**ICS 102 Lab01: Getting started with Java**

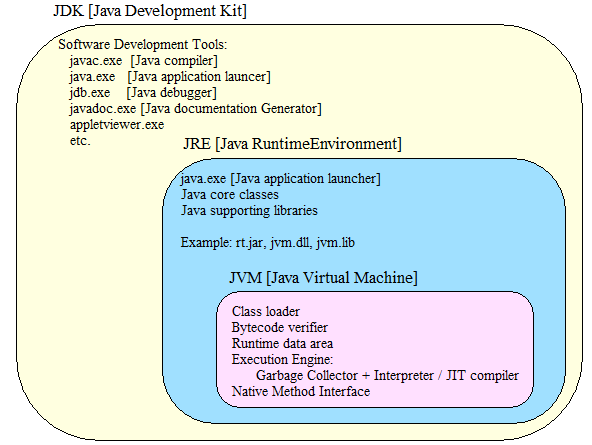
**Objectives:**

1. Learning how to use the Java API Documentation document.
2. Introduction to The Java Development and Execution environment.
3. Learning How to create, compile, and execute a simple Java program.
4. Introduction to the Eclipse IDE.
5. Introduction to the elements of a Simple Java program:
   1. Comments
   2. Indentation
   3. Literal strings
   4. The new line '\n' and tab characters '\t' within a literal string.
   5. Console output methods **System.out.print** and **System.out.println**.
6. Introduction to the types of Java Program Errors.

### The Java Development and Execution Environment

The main purpose of this lab is to introduce you to the computing environment of your laboratory. You will use the ideas in this lab repeatedly throughout this course, so you should make every effort to understand not only what, but why you are doing what you are doing at each step.

To be able to write, compile and execute Java applications, you need a Java Environment. The Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as

the Application Programming Interface (API).  
  


**Note**: Since JRE version 1.2, Sun's JVM implementation has included a just-in-time (JIT) compiler instead of an interpreter.

1. **How to Create, Compile and Execute a simple Java program**

**Step 1 : Creation or modification of a Java program** :-

Write or modify the Java program using any text editor or IDE. Save the program on secondary storage device with a **.java** extension. A java source file can contain more than one class; one of which must be declared as **public**. The name of a java source file must be the same as the name of the public class it contains.

**Step 2 : Compiling a Java Program**  
Compile the program using the **javac** compiler. By a command such as:

**Javac WelcomeJavaPrograms.java**

This can be done at the command prompt or through an IDE. After executing the **javac** command, if there are no syntax errors (i.e., grammatical errors) in your program you get a **.class** file which contains an intermediate machine language code called the java bytecode. If a source file has more than one class, each class is compiled into a separate class file.

**Step 3a : Running (Excecuting) a Java Program**  
  
To execute **.class** files they must be translated to the native machine language; this is done by invoking the Java Virtual Machine (JVM) by using the java launcher **java**.exe . The command is similar to:

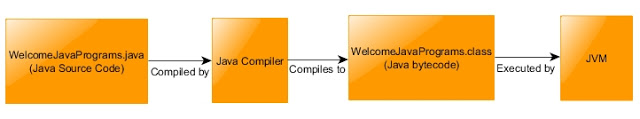
**java -cp D:\ MyFolder WelcomeJavaPrograms**

This can be done at the command prompt or using an IDE.  
  
**Step 3b : Program loading into memory by JVM:-**

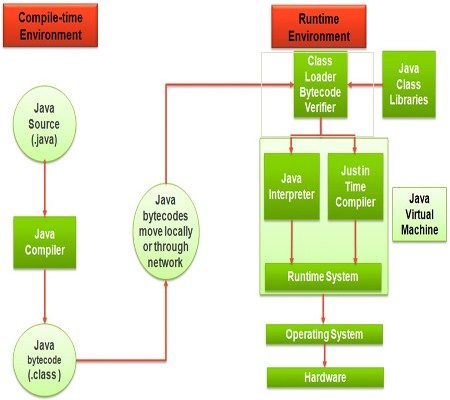
JVM loads the **.class** file to primary memory before its execution. The JVM does this by using a class loader. All the **.class** files required by our program for the execution are loaded by the class loader into the memory just before the execution.

**Step 3c: Bytecode Verification by JVM :-**

In order to maintain security of the program JVM has bytecode verifier. After the classes are loaded in to memory, the bytecode verifier verifies the bytecode of the loaded class or classes in order to maintain security. It checks whether the bytecodes are valid. Thus, it prevents our computer from malicious viruses and worms.

[](http://2.bp.blogspot.com/-ooDB7OdHVqs/T2yqXUWiOHI/AAAAAAAAADE/YzngzhvTkGw/s1600/6272244.jpg)

**Step 3d : Execution of Java program : -**  
Whatever actions we have written in our Java program, JVM executes them by interpreting or JIT-compiling the bytecode.



