Selenium Automation Testing

(JAVA, MAVEN, TESTNG, LOG4J, EXTENT REPORT, APACHE POI, SELENIUM, JENKINS, GITHUB)

Contents

[1. Outline: what is in the class 4](#_Toc467781941)

[2. Contact Us 8](#_Toc467781942)

# OutLine: What is in Selenium Automation Testing Class:

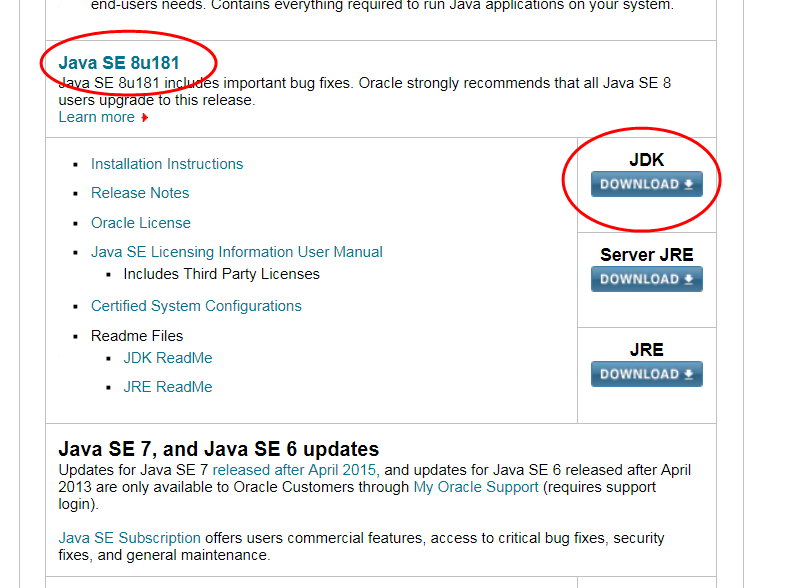
**Section 1: Selenium History**

1. Web Testing Automation
2. What is Selenium
3. Selenium Features

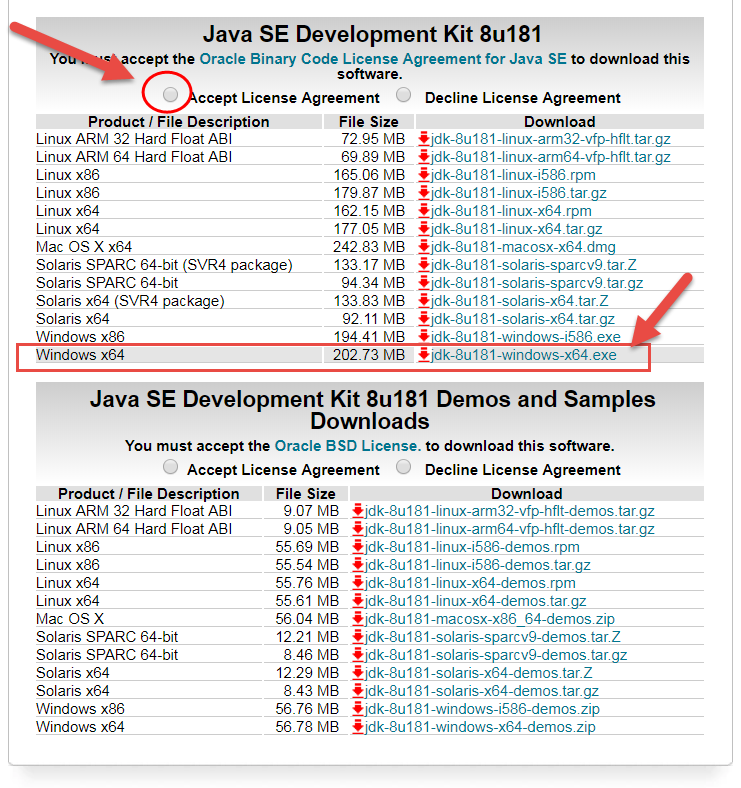
**Section 2**: **Java Essentials for Selenium**

1. ***Download and install Java***

* Install JDK (Java development kit) from: <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

