text Font" ⇒ "Edit". (Alternatively, in "Window" ⇒ "Preferences" ⇒ type "font" as filter text and choose the appropriate entry.)
11. **Unicode Support:** To enable Unicode support, select "Window" menu ⇒ Preferences ⇒ General ⇒ Workspace ⇒ Text file encoding ⇒ UTF-8. This sets the default character set used for file encoding, similar to VM's command-line option `-Dfile.encoding=UTF-8`. Commonly used charsets for Unicode are UTF-8, UTF-16 (with BOM), UTF-16BE, UTF-16LE. Other charsets are US-ASCII, ISO-8859-1.
12. **Mouse Hover-over:** In debug mode, you could configure to show the variable's value when the mouse hovers over the variable. Select "Window" menu ⇒ "Preferences" ⇒ "Java" ⇒ "Editor" ⇒ "Hover".
13. **Comparing Two Files:** In "Package Explorer", select two files (hold the control key) ⇒ Right-click ⇒ Compare with ⇒ Each Other.
14. **Setting Keyboard Shortcut Keys:** You can set/change the keyboard shortcut keys at "Window" ⇒ "Preferences" ⇒ "General" ⇒ "Key".
    I like to set the frequently-used commands to Ctrl-1 to Ctrl-10, for examples, "Run Java Application" to "Ctrl-1", etc.
15. **Useful Eclipse Shortcut Keys:**
    - F3: Goto the declaration of the highlighted variable/method.
    - Ctrl-Shift-G: Search for ALL references of the highlighted variable/method in workspace.

- Ctrl-G: Search for the Declaration of a variable/method in workspace.
  Don't use Find (Ctrl-F), but use the above context-sensitive search.
- Ctrl-Shift-F: Format the source code.
- Ctrl-Shift-O: Organize imports.
- Alt-Shift-R: Rename. (Don't use Find/Replace.)
- Ctrl-Space: auto-complete.

16. **Package Explorer vs. Navigator:** We usually use "Package Explorer" in programming, but it will not show you all the folders and files under the project. On the other hand, "Navigator" is a file manager that shows the exact file structure of the project (similar to Windows Explorer). You can enable the Navigator by "Window" ⇒ Show view ⇒ Navigator.
17. **Spell Check:** To enable spell check, select Window ⇒ Preferences ⇒ type "spell" in the filter ⇒ General ⇒ Editors ⇒ Text Editors ⇒ Spelling ⇒ Check "Enable spell checking". Also provide a "User defined dictionary" (with an initially empty text file).
    To correct mis-spell words, right-click and press ctrl-1 (or Edit menu ⇒ Quick Fix).
18. **Eclipse's Log File:** Goto Help ⇒ about Eclipse ⇒ Installation details ⇒ Configuration ⇒ View Error Log.
19. **Viewing two files in split screen:** Simply click and hold on the title of one file and drag it to the editor screen. You can split horizontally or vertically by varying the drag target.
    To view the SAME file on split screen, create a new editor window by selecting Window ⇒ New Editor; and drag one window to the lower side of the screen. Alternatively, select Window ⇒ Editor ⇒ Toggle Split Editor (Horizontal) or Toggle Split Editor (Vertical).
20. **Block Select (Column Select):** Push Alt-Shift-A to toggle between block-select mode and normal mode.
21. **Snippets:**
    - To view the snippet window: choose "Window" ⇒ Show View ⇒ Snippets.
    - To create a new snippet category: Right-click ⇒ Customize ⇒ New.
    - To create a new snippet item: Copy the desired text ⇒ Select the snippet category ⇒ paste as snippet.
    - To insert a snippet: place the cursor on the desired location at the editor panel ⇒ click the snippet item.
22. **Word Wrap (Line Wrap):** Word-wrap (or line-wrap) is essential for editing long HTML documents without the horizontal scroll bar. However, the Eclipse's HTML Editor and Text Editor do not support word-wrap.
    You could install a plug-in called "Word Wrap" from http://ahtik.com/eclipse-update/.
    Choose "Help" ⇒ Install New Software ⇒ in "Work with" Enter "http://ahtik.com/eclipse-update/".
    To activate word wrap, right-click on the editor panel ⇒ select "Word Wrap".
23. **Creating "link folder" in project**: You do not have to place all the folders under the project base directory, instead, you can use so-called "link folders" to link to folder outside the project base directory.
    To create a link folder in a project, right-click on the project ⇒ File ⇒ New ⇒ Folder ⇒ Advanced ⇒ Check Link to alternate Location (Linked Folder).
24. **Running Eclipse in "clean" mode:** You can run eclipse in so-called "`clean`" mode, which wipes all the cached data and re-initialize the cache, by running eclipse from command-line with "-clean" argument (i.e., "`eclipse -clean`"). It is useful if something is not working proper, especially if you install a new copy of Eclipse.
25. **Show the Right Margin:** Window ⇒ Preferences ⇒ General ⇒ Editors ⇒ Text Editors ⇒ Show Print Margin and set the column number.
26. **Zoom in/out (ctrl++ or ctrl+-)**
27. Let me know if you have more tips to be included here.

