Chapter1: Object Oriented Programming (OOP)

- Introduction and Concepts

The Fetch and Execute Cycle: Machine Language

A computer is a complex system consisting of many different components. But the brain of the computer is a single component that does the actual computing. This is the ***Central Processing Unit***, or ***CPU***. The job of the CPU is to execute **programs**.

A ***program*** is simply a list of instructions (Code) written by high level programming language. A computer is built to carry out instructions that are written in a very simple type of language called **machine language**. The computer can directly execute a program only if the program first translated into machine language.

Retrieve and Write data in the Main memory RAM

When the CPU executes a program, that program (**active process**) is stored in the computer's ***main memory*** (also called the RAM or Random Access Memory). In addition to the program, memory can also hold data that is being used or processed by the program. Main memory consists of a sequence of ***locations***. These locations are numbered, and the sequence number of a location is called its ***address***. An **address** provides a way of picking out one particular piece of information stored in that address. When the CPU needs to access the program instruction or data in a particular location, it sends the address of that information as a signal to the memory; the memory responds by sending back the value contained in the specified location. The CPU can also store information in memory by specifying the information to be stored and the address of the location where it is to be stored.

On the level of machine language, the operation of the CPU is very complicated in detail. The CPU executes a program that is stored as a sequence of machine language instructions in main memory. It does this by repeatedly reading, or ***fetching***, an instruction from memory and then carrying out, or ***executing***, that instruction. This process fetches an instruction, execute it, fetch another instruction, execute it, and so on forever is called the ***fetch-and-execute cycle***.

Registers

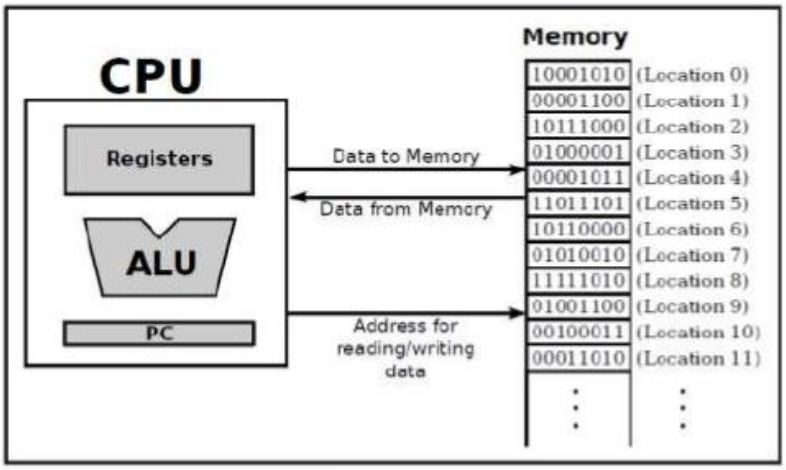
A CPU contains an ***Arithmetic Logic Unit***, or ***ALU***, which is the part of the processor that carries out operations such as addition and subtraction. It also holds a small number of registers, which are small memory units capable of holding a single number. A typical CPU might have 16 or 32 **"general purpose registers"**, which hold data values that are immediately accessible for processing, and many machine language instructions refer to these registers. For example, there might be an instruction that takes two numbers from two specified registers, adds those numbers (using the ALU), and stores the result back into a register. And there might be instructions for copying a data value from main memory into a register, or from a register into main memory.

The CPU also includes **"special purpose registers"**. The most important of these is the ***program counter***, or PC. The CPU uses the PC to keep track of where it is in the program it is executing. The PC simply stores the memory address of the next instruction that the CPU should execute. At the beginning of each fetch-and-execute cycle, the CPU checks the PC to see which instruction it should fetch. During the course of the fetch-and-execute cycle, the number in the PC is updated to indicate the instruction that is to be executed in the next cycle.

Machine language

Machine language instructions are expressed as binary numbers. A binary number is made up of just two possible digits, zero and one. Each zero or one is called a ***bit***. So, a machine language instruction is just a sequence of zeros and ones. Each particular sequence encodes some particular instruction. The data that the computer manipulates is also encoded as binary numbers.

In modern computers, each memory location holds a ***byte***, which is a sequence of eight bits. A machine language instruction or a piece of data generally consists of several bytes, stored in consecutive memory locations. For example, when a CPU reads an instruction from memory, it might actually read four or eight bytes from four or eight memory locations; the memory address of the instruction is the address of the first of those bytes.