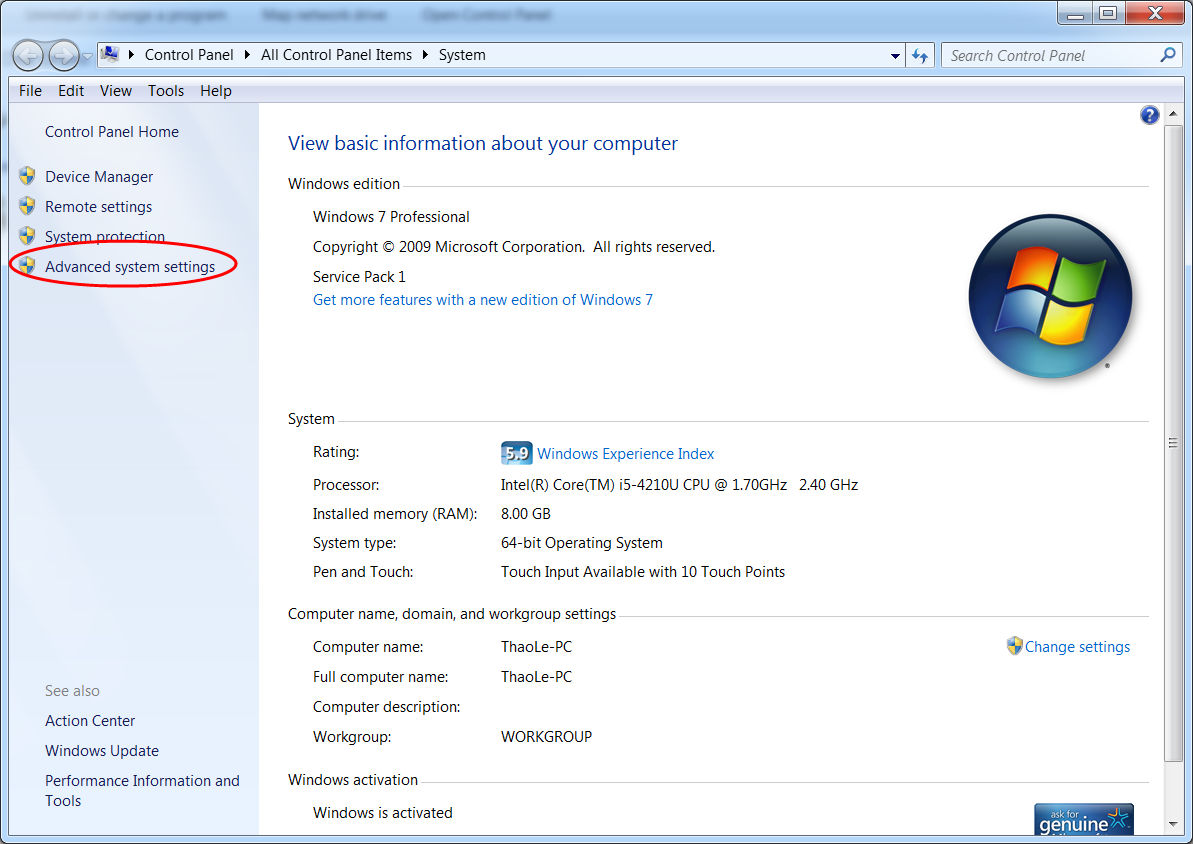


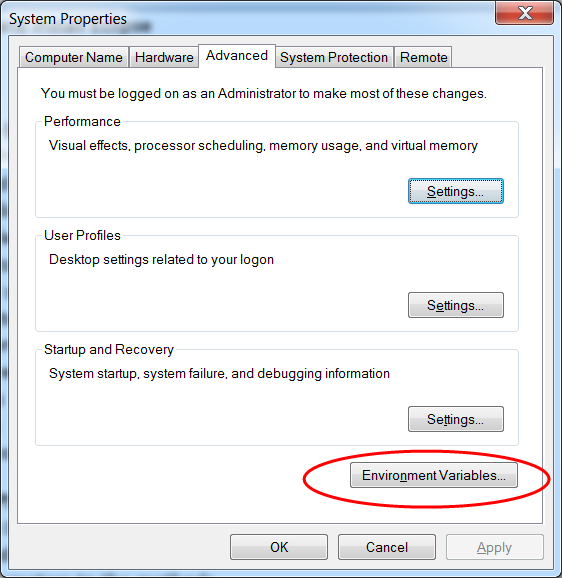
* Select “Accept License Agreement”, then if your computer OS is window 64 bits, then clicking on Window 64 bits link same as screen-shot below:



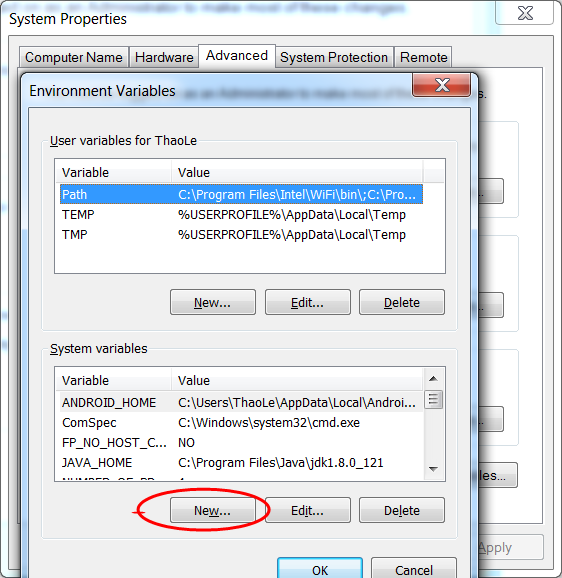
1. Setup Java Environment

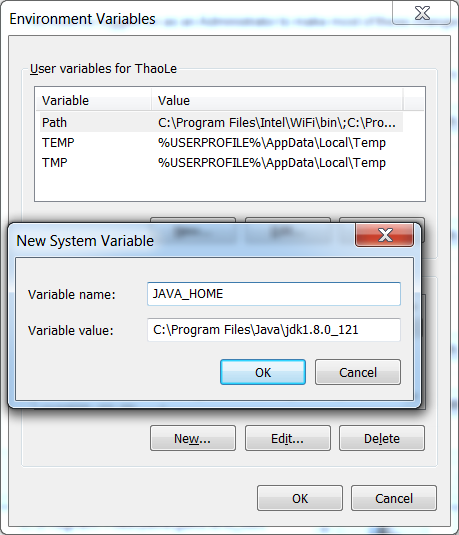
* Open System Properties from “Advance System Settings”



