Eclipse for Java

How To Install Eclipse and Get Started with Java Programming (on Windows, Mac and Linux)

TABLE OF CONTENTS (HIDE)

Eclipse (@ www.eclipse.org) is an *open-source* Integrated Development Environment (IDE) supported by IBM. 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 Mac OS.

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)

1. How to Install Eclipse 4.5 (Mars) for Java

1.1 For 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 (on Windows)".

Step 1: Download

Download Eclipse from http://www.eclipse.org/downloads. For beginners, choose "**Eclipse IDE for Java Developers**" (32-bit or 64-bit) (e.g., "`eclipse-java-luna-SR1a-win32-x86_64.zip`").

Step 2: Unzip

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

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 For Macs

Read "How to Install Eclipse on Mac".

1.3 For Ubuntu Linux

Read "How to Install Eclipse on Ubuntu".

2. Writing your First Java Program in Eclipse

Step 0: Launch Eclipse

1. Start Eclipse by running "`eclipse.exe`" in the Eclipse installed directory.
2. Choose an appropriate directory for your *workspace* (i.e., where you would like to save your files).
3. If the "Welcome" screen shows up, close it by clicking the "close" button.

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".
2. The "New Java Project" dialog pops up.
    a. In the "Project name" field, enter "`FirstProject`".
    b. Check "Use default location".
    c. In the "JRE" box, select "Use default JRE (currently 'JDK1.x')". But check the JDK version, you should be using JDK 1.5 and above.
    d. Click "Finish".

Step 2: Write a Hello-world Java Program

1. In the "Package Explorer" (left panel) ⇒ Right-click on "`FirstProject`" (or use the "File" menu) ⇒ New ⇒ Class.
2. The "New Java Class" dialog pops up.
    a. In "Name" field, enter "`Hello`".
    b. In "package" field, delete the content if it is not empty.

c. Check "`public static void main(String[] args)`" box.

d. Click "Finish".

