Eclipse Tutorial

Thanks to Sandeep Pasuparthy

- In this tutorial we will be walking through a small demo application using the Eclipse Development Environment. Previous knowledge of any kind of tool is not necessary, but can be helpful when understanding why we are following certain steps or how we are making this application work.

- **www.eclipse.org**

What's Eclipse?

- It is a free software / **open source** platform-independent software tool for delivering what the project calls "rich-client applications"
- It is an **Integrated Development Environment** (IDE), it manages the whole development process of Java applications, by providing many features for programming (editor, debugger, etc.)
- It supports other languages by means of plug-ins (C/C++)
- Multi-platform (Linux, Windows, Mac OS)

What's Eclipse?

- Eclipse is also a community of users, constantly extending the covered application areas.

- Eclipse was originally developed by IBM as the successor of its VisualAge family of tools.

- Eclipse is now managed by the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors.

Getting Eclipse

- In the Lab: already installed.

- On your laptop
  - Download the latest version at:
    - https://eclipse.org/downloads/ (Eclipse IDE for Java Developers)
    - Decompress the downloaded package and click eclipse.exe (under Windows) or run eclipse (under Linux)
 - Installation steps at http://wiki.eclipse.org/Eclipse/Installation

Essentials of Eclipse

Before going into Eclipse, some of the basic stuff you need to know are:

- In Eclipse you need to start with **creating a project**.
- Then choosing a particular project, we **create** our **java file** in it. No need to worry, it is simple and you will learn it by the end of this presentation.
- In Eclipse you need not remember the exact command syntax, it helps you writing the commands.
- Just by saving a file, Eclipse compiles the program by default. It makes work easy for programmers.
- Just go step by step.