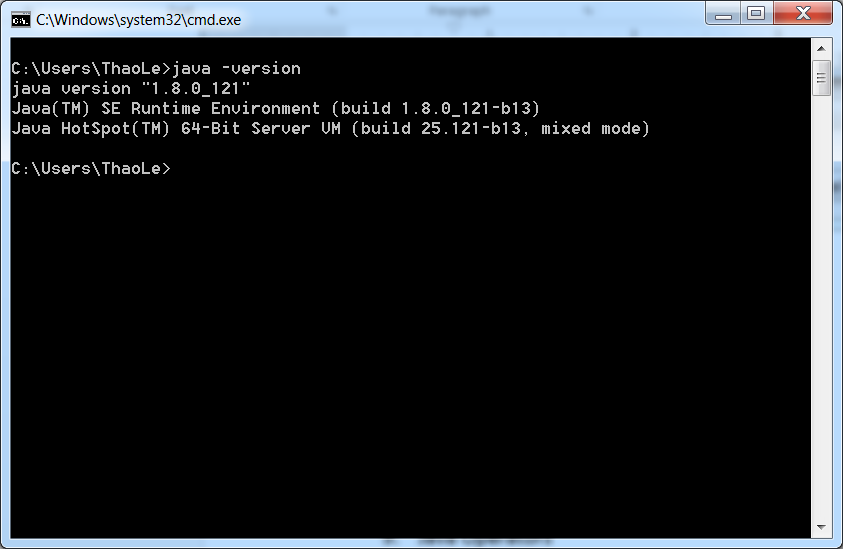


* Add a new “System Variable” name JAVA\_HOME



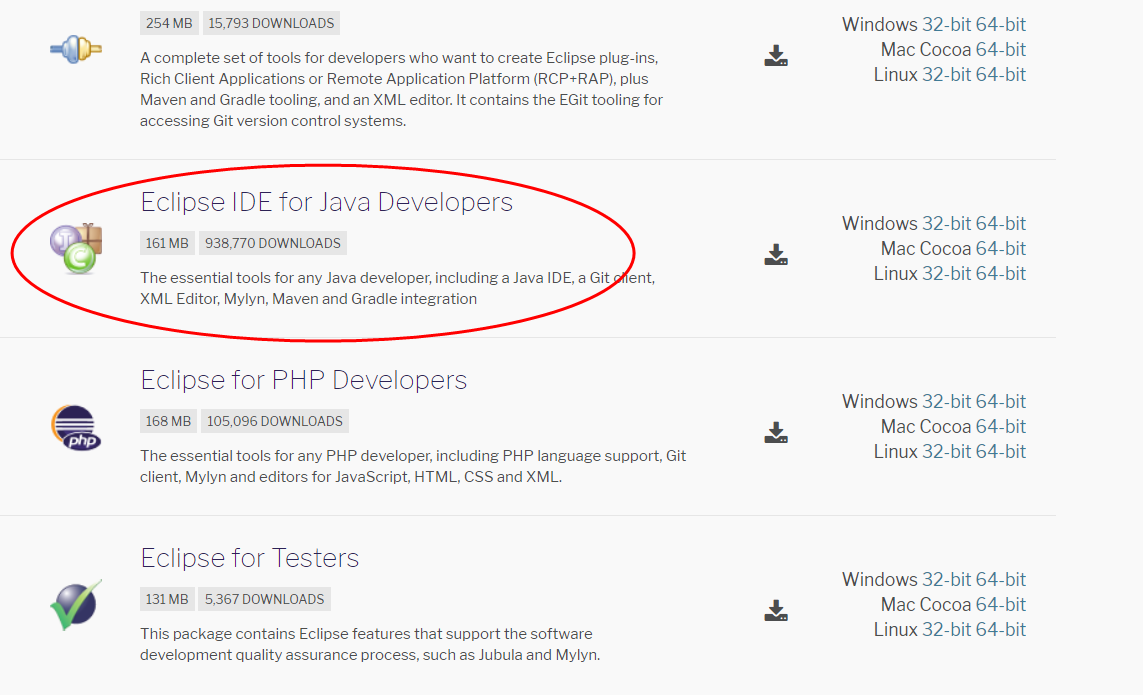


* Run “java –version” from command line to check if java is installed successfully.



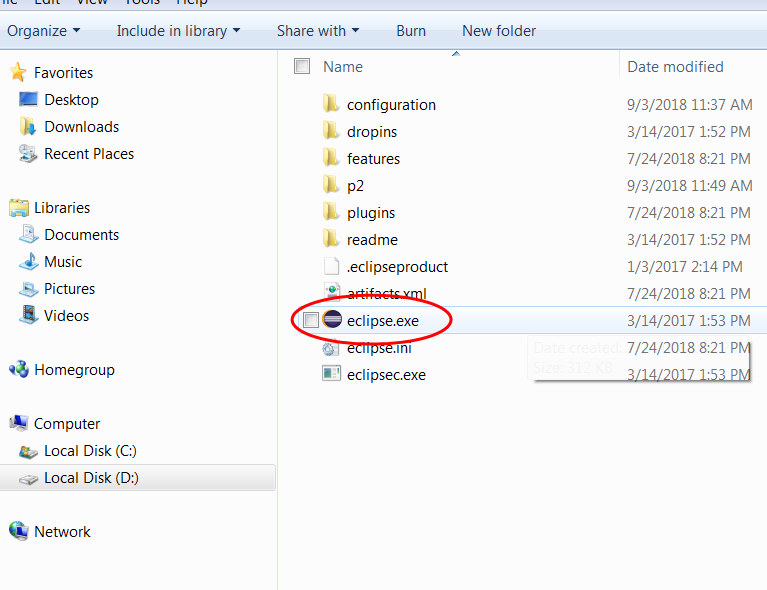
1. Download and install Eclipse

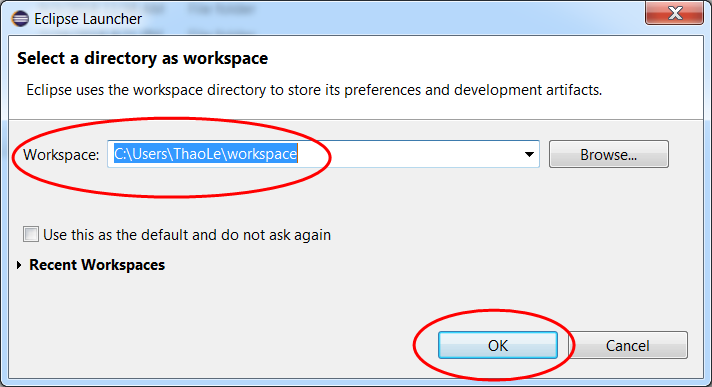
* Download and install Eclipse IDE: <https://www.eclipse.org/downloads/packages/release/neon/3>