3. The source file "`Hello.java`" opens on the editor panel. Enter the following codes:

```
public class Hello {   // "Hello.java"
   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 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 from the "Run" menu) ⇒ Choose "Run As" ⇒ "Java Application".
3. The output "Hello, world!" appears on the "Console" panel.

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). If you have more than one files with `main()` method in one project, you need to right-click the source and choose "Run As" ⇒ "Java Application" to run that particular source. Clicking the "Run" button runs the recently-run program (based on the previous configuration). Try clicking on the "down-arrow" besides the "Run" button.

3. Read the 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.

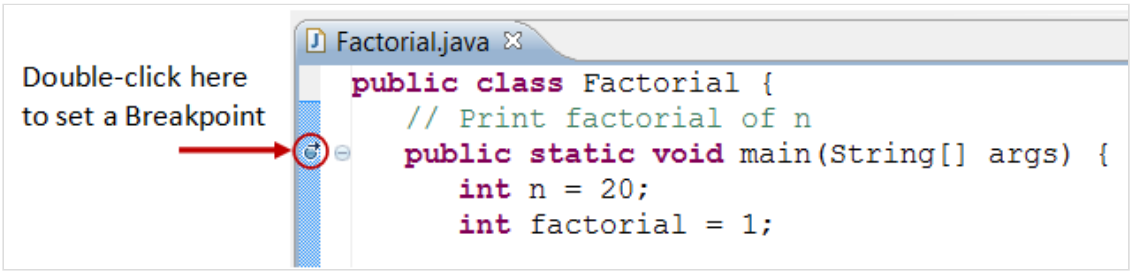
Step 0: Write a Java Program

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?!).

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

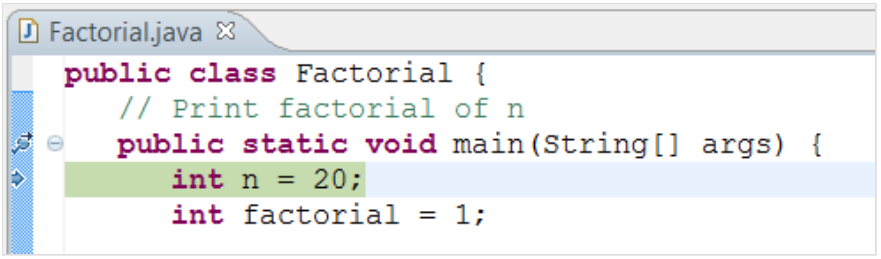
Let's use the graphic debugger to debug the program.

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.

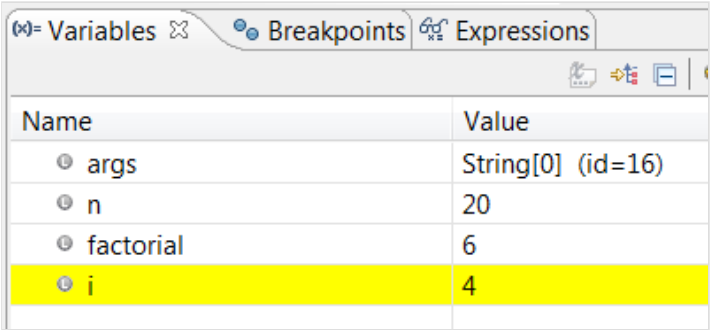
Step 2: Start Debugger



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.

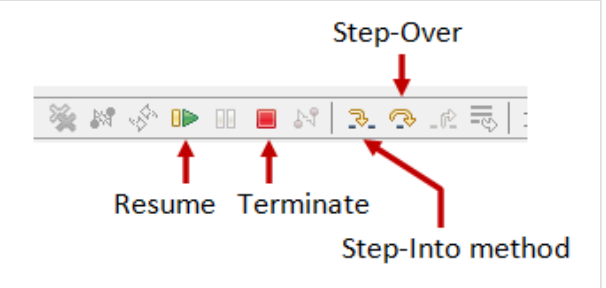
As illustrated in the following diagram, the highlighted line (also pointed to by a blue arrow) indicates the statement to be executed in the *next* step.

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's 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.

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.
14. **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.
15. **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.
16. **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).
17. **Eclipse's Log File:** Goto Help ⇒ about Eclipse ⇒ Installation details ⇒ Configuration ⇒ View Error Log.
18. **Viewing two files in split screen:** Simply click and hold on the title of one file and drag it to the lower side of the screen. [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.]
19. **Block Select:** Push Alt-Shift-A to toggle between block-select mode and normal mode.
20. **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.
21. **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".
22. **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).

23. **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.
24. Show the Right Margin: Window ⇒ Preferences ⇒ General ⇒ Editors ⇒ Text Editors ⇒ Show Print Margin and set the column number.
25. 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. Eclipse 3.7 does not support JDK 7. Eclipse 4.2 does.
2. **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".
3. **`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.
4. **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".
5. **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.
6. **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.
7. **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.
8. **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".
9. **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 Mac).
   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".

10. **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`".
11. **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.
12. **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)".
13. **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`.]

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

Suppose that your want to write a Java program, which inputs from a text file called "`xxxx.in`" and outputs to a text file called "`xxxx.out`". This is a little tricky under Eclipse due to:

1. When you create a text file in Windows' Notepad and saved it as "`xxxx.in`", Notepad will append the "`.txt`" to your file and it becomes "`xxxx.in.txt`". Worse still, the Windows' Explorer, by default, will not show the "`.txt`" extension. (The first thing I always do to an alien computer is to change this setting. From "Tools" menu ⇒ Folder Options... ⇒ View ⇒ Uncheck "Hide extensions for known file types".) You need to put a pair of double quotes around `xxxx.in` to override the default "`.txt`" extension. This is one good reason not to use Notepad for programming at all. You should use Eclipse to create the text file instead.
2. Which directory to keep the input file "`xxxx.in`" in Eclipse?
   - If you did not separate the sources and class files into two separate directories, then the answer is straight forward, because there is only one directory to place your input file.
   - If you choose to keep your sources and class files in two separate directories, eclipse will create two sub-directories "src" and "bin" under the *base* directory. BUT you need to put your input file "`xxxx.in`" in the *base* directory of your project, instead of the "src" or "bin"..

For writing simple programs:

- Put the sources, class files, and the input/output files in the same directory. (When you create a new project, select "Use project folder as root for sources and class files" in "Project Layout".) (But put your sources and class files in separate directories for big project.)
- You can create you input file from eclipse directly via "File" menu ⇒ "New" ⇒ "File".
- Remember to add a newline to the end of your input file.
- You may need to right-click the project and select "Refresh" to see the output file "`xxxx.out`" created in the package explorer.
- To open the "`xxxx.in`" and "`xxxx.out`": right-click ⇒ Open With ⇒ Text Editor.

This is a sample JDK 1.5 program for file input/output:

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

public class FileIOTest {  // saved as "FileIOTest.java"
   public static void main (String [] args) throws IOException {
      Scanner in = new Scanner(new File("FileIOTest.in"));        // file input
      Formatter out = new Formatter(new File("FileIOTest.out"));  // file output

      int a = in.nextInt();
      int b = in.nextInt();
      out.format("%d\n",a+b);  // format() has the same syntax as printf()

      out.close();    // flush the output and close the output file
   }
}
```