Let's start with basic stuff

Step1: Open Eclipse from start on your system, choose your workspace: workspace is the directory where the projects will be stored.
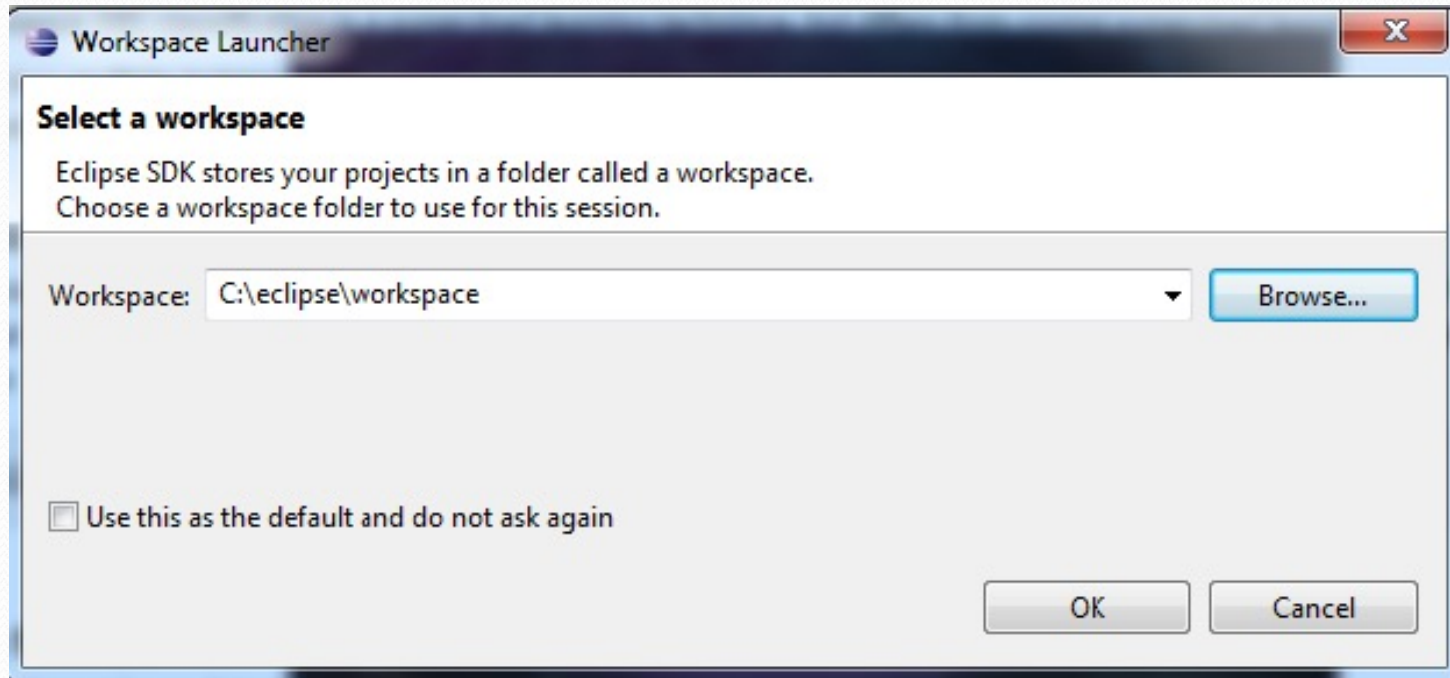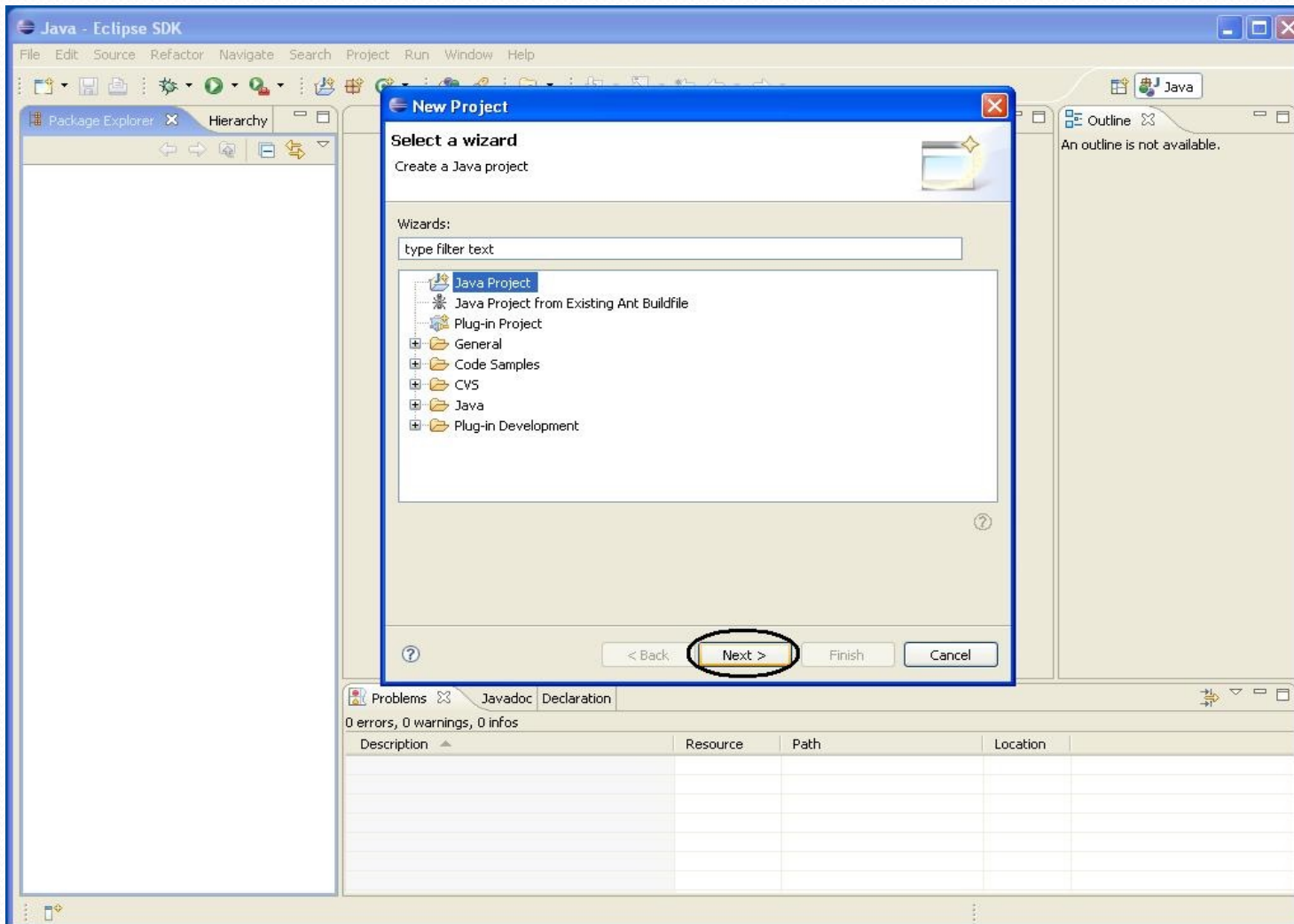