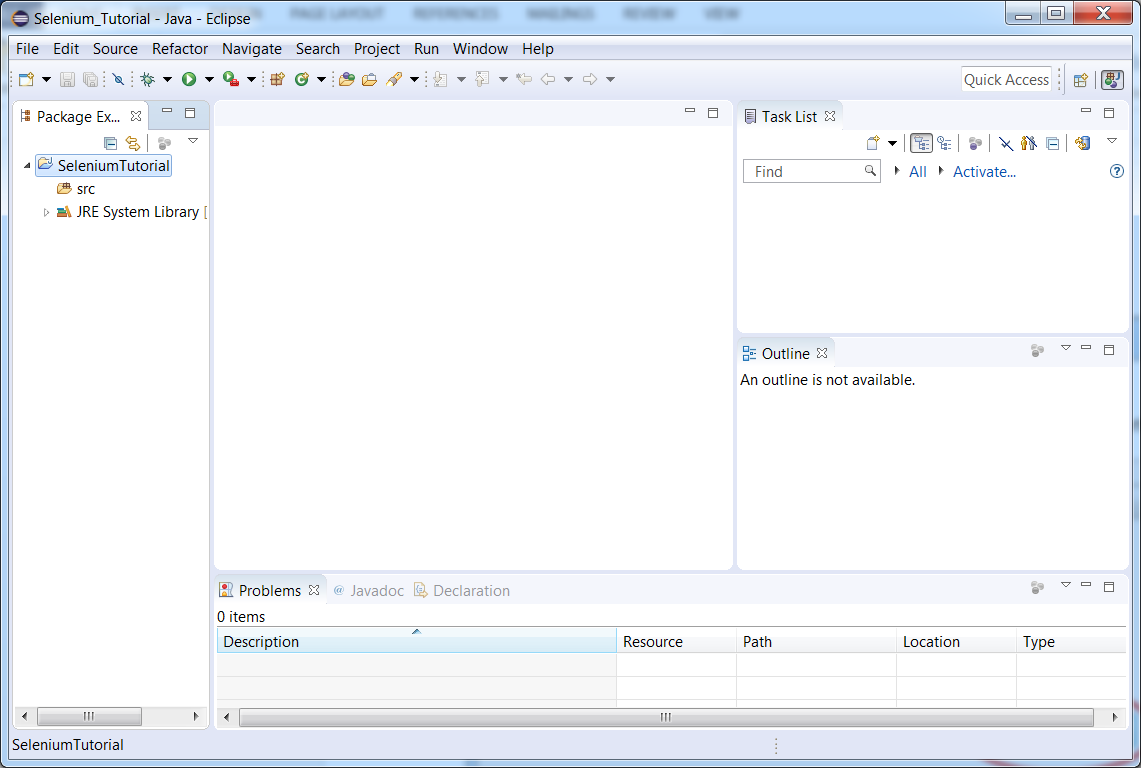


1. How to use Eclipse

* Open Eclipse and choose your work space

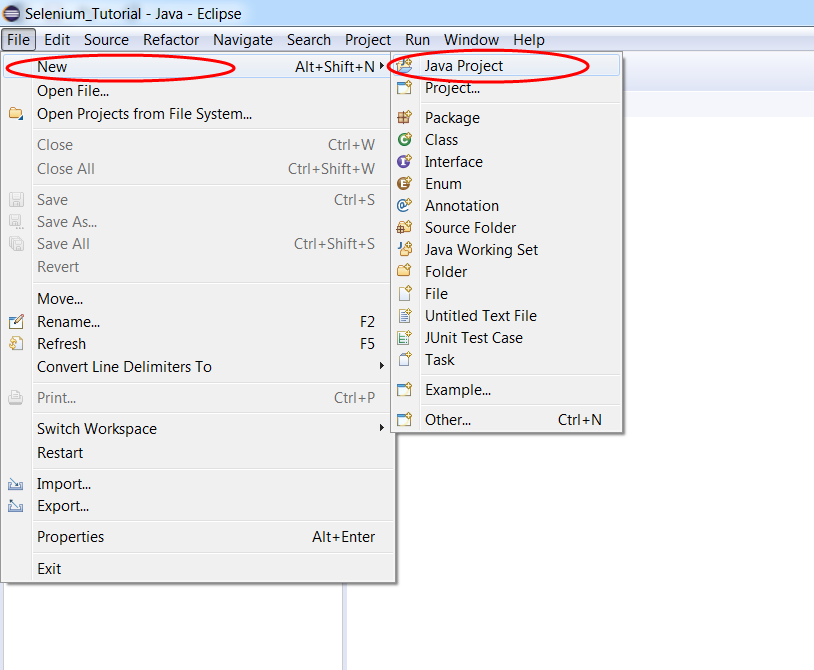


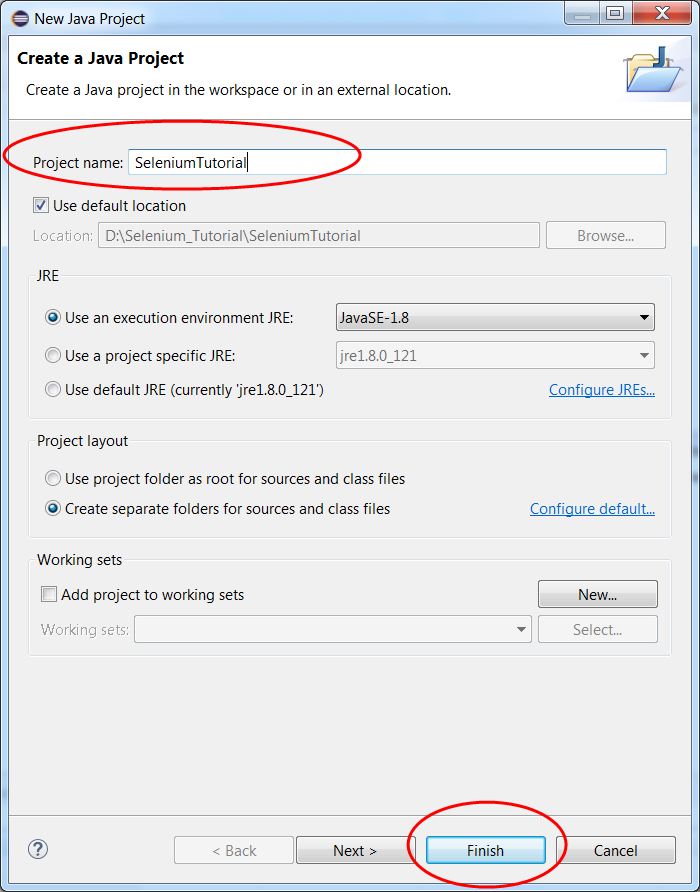




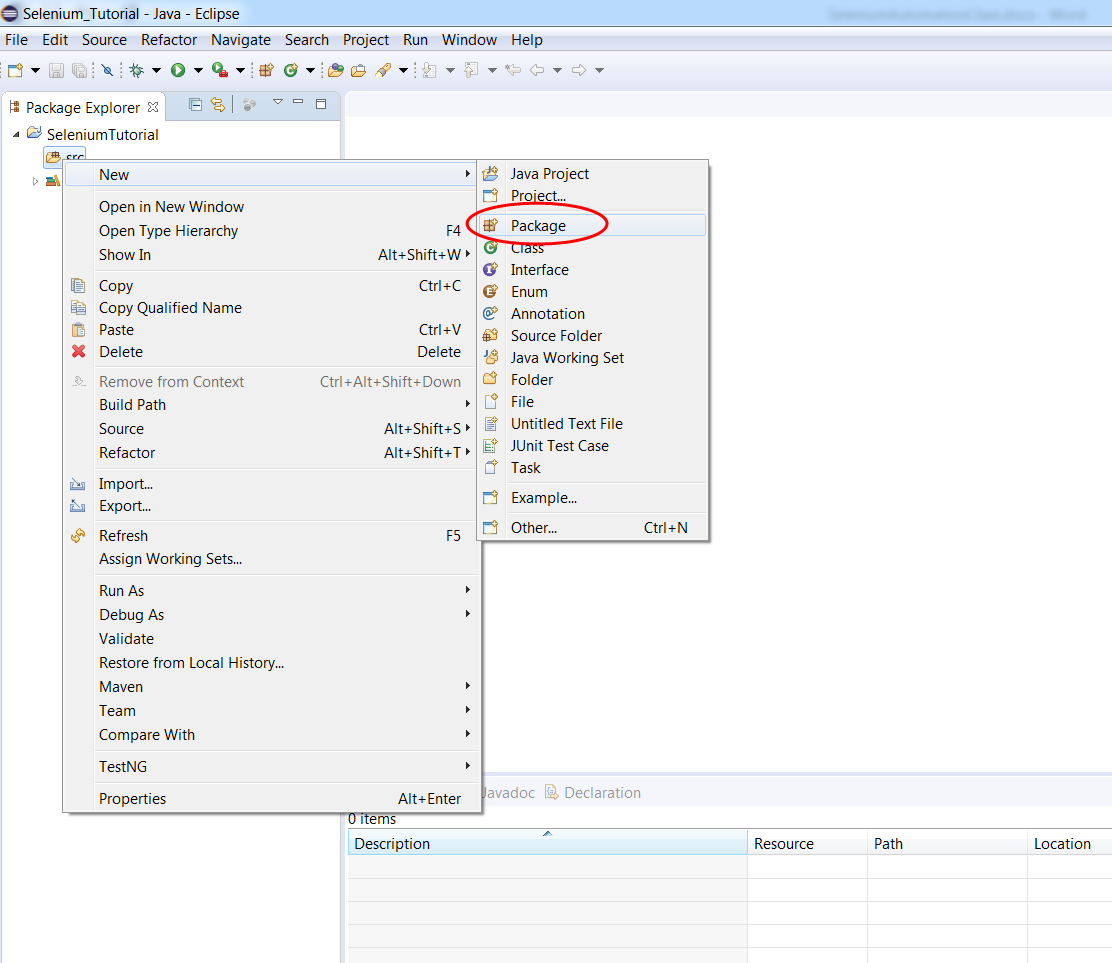
1. Basic Java Program