Threads and Processes

All modern computers use multitasking to perform several tasks at once. Since the CPU is so fast, it can quickly switch its attention from one task to another, devoting a fraction of a second to each task in turn. This is called timesharing.

For example, the user might be typing a paper while a clock is continuously displaying the time and a file is being downloaded over the network. Each of the individual tasks that the CPU is working on is **called a thread. (Or a process)**. Many CPUs can literally execute more than one thread simultaneously such CPUs contain multiple "cores," each of which can run a thread. Many CPUs can literally execute more than one thread simultaneously such CPUs contain multiple "**cores**," each of which can run a thread.

A Short Java History

In 1991, Sun Microsystems funded the research project called "Green" to design a programming language to be used in intelligent consumer electronic devices, like televisions, and washing machines. This new language was originally termed Oak (referring to the tree that was outside the main developer's, James Gosling's, window), but was quickly renamed Java.

In 1994, the first Java-enabled web browser HotJava was developed. A year later, Netscape incorporated Java support into its web browser. Other companies quickly followed and Java's popularity rapidly rose. Sun released Java 1.0 to the public in 1995. In 2007, Sun made Java's core code available as open source under the terms of the

GNU General Public License (GPL). In 2009, Sun was acquired by Oracle, which is currently continuing the development of Java.

Features of Java

Platform independent and portable: Java programs can be executed in a networked environment with different hardware platforms and architectures. This also makes Java applications extremely portable, effectively realizing the "write once, run everywhere" philosophy.

Object-oriented: Java implements the object-oriented programming paradigm by grouping data and operations into classes and/or objects.

Secure: Java has many facilities to guarantee security in a networked environment. It imposes various types of access restrictions to (networked) resources and carefully supervises memory allocation. It allows code to be downloaded over a network and executed safely in the confined spaces of memory.

Multi-threaded: Java delivers the power of advanced multi-threaded capabilities to the developer in an environment without complexity. More specifically, Java code can be run concurrently as multiple threads in a process, in order to improve its execution performance.

Dynamic: Java allows code to be added to libraries dynamically and then can determine which code should run at execution time. It also foresees a strict separation between interface and implementation.

Java Development Kit (JDK)

JDK stands for **Java Development Kit** it is the subset of SDK which stands for Software Development Kit. JDK contains two units the **development tools** and **java runtime environment (JRE)**. The **development tools** what are used by **developer** to write, compile, test, debug and trouble shoot the (code java) programs and the **java runtime environment JRE** facilitates the execution of java programs. So, we can conclude that JDK=JRE+ Development tools.

1. The development tools

Java provides vary development tools that support the development of large-scale Java systems. The development tools are present in a unit as part of **Java Development Kit** (JDK).

Tools	Description
Applet viewer	Enables to run java applets.
Java	Java interpreter which runs both applet and application by interpreting byte code.
Javac	The java compiler which translates java source code to byte code.
Javadoc	Create HTML format documentation.
Javah	Produces header files for use with native methods.

Java Runtime Environment (JRE):

It stands for Java Runtime environment. It contains the JVM, the library files and the other supporting files. To run a java program, the JRE must be installed in the system. JRE provides runtime environment and libraries to java program (**Bytecode / .class**) which is getting interpreted by JVM. JRE is what making it possible to compile once and execute anywhere. So, we can simply say JRE=JVM+ some packages.

JRE consisting of the following:

1. **Java Virtual Machine (JVM):**

It stands for Java Virtual Machine. When we compile the java file, we get a .class (not an .exe).This file contains java byte code which is interpreted by JVM to produce an output. It is responsible for loading, verifying and executing the code. We say that JVM is platform dependent because it is responsible to convert the bytecodes into the machine language for the specific computer/machine.

Various JVM implementations have been provided for various operating system environments. The most popular JVM is **HotSpot** produced by Oracle. It is available for Windows, Linux, Solaris, and Mac OS X. A key component of the JVM is the **interpreter** responsible for interpreting the **bytecode instructions**.

The JVM typically also includes facilities for **multithreading** and **synchronization**, whereby a Java program can be executed in one or more **parallel execution paths** (**threads**) scheduled on one or more CPUs, hereby significantly accelerating its execution time.

2. **Runtime libraries:**

it is a collection of class libraries used to execute java programs.