Step2: In Eclipse whenever you want to create a class, you need to select the new project by default (File → New → Project).
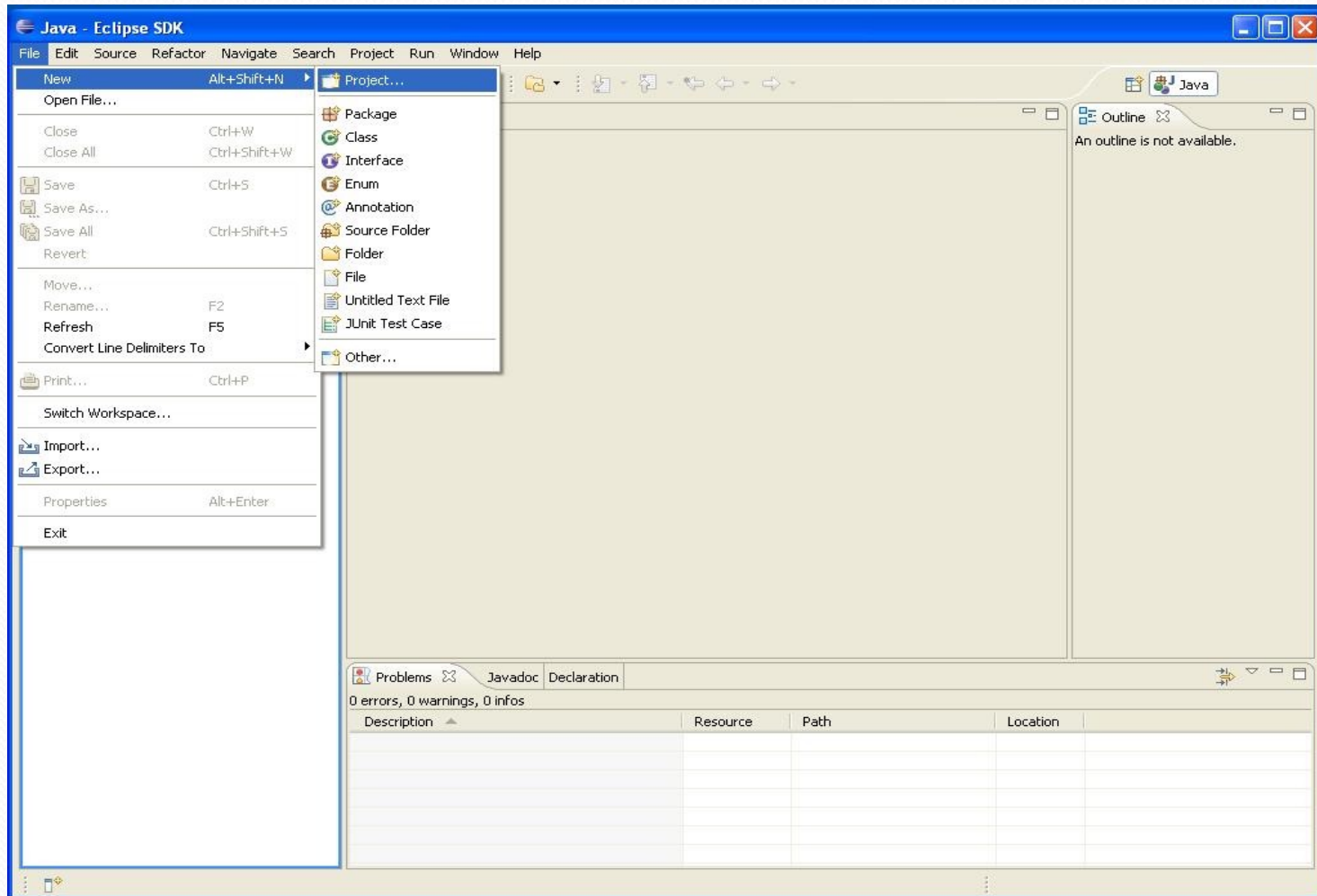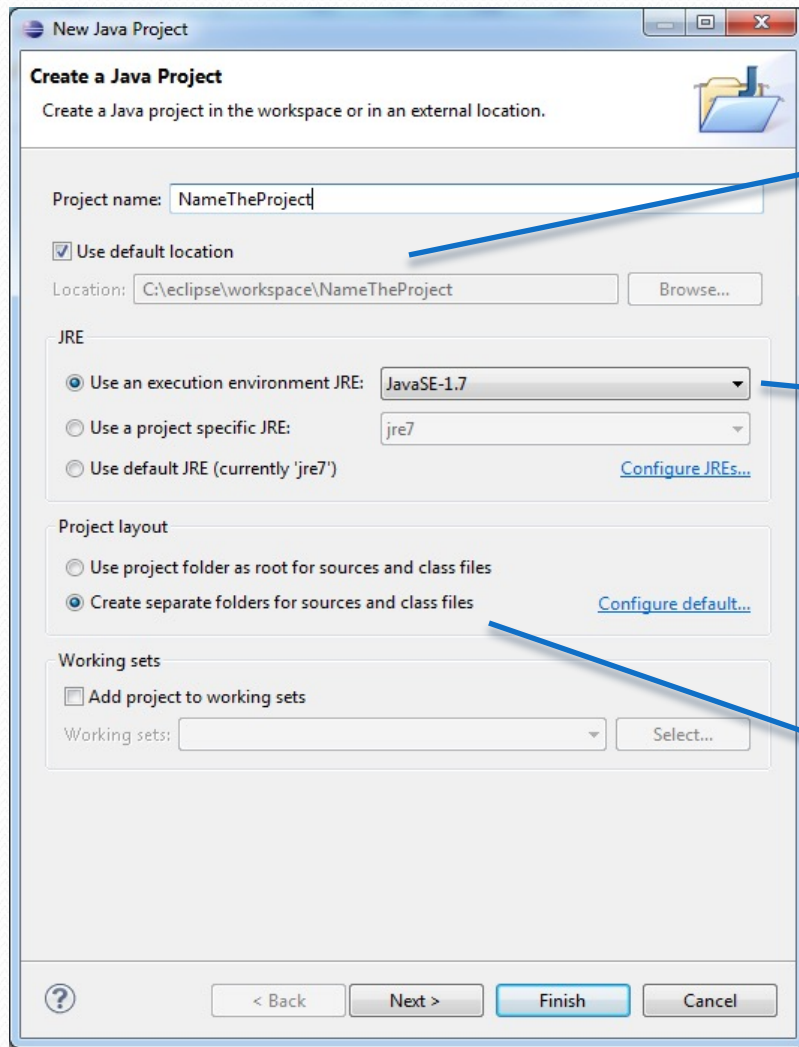
Step3: Select *Java project* and click next.