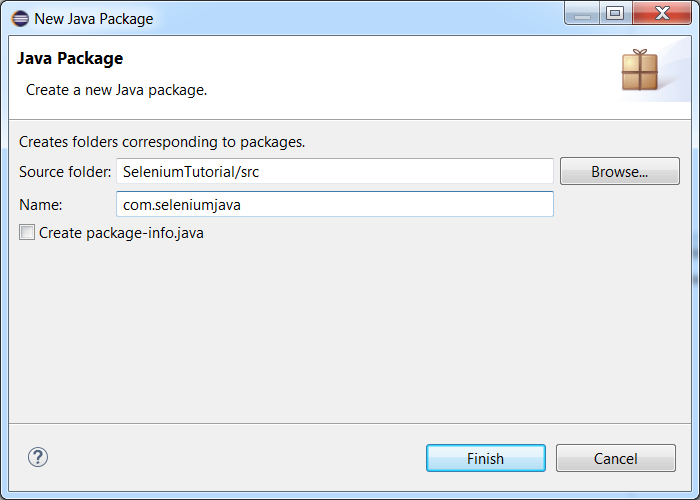
* Create new java project: click on File, select New, select “Java Project”, enter “SeleniumTutorial” in project name





* Create new Package named “com.seleniumjava” in src





* Create a “HelloWorld” class with java code of “HelloWorld” as follow:

**package** com.seleniumjava;

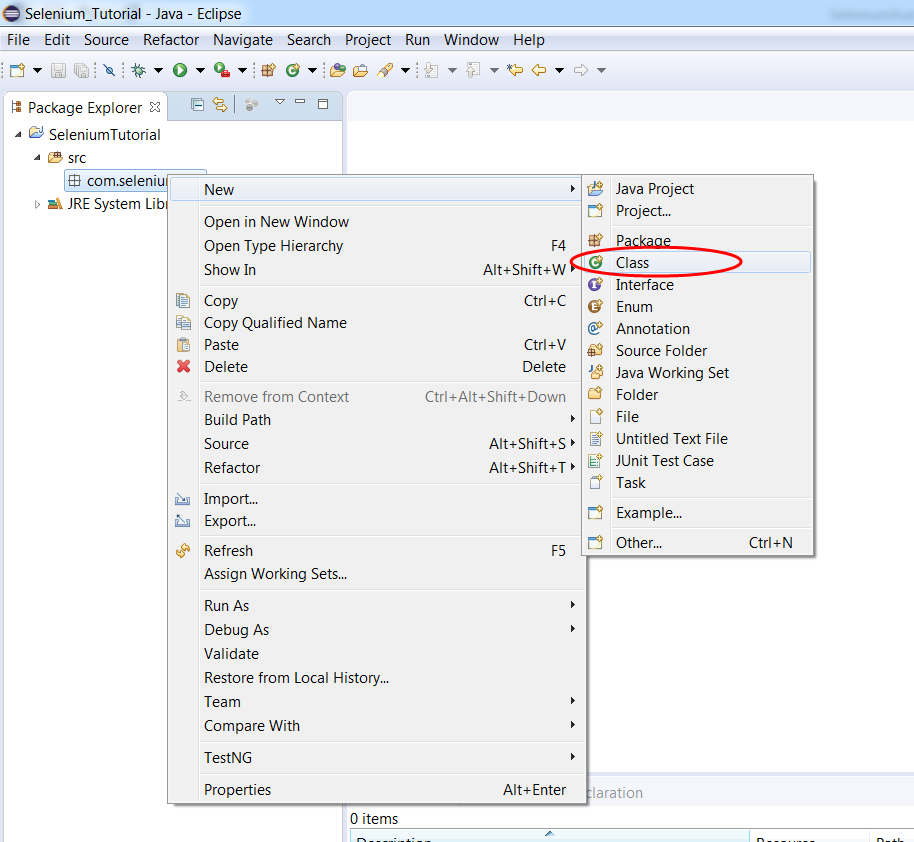
**public** **class** HelloWorld {

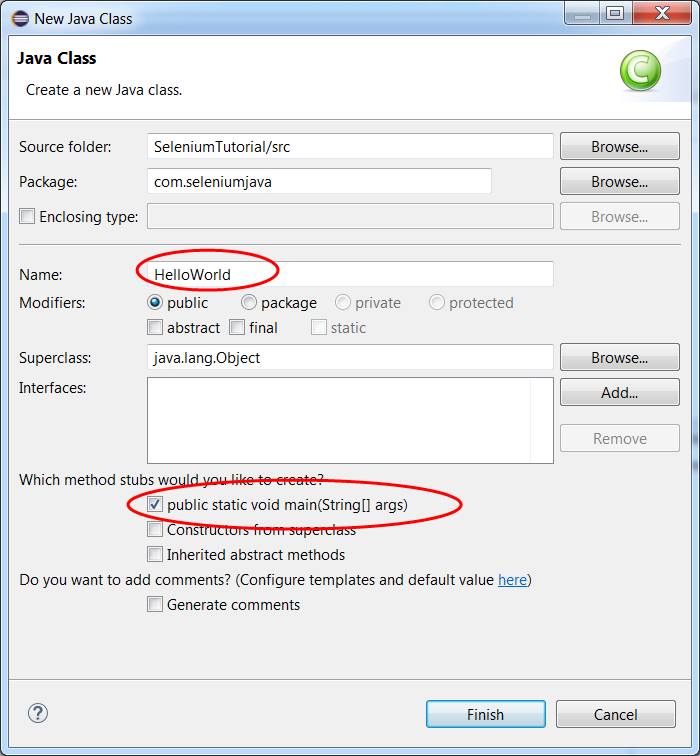
**public** **static** **void** main(String[] args) {

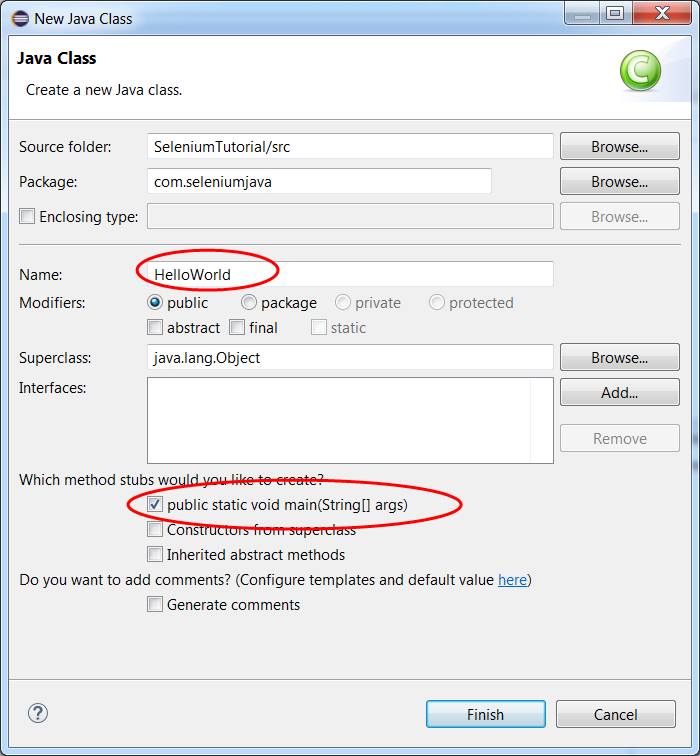
System.***out***.println("Hello World");

