JAVA PROGRAMMING

Introduction:

Java programming language was originally developed by Sun Microsystems, which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems.s Java platform (Java 1.0 [J2SE]). Java is guaranteed to be **Write Once, Run Anywhere.**

History of Java:

James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random words.

Features of Java:

- **Object Oriented :** In java everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform independent:** Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP java would be easy to master.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architectural-neutral:** Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence Java runtime system.
- **Portable:** being architectural neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler and Java is written in ANSI C with a clean portability boundary which is a POSIX subset.

- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multi-threaded:** With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
- **High Performance:** With the use of Just-In-Time compilers Java enables high performance.
- **Distributed:** Java is designed for the distributed environment of the internet.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Java Basic Syntax:

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other methods. Class, object, methods and instant variables mean are the basics of Java.

- **Object -** Objects have states and behaviors. Example: A dog has states like color, name, breed as well as behaviors like wagging, barking, eating. An object is an instance of a class.
- **Class -** A class can be defined as a template that, describe the behaviors/states that object of its type support.
- **Methods -** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instant Variables -** Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

First Java Program that prints ***Hello World***

```
public class MyFirstJavaProgram
{
   /* This is my first java program.
    * This will print 'Hello World' as the output
    */
    public static void main(String []args)
  {
       System.out.println("Hello World");   // prints Hello World
    }
}
```

Let us look at how to save the file, compile and run the Java program.

1. Open notepad and add the code as above.
2. Save the file as : MyFirstJavaProgram.java.
3. Open a command prompt window and go o the directory where you saved the class. Assume its C:\.
4. Type ' javac MyFirstJavaProgram.java ' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line.(Assumption : The path variable is set).
5. Now type ' java MyFirstJavaProgram ' to run your program.
6. The output ' Hello World ' is printed on the window.

```
C : > javac MyFirstJavaProgram.java
C : > java MyFirstJavaProgram
Hello World
```

Java Identifiers:

All Java components require names. Names used for classes, variables and methods are called identifiers. In java there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (-).
- After the first character identifiers can have any combination of characters.
- A keyword cannot be used as an identifier.
- Most importantly identifiers are case sensitive.

Java Variables:

There are different types of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non static variables)

Java Keywords:

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

abstract	Assert	boolean	break	short	this
byte	Case	catch	char	switch	try
class	Const	continue	default	throws	super
do	Double	else	enum	volatile	throw
extends	Final	finally	float	static	void
for	Goto	if	implements	synchronized	while
import	Instanceof	int	interface	transient	strictfp
long	Native	new	package		
private	Protected	public	return		

Java Basic Data Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. There are two data types available in Java:

1. Primitive Data Types : byte, short, int, long. Float, double, boolean, char

2. Reference/Object Data Types : Class objects, array variables

Java Literals:

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable. For example:

```
byte a = 68;
char a = 'A'
```

Java Objects and Classes:

Java is an Object Oriented Language. As a language that has the Object Oriented feature Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

A sample of a class is given below:

```
public class Student{
   String Name;
   int age;
   int mark[]=new int[3];

   void getmark(){
   }
```

```
  void grade(){
  }

  void display(){
  }
}
```

A Student has states-name, age, marks as well as behaviors -getmark, grade, display. An object is an instance of a class. A class can be defined as a template that describe the behaviors/states that object of its type support.

A class can contain any of the following variable types.

- **Local variables:** variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

Creating an Object:

The **new** operator dynamically allocates memory for an object and returns a reference to it. This reference is more or less, the address in memory of the object allocated by **new**.

Class_Name Object_Name=new Class_Name();

Example : Student s1 = new Student();

Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

```
/* create an object */
ObjectReference = new Constructor();

/* call the variable */
ObjectReference.variableName;

/* call a class method */
ObjectReference.MethodName();
```

Constructors:

A constructor initializes an object immediately upon creation. Every class has a constructor. If we do not explicitly write a constructor for a class the java compiler builds a default constructor for that class. Each time a new object is created at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
class Student{
  public Student(){
  }

  public Student(String name){
    // This constructor has one parameter, name.
  }
}
```

Java Inheritance:

In java, classes can be derived from classes. Basically if you need to create a new class and here is already a class that has some of the code you require, then it is possible to derive your new class from the already existing code.

This concept allows you to reuse the fields and methods of the existing class without having to rewrite the code in a new class. In this scenario the existing class is called the super class and the derived class is called the subclass.

For example a car class can inherit some properties from a General vehicle class. Here we find that the base class is the vehicle class and the subclass is the more specific car class. A subclass must use the **extends** clause to derive from a super class which must be written in the header of the subclass definition. The subclass inherits members of the superclass and hence promotes code reuse. The subclass itself can add its own new behavior and properties.

```
class Subclassname extends Superclassname
{
   // body of the class
}
```

Java Polymorphism:

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object. A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Java Packages:

A Package can be defined as a grouping of related types(classes, interfaces, enumerations and annotations) providing access protection and name space management. Some of the existing packages in Java are:

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces etc. It is a good practice to group related classes implemented by you so that a programmers can easily determine that the classes, interfaces, enumerations, annotations are related. Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classed.

When creating a package, you should choose a name for the package and put a **package** statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file. If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

```
/* File name : Mypackage.java */
package pack1;

interface interface1{
   public void func1();
   public void func2();
}
```

Java Interfaces:

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class. The **interface** keyword is used to declare an interface.

```
/* File name : InterfaceName.java */
import java.lang.*;

public interface InterfaceName
{
   //Any number of final, static fields
   //Any number of abstract method declarations\
}
```

A class uses the **implements** keyword to implement an interface. An interface can extend another interface, similarly to the way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Common Exceptions:

In java it is possible to define two categories of Exceptions and Errors.

- **JVM Exceptions:** These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples : NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException,
- **Programmatic Exceptions:** These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException.

Try and Catch:

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
```

```
{
  //Protected code
}catch(ExceptionName e1)
{
  //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter. A try block can be followed by multiple catch blocks.

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature. You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

A method can declare that it **throws** more than one exception, in which case the exceptions are declared in a list separated by commas.
The finally Keyword:
The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred. Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of the catch blocks.

Declaring you own Exception:
You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.

- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```
class MyException extends Exception
{
//coding to create our own Exception
}
```

Java Multithreading:

Java provides built-in support for *multithreaded programming*. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes. A *process* consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.
Life Cycle of a Thread:

A thread goes through various stages in its life cycle namely a thread is born, started, runs, and then dies.

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities:

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled. Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Creating a Thread:

Java defines two ways of creating a thread.

- implement the Runnable interface.
- extend the Thread class.

Java Applet Basics:

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.

Life Cycle of an Applet:

Five methods used to build an Applet class Framework are:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

The following is a simple applet named HelloWorldApplet.java:

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet
{
   public void paint (Graphics g)
   {
      g.drawString ("Hello World", 25, 50);
   }
}
```

References:

1. Java 2 : The Complete Reference , fifth edition, Herbert Schildt, TMH Publications
2. Programming with Java: A primer, E. Balagurusamy, TMH Publications
3. www.freejavaguide.com