You can also choose *File* → *New* → *Java project* directly.

Step4: Name the project and click finish.

Creates the directory with the specified «project name» in the workspace path.

Specifies an execution environment to be used for the new project (Java Runtime Environment).
Select: **JavaSE-1.8**

Creates a source folder (*src*) for Java source files and an output folder which holds the class files of the project. When selecting the other option, the project folder is used both as source folder and as output folder for class files.

Step5: IDE views

- **Now, make a choice…**
  - Create a file directly
  - Import a file
  - Creating/Exporting a .jar file
  - F A Q

Step6: Now we create the java file by selecting the "File" menu, then "New" and "File".
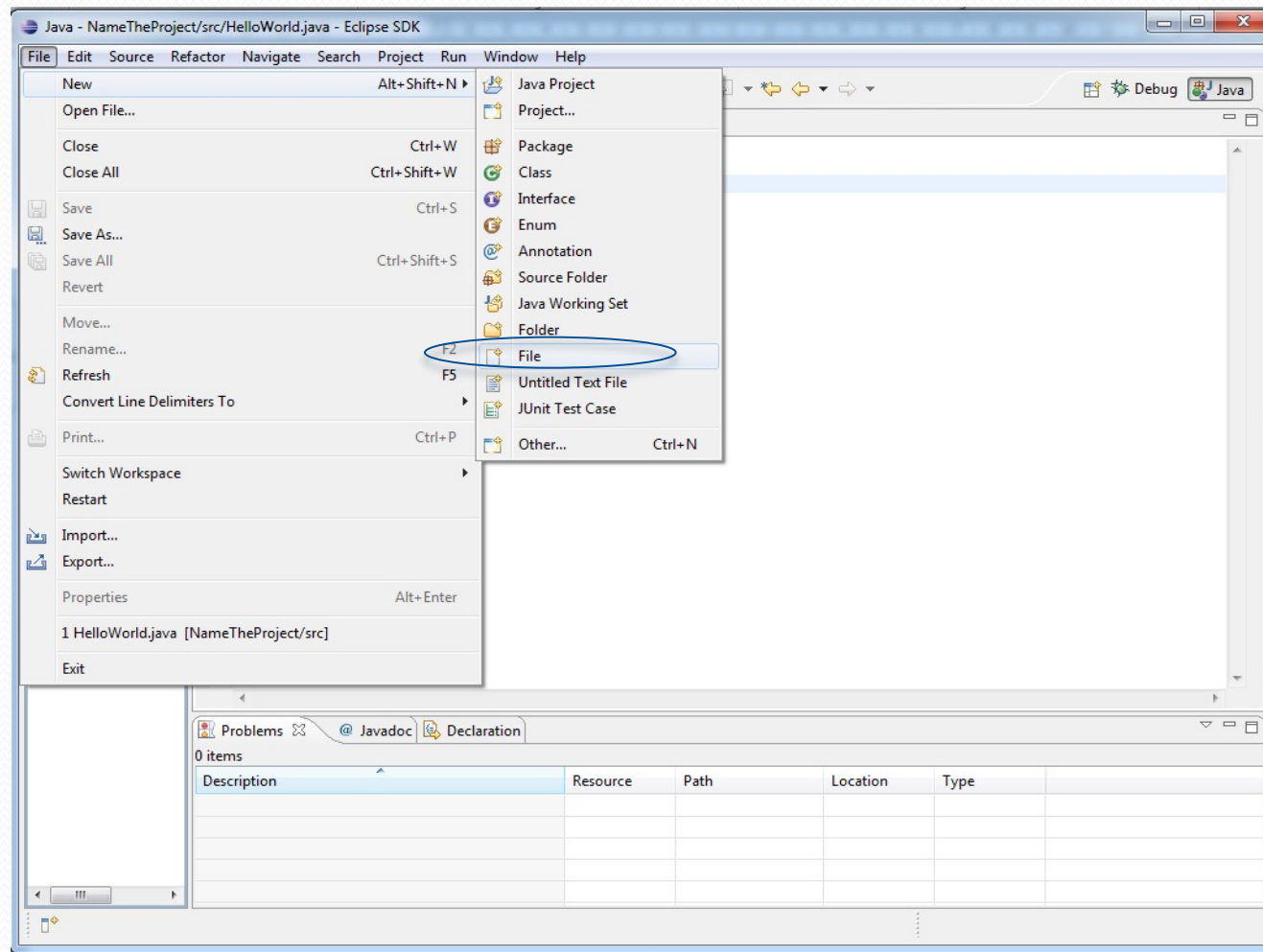
Step7: Now select the project in which you want to write, and the *src* folder if present. Then name the java class with extension (".java") and click "Finish".
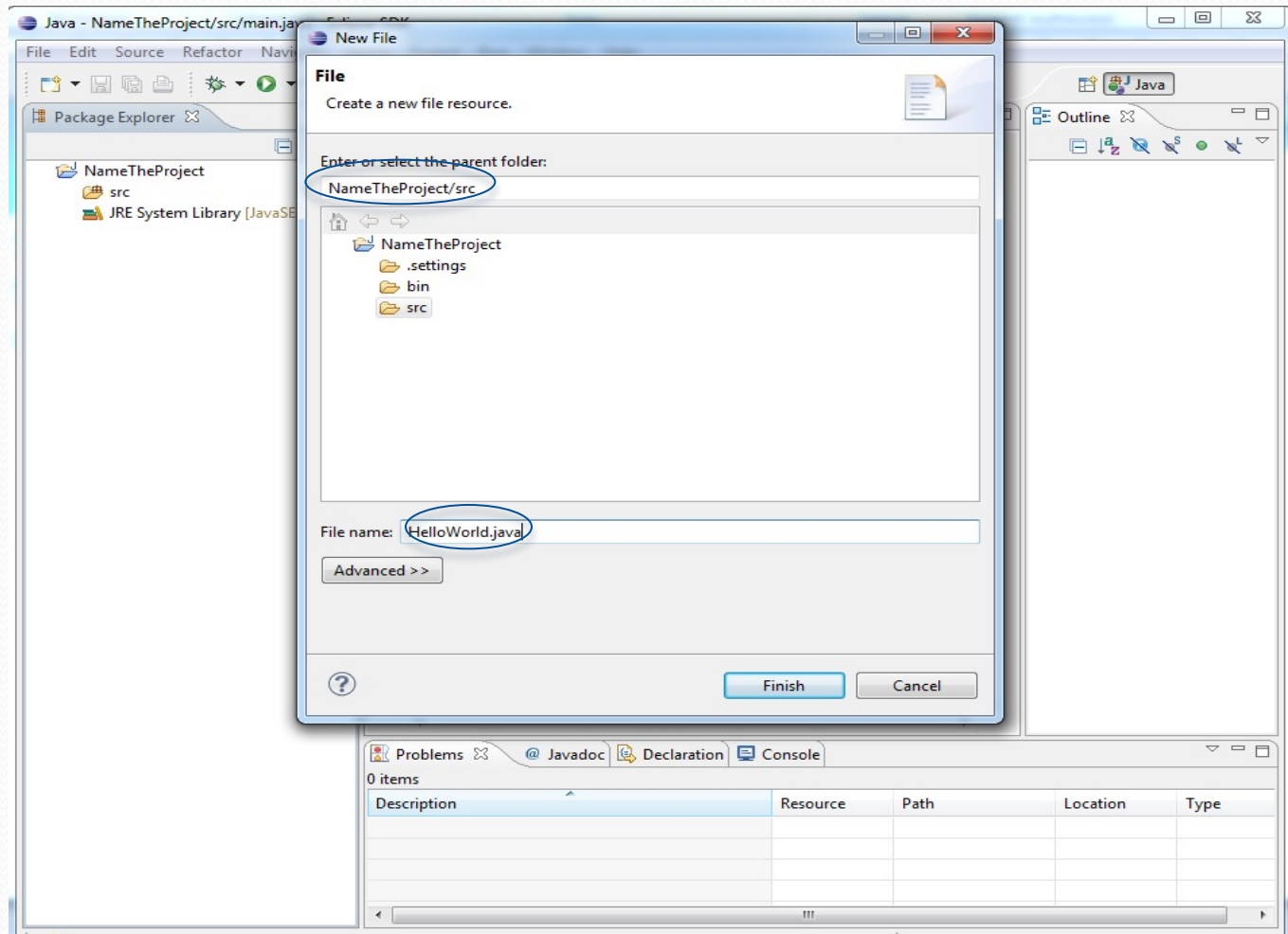
Step8: Now you have the editor space, start coding.
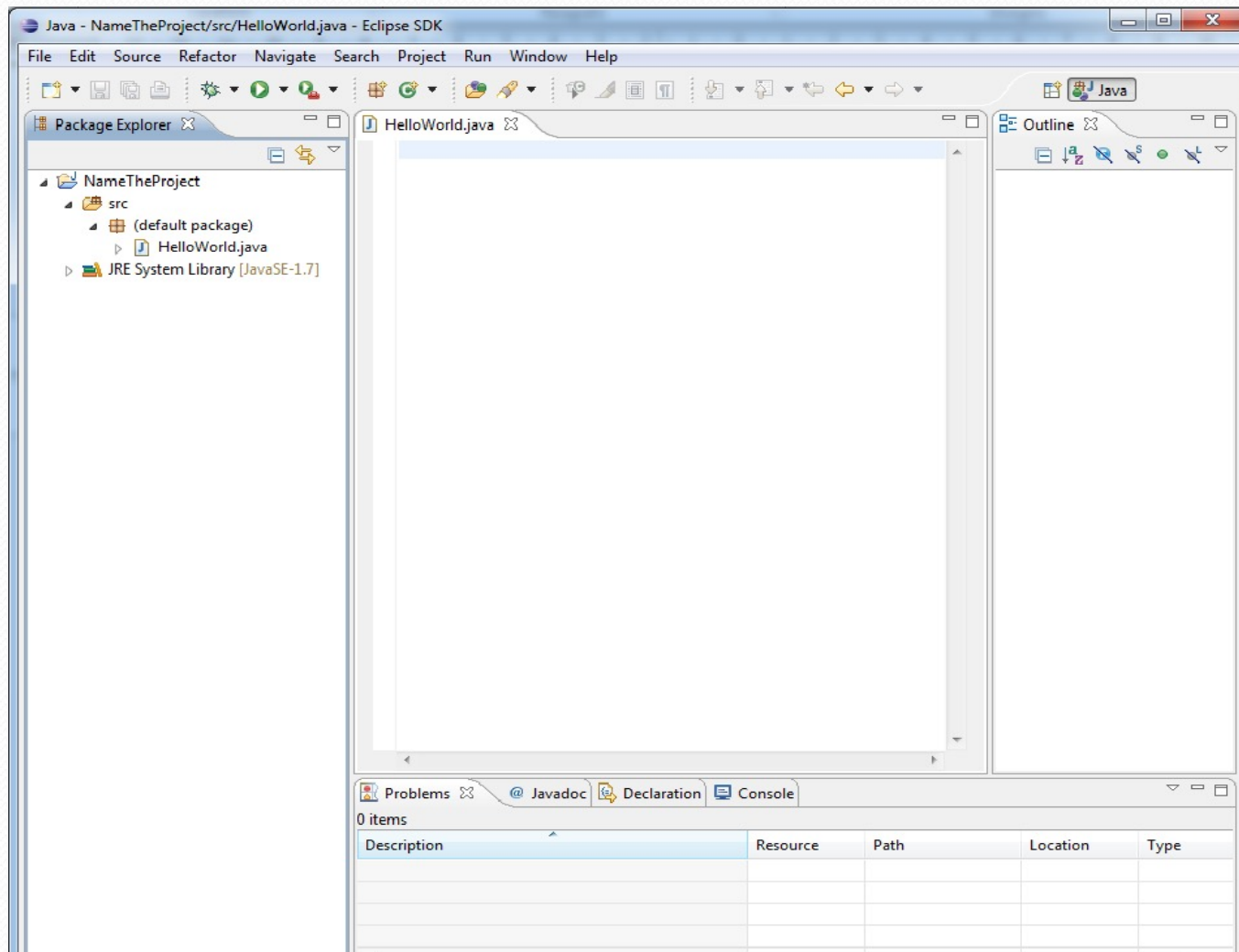
Writing the code

Step9: In Eclipse when ever you save the file, it will compile the code by default.