}

}

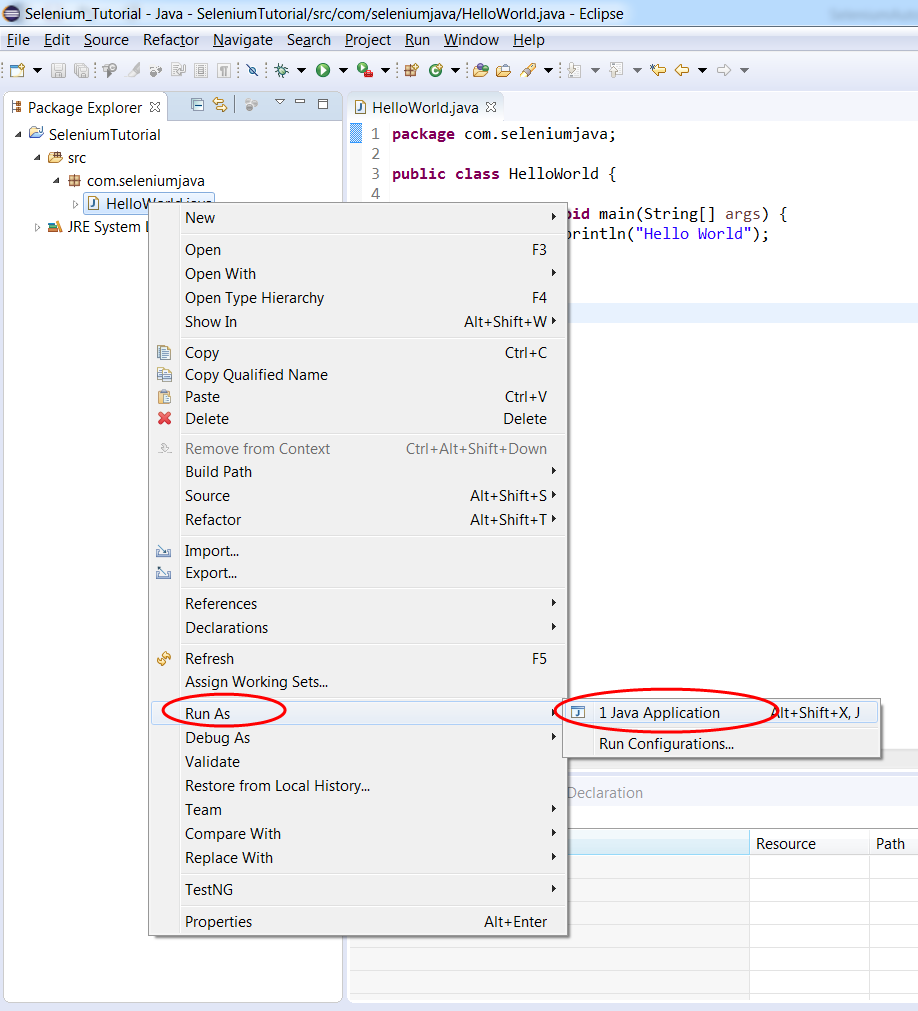




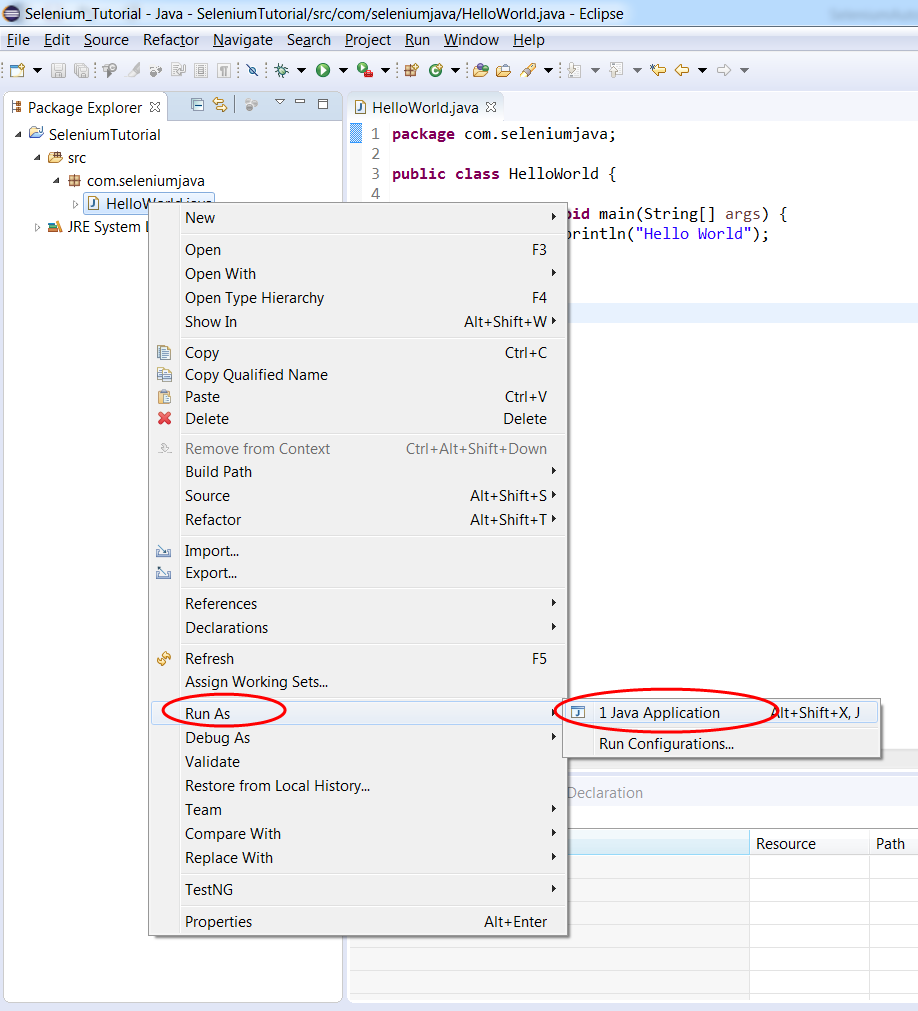


1. Compile and run a Java program

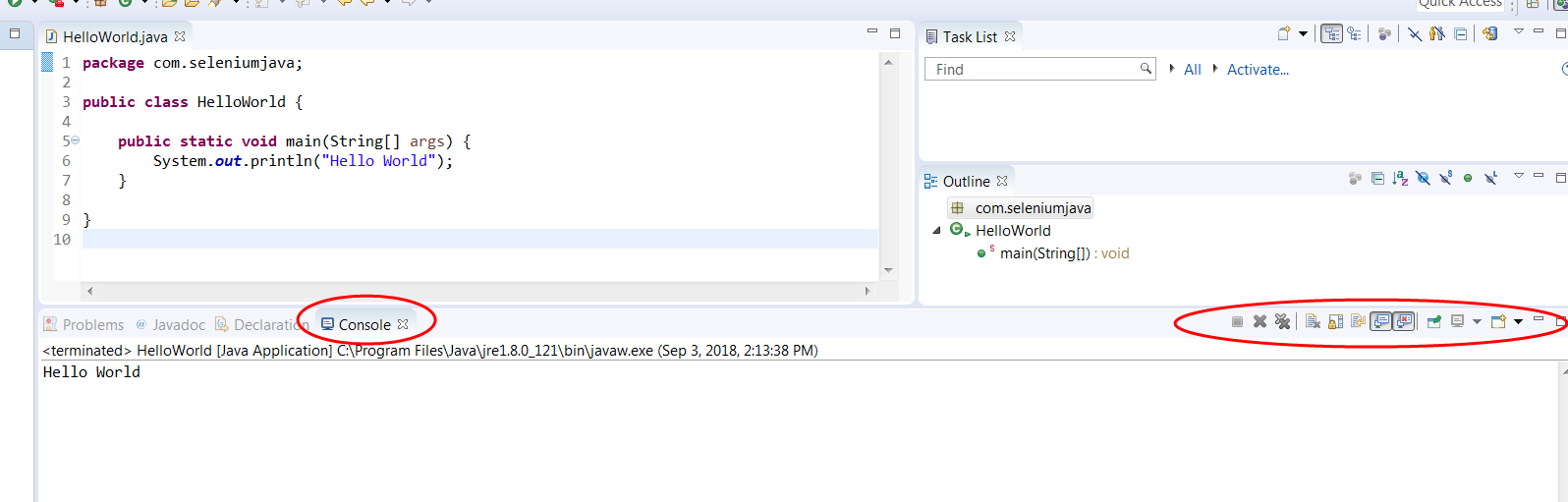
* Right click on HelloWorld.java, then select Run As, “Java Application”



* “Hello World” will be displayed in Console Output



1. Understanding console output



## Console view toolbar

The table below lists the toolbar options displayed in the **Console** view. Note that toolbar options may differ between the various consoles.

| Icon | Name | Description |
| --- | --- | --- |
| next error icon | Next Error | Show the next error in the Console view. |
| previous error icon | Previous Error | Show the previous error in the Console view. |
| show error icon | Show Error in Editor | Toggle to show the selected error in the Editor view (default) or not. |
| save output file icon | Save Console Output to File | Opens the Properties for Project > C/C++ Build dialog box where you can specify the Build output file location. |
| Scroll Lock icon | Scroll Lock | Toggles the Scroll Lock. |
| Clear Console icon | Clear Console | Clears the current console. |
| Pin Console icon | Pin Console | Forces the Console view to remain on top of other views in the window area. |
| Display Selected Console icon | Display Selected Console | If multiple consoles are open, you can select the one to display from a list. |
| open Console icon | Open Console | Opens a new console to display the log for:   * New Console View * Program Output Console * Windows Program Output Console * CVS, and others based on installed features |