Create the input text file called "`FileIOTest.in`" with the following contents and terminated with a newline:

```
55 66
```

7. Writing Swing Applications using Eclipse GUI Builder

Eclipse provides a visual GUI builder called "WindowBuilder" (@ https://www.eclipse.org/windowbuilder), which supports AWT/Swing, SWT (Eclipse's Standard Widget Toolkit - an alternative to JDK's AWT/Swing), XWT, GWT, eRCT.

Step 1: Create a New "Java Application" Project

1. Choose "File" menu ⇒ "New" ⇒ "Java project".
2. The "New Java Project" dialog pops up.
    a. In the "Project name" field, enter "`FirstSwingProject`".
    b. Check "Use default location".
    c. In the "JRE" box, select "Use default JRE (currently 'JDK1.x')". But check the JDK version, you should be using JDK 1.5 and above.
    d. Click "Finish".

Step 2: Create a Swing JFrame Subclass

1. Choose "File" menu ⇒ "New" ⇒ "Other..." ⇒ "WindowBuilder" ⇒ "Swing Designer" ⇒ "JFrame" ⇒ "Next". [If you cannot find "WindowBuilder" option, you need to install the "WindowBuilder" plug-in for Eclipse. Check https://www.eclipse.org/windowbuilder.]
2. In the "Create JFrame" dialog ⇒ Enter "SwingMain" in the "Name" field ⇒ "Finish".
3. Select the "Design" pane.
4. In "Layouts", select "FlowLayout" and click on the "design form".
5. From "Components", select "`JLabel`" and click on the design form. Change the label text to "Counter: ". Select a "JTextField" and place it on the design form. Change the text to "0". Select a "JButton" and place it on the design form. Change the text label to "Count".
6. To attach a event-handler to the button, double-click the `JButton` to switch into the "Source" pane, with the event-handler skeleton created. Complete the `actionPerformed()` as follows:

    ```
    public void actionPerformed(ActionEvent e) {
       count++;
       textField.setText(count + "");
    }
    ```

    Add an instance variable called `count` as follow:

    ```
    public class SwingMain extends JFrame {
       private int count = 0;
       ......
    ```