- Basic tip: Class name and the file name should be same.
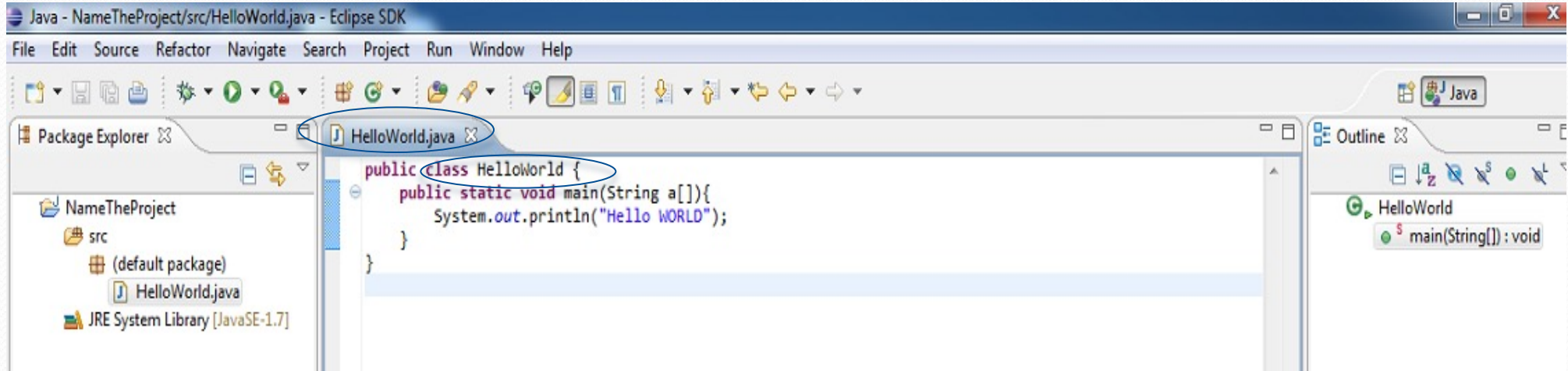
Step10: Running the java class. Right click on the class file and choose "Run as Java Application".