5.2 Update Eclipse and Install new Software

1. **Install New Software:** Select "Help" menu ⇒ Install New Software ⇒ In "Work With", pull down the select menu and choose a software site.
2. **Update:** Select "Help" menu ⇒ Check for Updates.

5.3 For Java Application Development Only

1. **Small Toy Java Programs:** You can keep many small programs (with `main()`) in one Java project instead of create a new project for each toy program. To run the desired program, right-click on the source file ⇒ "Run as" ⇒ "Java Application".
2. **`Scanner/printf()` and JDK 1.5:** If you encounter syntax error in using `printf()` or `Scanner` (which are available from JDK 1.5), you need to check your compiler settings. Select "Window" menu ⇒ Preferences ⇒ open the "Java" node ⇒ select "Compiler" ⇒ in "Compiler compliance level" ⇒ select the latest release, which should be "1.5" or above.
3. **Command-Line Arguments:** To provide command-line arguments to your Java program in Eclipse, right-click on the source file ⇒ "Run Configurations" ⇒ Under the "Main" panel, check that "Project" name and "Main Class" are appropriate ⇒ Select the "Argument" tab ⇒ type your command-line arguments inside the "Program Arguments" box ⇒ "Run".
4. **Resolving Import (Ctrl-Shift-o)**: To ask Eclipse to insert the `import` statements for classes. Useful when you copy a large chunk of codes without the corresponding import statements.
5. **Including Another Project:** To include another project in the same work space, right-click on the project ⇒ Build Path ⇒ Configure Build Path... ⇒ Select "Projects" tab ⇒ "Add..." to select project in the existing work space ⇒ OK.
6. **Exporting a Project to a JAR file:** Right-click on the project ⇒ Export... ⇒ Java, JAR File ⇒ Next ⇒ Select the files to be exported ⇒ Next ⇒ Next ⇒ In "JAR Manifest Specification" dialog, enter the main class (if you wish to run the JAR file directly) ⇒ Finish.
7. **Unit Testing:** If you keep your test in another project, you need to include the project under test in your Build Path (see above).
   To create a test case: Right-click on the project ⇒ New ⇒ JUnit Test Case ⇒ the "New JUnit Test Case" dialog appears. Select "New JUnit 4 Test". In "Name", enter your class name. In "Class under test", browse and select the class to be tested.
   To run the test: Right-click ⇒ "Run As" ⇒ "JUnit Test". The results are displayed in a special "JUnit console".
8. **Adding External JAR files & Native Libraries ("`.dll`", "`.lib`", "`.a`", "`.so`"):** Many external Java packages (such as JOGL, Java3D, JAMA, etc) are available to extend the functions of JDK. These packages typically provide a "`lib`" directory containing JAR files ("`.jar`") (Java Archive - a single-file package of Java classes) and native libraries ("`.dll`", "`.lib`" for windows, "`.a`", "`.so`" for Linux and macOS).
   To include these external packages into an Eclipse's project, right-click on the project ⇒ Build Path ⇒ Add External Archives ⇒ Navigate to select the JAR files ("`.jar`") to be included.
   In "Package Explorer", right-click on the JAR file added ⇒ Properties:
   - To include native libraries ("`.dll`", "`.lib`", "`.a`", "`.so`"), select "Native Library" ⇒ "Location Path" ⇒ "External Folder".
   - To include the javadoc, select "JavaDoc Location" ⇒ "JavaDoc URL" ⇒ You can specify a *local* file or a remote link.
   - To include source file (for debugging), select "Java Source Attachment".