7. You can now ready run the program. Right-click on the project ⇒ Run As ⇒ Java Application.

Eclipse Generated Codes

Study the codes generated by Eclipse GUI Builder, as follows, which is just a typical Swing application.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.border.EmptyBorder;
5
6  public class SwingMain extends JFrame {  // A JFrame application
7
8     // Define private variables of all the GUI components
9     private JPanel contentPane;
10    private JTextField textField;
11    private int count = 0;
12
13    /**
14     * Launch the application.
15     */
16    public static void main(String[] args) {
17       EventQueue.invokeLater(new Runnable() {  // same as SwingUtilities.invokeLater()
18          @Override
19          public void run() {
```

```
            try {
               SwingMain frame = new SwingMain();
               frame.setVisible(true);
            } catch (Exception e) {
               e.printStackTrace();
            }
         }
      });
   }

   /**
    * Create the frame.
    */
   public SwingMain() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setBounds(100, 100, 450, 300);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      setContentPane(contentPane);
      contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

      JLabel lblNewLabel = new JLabel("Counter: ");
      contentPane.add(lblNewLabel);

      textField = new JTextField();
      textField.setText("0");
      contentPane.add(textField);
      textField.setColumns(10);

      JButton btnCount = new JButton("Count");
      btnCount.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            count++;
            textField.setText(count + "");
         }
      });
      contentPane.add(btnCount);
   }
}
```

8. Eclipse for C/C++ Programming

Here.

9. Eclipse PDT (PHP Development Tool)

Here.

10. Eclipse and Database Development (MySQL)

Reference: "Data Tools Platform User Documentation" @ Eclipse Welcome page.

You need to install Eclipse for Java EE, MySQL and MySQL Connector/J Driver. Read "How to install and get started with MySQL".

To use Eclipse for MySQL development:

1. Switch to "Database Development" perspective: From "Window" menu ⇒ Open Perspective ⇒ Other ⇒ Database Development.
2. Connect to MySQL database server: Start your MySQL database server. Right-click "Database Connection" ⇒ New.

a. In "Connection Profile", choose "MySQL" ⇒ Next.

b. In "Driver", choose "New Driver Definition" ⇒ Choose the MySQL Connector/J Driver ⇒ In JAR List, select the MySQL Connector/J Driver JAR file (that you have installed). In "Properties", "General", specify URL (take note of the database name), User name and password ⇒ Test Connection ⇒ Finish.

c. In "Datasource Explorer", you can "connect" and "disconnect" the connection.

3. To create a new SQL script, choose File ⇒ New ⇒ SQL File ⇒ You may use an existing project or create a new project (General - Project or Web - Dynamic Web Project) ⇒ Enter filename, and set the connection profile name ⇒ Finish. Enter a SQL statement (e.g., SELECT * FROM tablename) ⇒ Right-click on the text ⇒ "Execute Current Text" or "Execute All".
4. To use an existing SQL file, drop the file into a project and open the SQL file. In Connection profile, set the type and connection name. Right-click on a statement ⇒ "Execute ...".
5. To CURD (create-update-read-delete) tables, in "Datasource Explorer" (of the "Database Development" perspective), expand "database" to view the tables. Right-right on the table ⇒ Datat ⇒ Edit.

11. Developing and Deploying Web Applications in Eclipse for Java EE

11.1 Setting Up Eclipse for Web Development

1. Install "Eclipse for Java EE (Enterprise Edition)".
2. Install Tomcat (or Glassfish) server.
3. Configuring Web Server: Launch Eclipse ⇒ Window ⇒ Preferences ⇒ Expand the "Server" node ⇒ "Runtime Environments" ⇒ "Add..." ⇒ Expand "Apache" and select "Apache Tomcat v7.0" ⇒ Enter the "Tomcat Installation Directory" ⇒ "Finish".

11.2 Writing a Hello-world JSP Page

1. Create a new Web Application: File ⇒ New ⇒ Dynamic Web Project (under "Web" category) ⇒ In "Project Name", enter "`HelloJSP`" ⇒ Finish.
2. Create a new JSP File: Right-click on the project "`HelloJSP`" ⇒ New ⇒ JSP File ⇒ The parent folder shall be "HelloJSP/WebContent" ⇒ In "File Name", enter "`Hello`" ⇒ "Finsih".
3. Enter the following HTML/JSP codes within the `<body>...</body>` tags:

```
<h1>Hello World!</h1>
<%
out.println("Your IP address is " + request.getRemoteAddr() + "<br/>");
out.println("Your user agent is " + request.getHeader("user-agent") + "<br/>");
%>
```

4. To execute the JSP, right-click on "`Hello.jsp`" ⇒ Run As ⇒ Run on Server.

11.3 Writing a Hello-world Servlet

1. Create a new Web Application: File ⇒ New ⇒ Dynamic Web Project (under "Web" category) ⇒ In "Project Name", enter "`HelloServlet`" ⇒ "Finish".
2. Create a new Servlet: Right-click on the project "`HelloServlet`" ⇒ New ⇒ Servlet ⇒ In "Java Package", enter "`hello`"; in "Class Name", enter "`HelloServlet`" ⇒ Next ⇒ In "URL Mappings", select "`\HelloServlet`", "Edit" to "`\Hello`" ⇒ Next ⇒ In "Which method stubs would you like to create", check "Inherited abstract method" and "doGet" ⇒ Finish.
   In "HelloServlet.java", enter the following codes:

```
package hello;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import javax.servlet.annotation.WebServlet;

@WebServlet("/sayhello")  // URL for this Servlet (for Servlet 3.0 with Tomcat 7)
public class HelloServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      // Set the response message MIME type (in Content-Type response header)
      response.setContentType("text/html");
      // Get an output Writer to write the response message over the network
      PrintWriter out = response.getWriter();
      // Write the response message (in an HTML page) to display "Hello, world!"
      out.println("<!DOCTYPE html>");
      out.println("<html><head>");
      out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
      out.println("<title>Hello Servlet</title></head>");
      out.println("<body><h1>Hello, World!</h1>");
      out.println("<p>Your IP address is " + request.getRemoteAddr() + "</p>");
      out.println("<p>Your user agent is " + request.getHeader("user-agent") + "</p>");
      out.println("</body></html>");
   }
}
```

(For Servlet 2.4/2.5 with Tomcat 6) The annotation `@WebServlet` is new in Servlet 3.0 and is not supported in Servlet 2.4/2.5. Hence, you need to manually configure the URL for the servlet in the Web Application Deployment Descriptor "`web.xml`" under directory "`WEB-INF`", as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0" metadata-complete="true">

  <servlet>
    <servlet-name>HelloServletExample</servlet-name>
    <servlet-class>hello.HelloServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloServletExample</servlet-name>
    <url-pattern>/sayhello</url-pattern>
  </servlet-mapping>

</web-app>
```

3. To execute the Servlet, right-click on the "`HelloServlet`" project ⇒ "Run As" ⇒ "Run on Server" ⇒ Change the URL to "`http://localhost:8080/HelloServlet/sayhello`".

11.4 Exporting a Web Application as a WAR file

Right-click on the project to be exported ⇒ Export ⇒ WAR File ⇒ In "Destination", specify the destination directory and filename (the filename shall be the web application name) ⇒ Finish.

1. To deploy the war file in Tomcat, simply drop the war file into Tomcat's "`webapps`" folder. The war file will be automatically extracted and deployed. The web application name is the war-filename.
2. You could use WinZip (or WinRAR) to view the content of the war file, as war-file is in ZIP format.

11.5 Writing a Hello-world JSF Page

[TODO]

11.6 Debugging Web Applications

You can debug a webapp just like standalone application. For example, you can set breakpoints, single-step through the programs, etc.

REFERENCES & RESOURCES

1. Eclipse mother site @ www.eclipse.org.
2. Eclipse documentation and user guides, accessible via Eclipse's Help and menu.

Latest version tested: Eclipse 4.5 (Mars)
Last modified: January, 2016