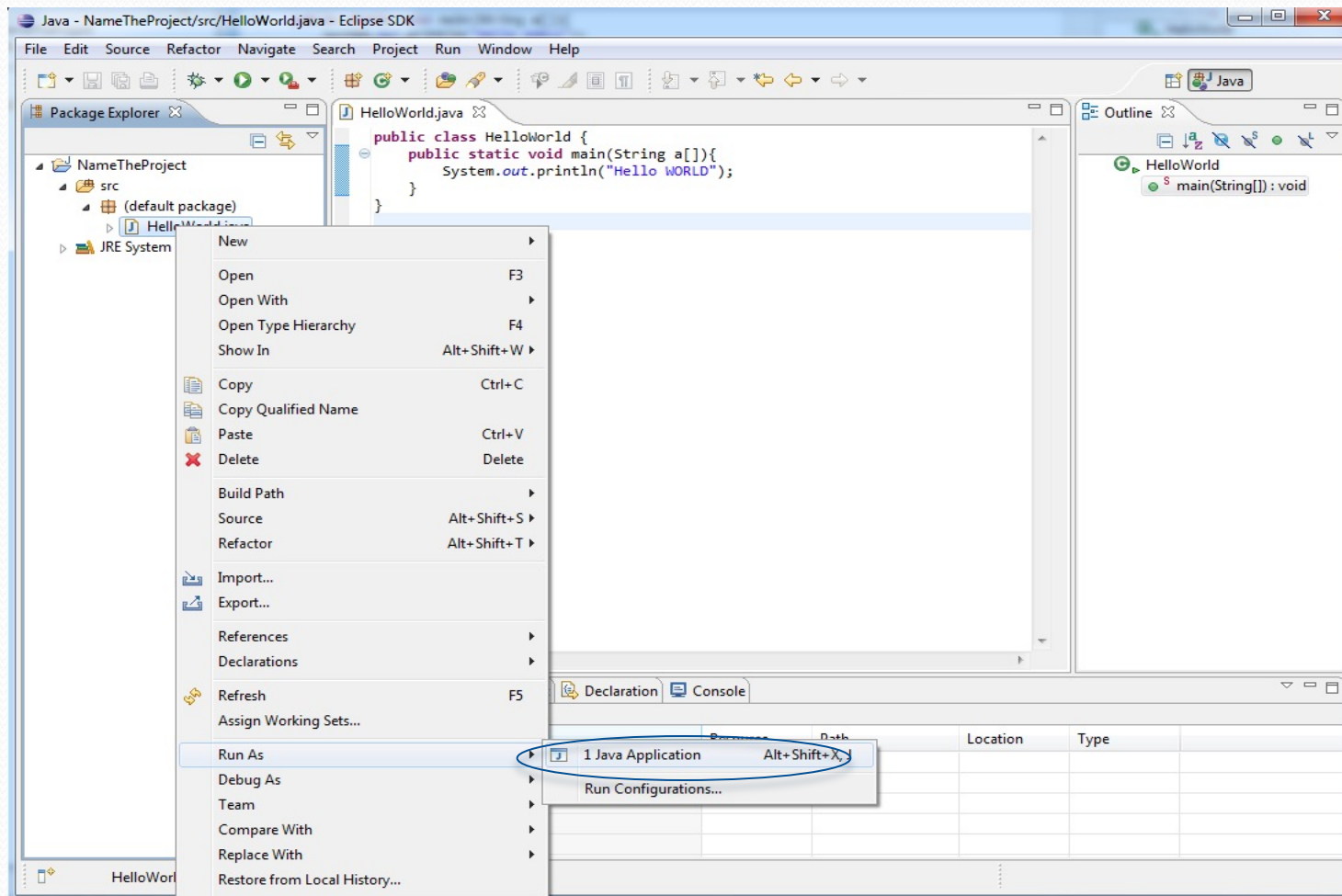
Step11: Here you can find the output (Console).
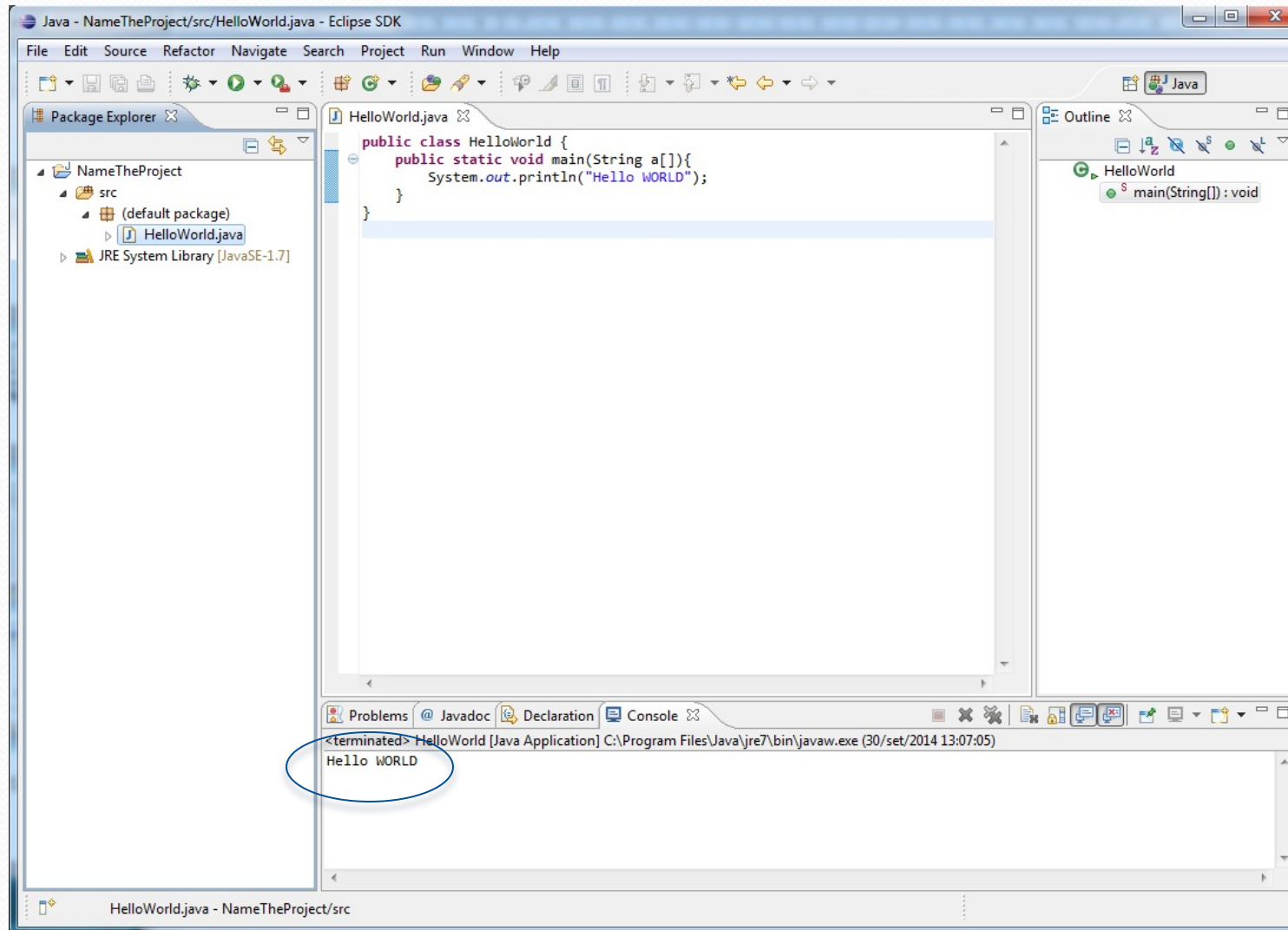
Using the command line