   All the above options are also accessible via project's property ⇒ "Build Path".
   Notes: The JAR files must be included in the `CLASSPATH`. The native library directories must be included in JRE's property "`java.library.path`", which normally but not necessarily includes all the paths from the `PATH` environment variable. Read "External JAR files and Native Libraries".
9. **Creating a User Library:** You can also create a Eclipse's *user library* to include a set of JAR files and native libraries, that can then be added into subsequent Eclipse projects.
   For example, I created a user library for "JOGL" as follows:
   a. From "Window" menu ⇒ Preferences ⇒ Java ⇒ Build Path ⇒ User Libraries ⇒ New ⇒ In "User library name", enter "`jogl`". The "User Library" dialog appears.
   b. In "User Library" dialog ⇒ Select "`jogl`" ⇒ Add JAR... ⇒ Navigate to `<JOGL_HOME>/lib`, and select "`gluegen-rt.jar`" and "`jogl.jar`".
   c. Expand the "`jogl.jar`" node ⇒ Select "Native library location: (none)" ⇒ Edit... ⇒ External Folder... ⇒ select `<JOGL_HOME>/lib`.
   d. Expand the "`jogl.jar`" node ⇒ Select "Javadoc location: (none)" ⇒ Edit... ⇒ Javadoc in archive ⇒ In "Archive Path", "Browse" and select the downloaded JOGL API documentation zip-file ⇒ In "Path within archive", "Browse" and expand the zip-file to select the top-level path (if any) ⇒ Validate. Alternatively, you can provide the path to

the un-zipped javadocs. This is needed for Eclipse to display javadoc information about classes, fields, and methods.

e. You may provide the source files by editing "Source attachment: (none)". Source is needed only if you are interested to debug into the JOGL source codes.

For EACH subsequent Java project created that uses JOGL, right-click on the project ⇒ Build Path ⇒ Add Libraries ⇒ Select "User Library" ⇒ Check "`jogl`".

10. **Running an External Program:** Suppose that you want to run a Perl script on the selected file, you can configure an external tool as follows:
    a. From "Run" menu ⇒ External Tools ⇒ External Tools Configuration... ⇒ The "External Tools Configuration" dialog appears.
    b. In "Name", enter your tool name.
    c. Choose the "Main" tab ⇒ In "Location", "Browse File System..." to choose the perl interpreter "perl" ⇒ In "Arguments", enter "`path/scriptname.pl ${resource_loc}`", where `${resource_loc}` is an Eclipse variable that denotes the currently selected resource with absolute path.
    d. Choose the "Common" tab ⇒ In "Standard Input and Output", uncheck "Allocate Console", check "File" and provide an output file (e.g., `d:\temp\${resource_name}.txt`).
    e. (If you use the CYGWIN perl interpreter, need to set environment variable CYGWIN=nodosfilewarning to disable warning message.)

    To run the configured external tool, select a file ⇒ run ⇒ external tool ⇒ tool name.
11. **Viewing Hex Code of Primitive Variables in Debug mode:** In debug perspective, "Variable" panel ⇒ Select the "menu" (inverted triangle) ⇒ Java ⇒ Java Preferences... ⇒ Primitive Display Options ⇒ Check "Display hexadecimal values (byte, short, char, int, long)".
12. **Adding a New Version of JDK/JRE:** First, you can check the installed JDK/JRE via "Window" menu ⇒ "Preferences" ⇒ Expand "Java" node ⇒ "Installed JREs". Check the "Location" current JRE installed to make sure that it is the intended one. You can use the "Add" button to add a new version of JRE. For program development, I recommend that you add the JDK (instead of JRE). [The "Location" decides the extension directory used for including additional JAR files, e.g., `$JAVA_HOME\jre\lib\ext`.]
13. To highlight matching variables: Select Window ⇒ Preferences ⇒ Java ⇒ Editor ⇒ Mark Occurrences.

5.4 For Web Developers

1. **HTML Editor:** Use the "Web Page Editor" (available in Eclipse Java EE), which provides the design view (WYSISYG).
   To use the "Web Page Editor", right-click on the HTML file, open as "Web Page Editor".
   To make the "Web Page Editor" as default for HTML file, goto Window ⇒ Preferenes ⇒ General ⇒ Editor ⇒ File Associations ⇒ .htm and .html ⇒ Select "Web page editor" ⇒ default.

6. File I/O in Eclipse

The question always is: where to place the files or external resources?

The following program create and write to a text file "`out.txt`" (via `java.util.Formatter`), and read it back (via `java.util.Scanner`). I do the write first so that you can check the location of the exteranl files under eclipse, which is at the project base directory, at the same level as the "`src`" and "`bin`".

```java
import java.util.Scanner;
import java.util.Formatter;
import java.io.File;
import java.io.FileNotFoundException;

public class TestFileIO {
   public static void main (String [] args) {
      // Create and write text file - You can check the location of the file created
      try {
         Formatter out = new Formatter(new File("out.txt")); // filename only, no path
         out.format("%d %f %s%n", 1234, 55.66, "hello");
         out.close();    // flush the output and close the output file
```