## Console view context menu

When you right-click in the **Console** view (or when you press **Shift+F10** when the focus is on the **Console** view), you see one or more of the following options in a context menu depending on the console view visible:

| Name | Description |
| --- | --- |
| **Clear** | Clears the current console. |
| **Editor options: Cut, Copy, Paste, Select All** | These options perform the standard edit operations. Which options are available depends on where the focus is in the **Console** view. For example, you cannot paste text into the program output, but you can paste text to the bottom of the file. |
| **Go to Line** | Opens a dialog that moves the focus to the line you specify. The dialog also indicates the total number of lines in the console buffer. |
| **Find/Replace...** | Opens a **Find/Replace** dialog that operates only on the text in the **Console** view. |
| **Open Link** | Select to follow a detected hyperlink in any console view that supports. For example, the CVS console. |
| **Scroll Lock** | Toggles the Scroll Lock. |

1. **Java Variables and Data Types**

**Java Variables:**

* **Instance Variables (Non-Static Fields)** Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class (to each object, in other words); the currentSpeed of one bicycle is independent from the currentSpeed of another.
* **Class Variables (Static Fields)** A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code static int numGears = 6; would create such a static field. Additionally, the keyword final could be added to indicate that the number of gears will never change.
* **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to declaring a field (for example, int count = 0;). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.
* **Parameters** You've already seen examples of parameters, both in the Bicycle class and in the main method of the "Hello World!" application. Recall that the signature for the main method is public static void main(String[] args). Here, the args variable is the parameter to this method. The important thing to remember is that parameters are always classified as "variables" not "fields". This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you'll learn about later in the tutorial.

# Primitive Data Types

* **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large [arrays](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html), where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.
* **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a shortto save memory in large arrays, in situations where the memory savings actually matters.
* **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the [Integer](https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html) class to support the arithmetic operations for unsigned integers.
* **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 264-1. Use this data type when you need a range of values wider than those provided by int. The [Long](https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html) class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.
* **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the [java.math.BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html) class instead. [Numbers and Strings](https://docs.oracle.com/javase/tutorial/java/data/index.html) covers BigDecimal and other useful classes provided by the Java platform.
* **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
* **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
* **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

|  |  |
| --- | --- |
| **Data Type** | **Default Value (for fields)** |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

1. **Java Operators**

1. Conditional statements
2. Looping statements
3. Single Dimensional array
4. Double Dimensional array
5. Object class
6. String Class
7. String methods

**Section 3**: **Java OOPS Concepts**

1. Classes and Objects
2. Java methods
3. Passing parameters to the methods
4. Call by value and call by reference
5. Java Constructor
6. Method Overloading
7. Constructor Overloading
8. this, final keywords
9. Static variables and methods
10. Java Inheritance
11. Method Overriding
12. Java Interfaces
13. Access Modifiers
14. Java Packages
15. Exception Handling
16. Array List
17. Hash Map
18. JDBC

**Section 8: Maven Integration with Selenium**

1. What is Maven and Why Maven?
2. Installing/Configuring Maven
3. Creating Maven Project
4. Importing Maven Project into Eclipse
5. What is POM.xml?
6. Adding Dependencies to POM.xml

**Section 4: Selenium Webdriver**

1. Create Maven Project in Eclipse
2. Create WebDriver test case
3. Execute test case on multiple browsers
4. Locators
5. What are locators?
6. Types of Locators
7. Capturing Xpath on IE, Chrome and Firefox browsers
8. Types Xpath
9. Writing Xpath
10. Built-in functions in XPath
11. Handling Dynamic elements using Xpath
12. WebDriver commands
13. Browser commands
14. Get commands
15. Switching commands
16. Navigation commands
17. Conditional commands
18. Implicit, Explicit and Fluent Waits
19. Handling Web Elements
20. Textbox/Input box
21. Web Button
22. Radio Button
23. Checkbox
24. Dropdown box/Combo box
25. Bootstrap dropdown
26. List box
27. Calendar/Date Picker
28. Web Links
29. HTML frames/Iframes
30. Web/HTML Tables
31. Mouse actions using Actions class
32. Mouseover
33. Mouse double click
34. Mouse right click
35. Drag and Drop
36. Handling Slider/scroll bar
37. Resizing
38. Handling Tooltips
39. File Upload and Download

**Section 5: Data Driven Testing using Excel**

1. What is data driven testing?
2. Usage of Apache POI API
3. Read data from Excel file
4. Write data into Excel file
5. Data Driven Testing using Excel

**Section 6: TestNG**

1. What is TestNG
2. Install TestNg in Eclipse
3. How to write TestNG Test case
4. Annotations in TestNG
5. Understanding testng.xml
6. TestNG Report
7. Prioritizing tests
8. dependsOnMethods
9. Skipping tests
10. Grouping methods
11. TestNG batch testing
12. Parameterization
13. Passing parameters using xml
14. Parallel testing
15. Data Provider
16. TestNG Listeners

**Section 7: Advanced Concepts**

1. Extent Reports
2. Capturing Screenshots
3. Log4j for logging
4. Page Object Model - Creating Page objects
5. Handling cookies
6. Desired Capabilities in Selenium
7. Setting up Browser Profiles

**Section 9: Automation Framework**

1. What is Framework?
2. Types of Frameworks
3. Prerequisites for designing frameworks
4. Implementation of Hybrid Driven Framework
5. Creating Maven Project
6. Update pom.xml with dependencies
7. Creating page objects and Object repository
8. Creating utility files
9. Setting up configuration files
10. Creating automation test scripts
11. Creating data driven test scripts
12. Generating extent reports
13. Generating logs using log4j
14. Execute test scripts using TestNG XML File
15. Emailing test reports
16. Execute test scripts through Maven CLI
17. Execute test scripts using bat file

**Section 10: Continuous Integration (CI) - Maven, Jenkins & GIT**

1. What is Continues Integration?
2. Continuous Integration Tools
3. Download and install Jenkins
4. Maven integration with Jenkins
5. Run selenium test scripts through Jenkins
6. Downloading and Installing GIT
7. Installing GIT and GITHUB plug-ins for Jenkins
8. Uploading project to GIT

# Contact Us

**Email:** thaocanada2013@yahoo.com

**Phone:** 416-450-6714