- Type the Java program using an editor (vi/emacs on Linux, Notepad on Windows, etc.) and save it with the *.java* extension

- Compile the program with the Java compiler:

  - javac HelloWorld.java

  - The compiler produces a .class file called HelloWorld.class. It translates the Java source code into bytecodes for the Java Virtual Machine (JVM)—a part of the JDK.

Using the command line

- The JVM is invoked by the **java** command. For example, to execute the example Java application type:
  - java HelloWorld
  - The output will be shown in the command window.
  - In general (more than one source file), specify the name of the class which is the application's entry point (contains the main() method)

Features of Eclipse

- Eclipse has the basic features required for editing, running, and debugging Java code. In addition to basic programming features, Eclipse support for **more advanced Java development tools** such as Ant, CVS, git, JUnit, and refactoring.
- Eclipse has a complete and easy-to-use help system
- Eclipse's GUI builder is a separate component.

# Running code

- Eclipse uses an incremental compiler, so it isn't necessary to explicitly compile your Java files; **the compiled class files are saved automatically when you save your Java files.**

- To run a program, the easiest way is to **select the file containing a main**() method in the Package Explorer and then select **Run** > **Run As** > **Java Application** from the main Eclipse menu.

Debugging

- First, set a breakpoint in the main() method by double-clicking in the left margin next to the call. If this code were a little less trivial, it would also be possible to set a conditional breakpoint -- one that stops when a particular expression is true, or one that stops after a specific number of hits -- by right-clicking the breakpoint and selecting **Breakpoint properties** from the context menu.
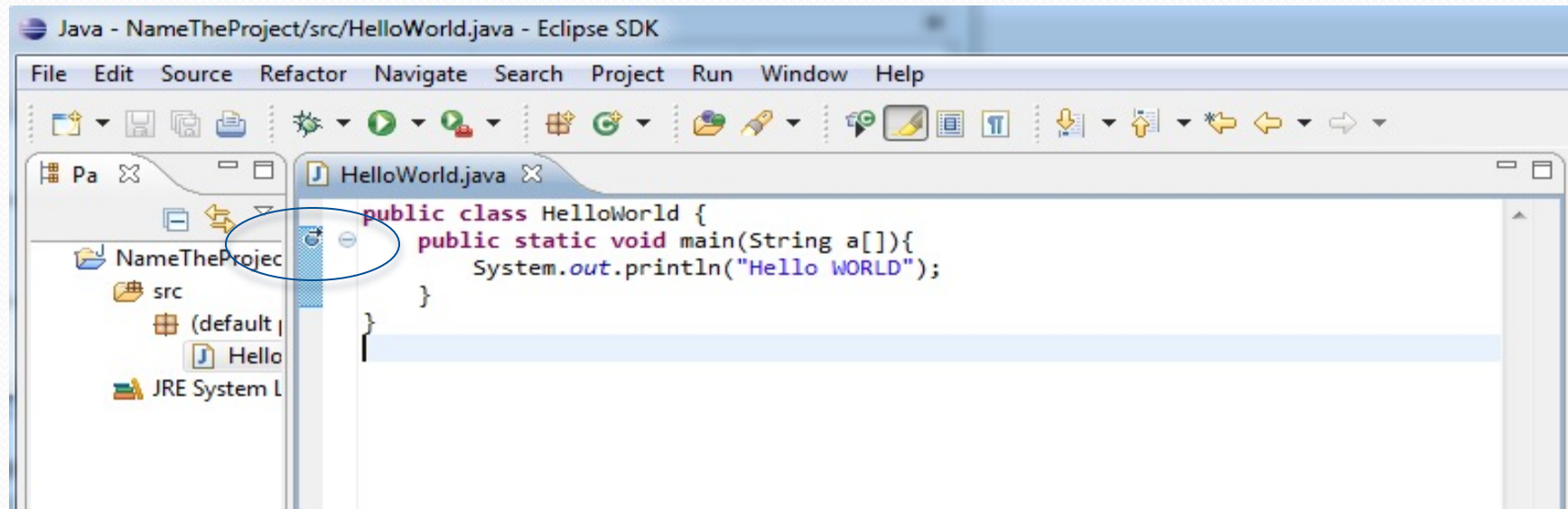
Debugging

- To start debugging, select **Run > Debug As > Java Application** from the main menu. Because Eclipse has a *Debug perspective* that is better suited for debugging than the Java perspective, it will ask if you want to change to this perspective - click Yes.

Debug Perspective

New Debug perspective – click Java to exit

Buttons to step thorugh the code. Shortcuts available from Run menu

List of breakpoints

Variables in the scope with their current values.

Line of code where we stopped

Output Console

Step **into**

- Step **into** will cause the debugger to descend into any method calls on the current line. If there are multiple method calls, they'll be visited in order of execution; if there are no method calls, this is same as step over. This is broadly equivalent to following every individual line of execution as would be seen by the interpreter.

Step **over**

- Step **over** proceeds to the next line in your current scope (i.e. it goes to the next line), without descending into any method calls on the way. This is generally used for following the logic through a particular method without worrying about the details of its collaborators, and can be useful for finding at what point in a method the expected conditions are violated.

Step **out**

- Step **out** proceeds until the next "return" or equivalent - i.e. until control has returned to the preceding stack frame. This is generally used when you've seen all you need at *this* point/method, and want to bubble up the stack

The famous Dos and Don'ts:

- Set the workspace of Eclipse where you can easily access it.
- **Never start writing the code without making a project**. You need to create a project folder every time you start a new assignment
- **Main classname and the file name** should always match.

Strange Error:

- The Eclipse IDE reports a strange error similar to: "Cannot create workbench".

- In many situations this problem can be resolved by deleting the workspace /.metadata /.registry file in user's home directory and restarting the IDE.