1. **Introduction to the Eclipse IDE**

See Eclipse Tutorial 01: Setting the Eclipse Workspace and Creating Projects

See Eclipse Tutorial 02: Preparing Eclipse

* Perspectives
* Minimizing, Maximizing and Displaying Windows
* Changing colours and fonts
* Displaying and Hiding line numbers
* Encoding text-files with UTF-8 and displaying Arabic text

**4. The elements of a simple Java program**

**4.1. General Structure**

public class ClassName{

public static void main(String[] args){

statement1;

statement2;

// . . .

statementN;

}

}

**Note: Java is a free form, case-sensitive language.**

**4.2 Comments**

**Comments** are English sentences inserted in a program to explain its purpose; they are ignored by the compiler.

We use comments to explain the purpose of a class, a method or a variable.

* **Single-Line comments** begin with two forward slashes. The compiler ignores everything from the two forward slashes to the end of the line.

**//**

* **Multi-Line comments** start with a forward slash followed by an asterisk, /\*, and are terminated by an asterisk followed by a forward slash, \*/. The compiler ignores everything between the /\* and the \*/. This type of comment can span multiple lines.

**/\***

**\*/**

**Note**: You cannot nest a comment /\* …. \*/ inside another comment /\* …. \*/

**4.3 Literal strings**

A literal string is enclosed in double quotes. Example:

"Welcome to ICS 102"

"السَّلامُ عَلَيكُمْ وَ رَحْمَةُ اللهِ وَ بَرَكَاتُه يَا الشَّبَاب"

**4.4 The newline and tab characters**

If the new line character ‘\n’ appears anywhere in a string literal, it causes an output statement to

generate a new line. The tab character ‘\t’ causes an output statement to generate horizontal spaces.

**4.5 Output Statements**

* In programming, a statement is an instruction to a computer to do something.
* A statement forms part of the sequence of program execution.
* **In Java, every simple statement is terminated with a semicolon.** Example:

The **System.out.println** statement outputs to the monitor and then generates a new line.

The **System.out.print** statement outputs to the monitor without generating a new line.

System.out.println(**"**ICS 102 Lab02**"**);

* Multiple simple statements can be written on a single line. Example:

System.out.println(**"**ICS 102 Lab02**"**); System.out.print(**"**Java**"**);

**4.6 Indentation**

**Indentation** is the placement of text farther to the right, or left, to separate it from surrounding text. The purpose of indentation is to convey the program's structure.

# Example:

**/\* A simple Java program \*/**

**public class HelloJava**

**{**

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

**System.out.println("Welcome to ICS 102");**

**System.out.println("Java is an Object-Oriented Language");**

**}**

**}**

**Note: In this course, you are required to use indentation whenever you write a program. Don’t write code similar to the one below:**

**/\* A simple Java program \*/**

**public class HelloJava**

**{**

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

**System.out.println("Welcome to ICS 102");**

**System.out.println("Java is an Object-Oriented Language");**

**}**

**}**

**Lab Tasks**

**Task 1:** Using the Java API Documentation document find and write down the headers of the methods:

* sqrt, pow, and max of the **Math** class
* nextInt , nextDouble, nextLine of the **Scanner** class

**Note:** If a method has more than one header, choose any one.

An example of a method header is: **static double abs(double a)**

**Task 2:** Using **Notepad** or any text-editor, write the following Java program, save it as **HelloJava.java**, and then compile and execute it from the command-prompt.

**public class HelloJava**

**{**

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

**System.out.println("Welcome to ICS 102");**

**System.out.println("Java is an Object-Oriented Language");**

**}**

**}**

**Task 3:** Using Eclipse, create a Java project called Project01, then create a Java class Welcome as the source code of the project. Finally, compile and execute your project:

**public class Welcome**

**{**

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

**System.out.println("Welcome to the Eclipse IDE");**

**}**

**}**

**Task 4:** Design then implement a Java program that displays the numbers **1** to **4** on the same line, with each pair of adjacent numbers separated by one space, i.e., the output of the program must be:

**1 2 3 4**

Write the program using **each** of the following techniques:

1. Use only one **System.out.println** statement
2. Use four **System.out.print** statements

**Task 5:** Design then implement a Java program to draw a square-like shape that has 4 sides:

\*\*\*\*\*\*

\* \*

\* \*

\* \*

\* \*

\*\*\*\*\*\*

**Task 6:** Read the document **Types of Java Program Errors.docx**. Copy and paste the program below in **Eclipse IDE**. Save the program as **Errors.java** . Determine, classify and correct all errors in the program by compiling and executing the program:

**public Errors{**

**public static void MAIN(string[] args);{**

**int num1, num2;**

**num3 = 25;**

**num2 = 16**

**system.out.println(num1 + " + " + num2 + " = " + (num1 - num2));**

**}**