3. **User interface toolkits:**

Abstract Window Toolkit (AWT) and swing support various input methods to interact with the application program.

4. **Deployment tools:**

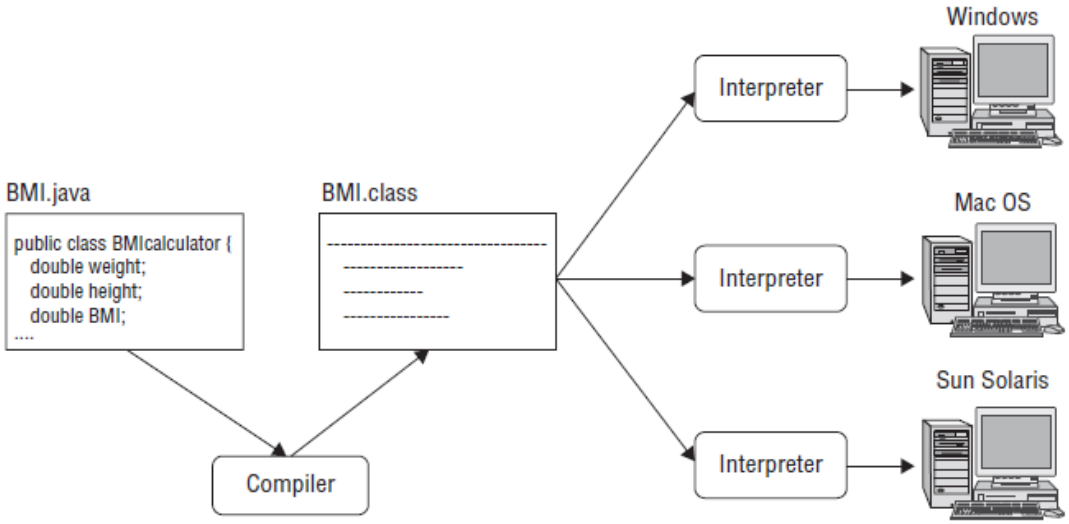
Java Runtime Environment consisting the following tools.

- **Java plug-in:** This plug-in enables the execution of a java applet on the browser.
- **Java web start:** This tool enables remote deployment of an application.

Bytecode (machine-readable byte code**)**

Bytecode is a result of compiling java code which called a machine language of the JVM or (a virtual machine language). This byte code is also called an intermediate

code that can be executed on any system that has a JVM. In order to develop **a cross-platform** solution that would not require developing several compilers to compile **Java source code** to **machine code** for every possible **platform (Windows, Linux, Mac OS, Sun Solaris)**; Java introduced a hybrid approach to run programs by combining both compiler and interpreter technology.



Java Compiler

First, every Java source program (BMI.java) is **compiled** into an intermediate language called ***bytecode*** (BMI.class), which is **platform independent**. During this compilation step, errors in the code can be reported. Java bytecode is not native machine code, so it cannot be run as such on a host computer.

Javas' platform-specific interpreter

In Java, **interpreters** have been developed for various platforms. All of them are implementations of the Java virtual machine (JVM). The **bytecode** can then be considered as machine code for the JVM.

The **bytecode** will be parsed by **a platform-specific interpreter** in order to run it on a particular architecture, such as on Windows, Linux, Mac OS, Sun Solaris, and so on. Irrespective of which platform you are on, the **bytecode** is the exact same.

Resume

So once the java program (Java source code) is compiled into **bytecode** by java (javac) compiler. This bytecode format is same for every platform (Windows/Linux/Solaris etc.). The compiled file typically has .class extension. The .class file can be share across different computers and then it is interpreted and executed by Java Virtual Machine.

Java class (compiled/bytecode) files are not directly executed on processor. JVM takes care of hardware specifics and platform. So, we don't need to worry about recompiling code again and again for each targeted platform and hardware.

The trio- JVM, JRE and JDK are platform dependent (because of the OS dependence) but Java is platform independent (the pair -JVM and bytecode make Java portable). We must remember the simple fact: Any machine language is dependent on the OS of the machine. So, if we have dependency on the machine specific OS, we are not platform independent. Java is platform independent because once the source code is compiled into standard bytecodes, those bytecodes are platform independent. Because of this facility Sun Microsystem is created the slogan WORA (Write Once Run Anywhere) for Java.