UNIT 1

Java background : History

- Developed by James Gosling and his colleagues at **Sun Microsystems** in **1991.**
- Initially it was called '**oak'**, and renamed as **'Java'** in **1995.**

The **original motivation** for Java

- The need for **platform independent** language that could be embedded in various consumer electronic products like toasters and refrigerators.

One of the first projects developed using Java

- a personal hand-held remote control named Star 7.

At about the same time

- The **World Wide Web** and the **Internet** were gaining
- popularity. Gosling et. al. realized that **Java** could be **used for Internet** programming.

- Internet users had problems of **portability** and **platform independence.**
- Java being secure, portable and platform independent was found to be capable of addressing large scale problems across the Internet.

Lack of OO purity and facilities

- Java's primitive types are not objects.
- Primitive types hold their values in the stack rather than being references to values. This was a conscious decision by Java's designers for performance reasons. Because of this, Java is not considered to be a pure object-oriented programming language.
- However, as of Java 5.0, auto boxing enables programmers to write as if primitive types are their wrapper classes, and freely interchange between them for improved flexibility:

- Java designers decided not to implement certain features present in other OOP language:

  - multiple inheritance
  - operator overloading

What is Java?

- An object-oriented programming language.
- The language itself borrows much syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java is only distantly related to JavaScript, though they have similar names and share a C-like syntax.
- A cross platform language.
- It is used for stand-alone applications, net based programs and to program consumer devices.

  For e.g.: cellular phones, palm pilots.

Types of Java programs

1. Applets
2. Command line applications
3. GUI applications
4. Servelts
5. Packages
6. Database connectivity

Java's Magic : The Bytecode And JVM

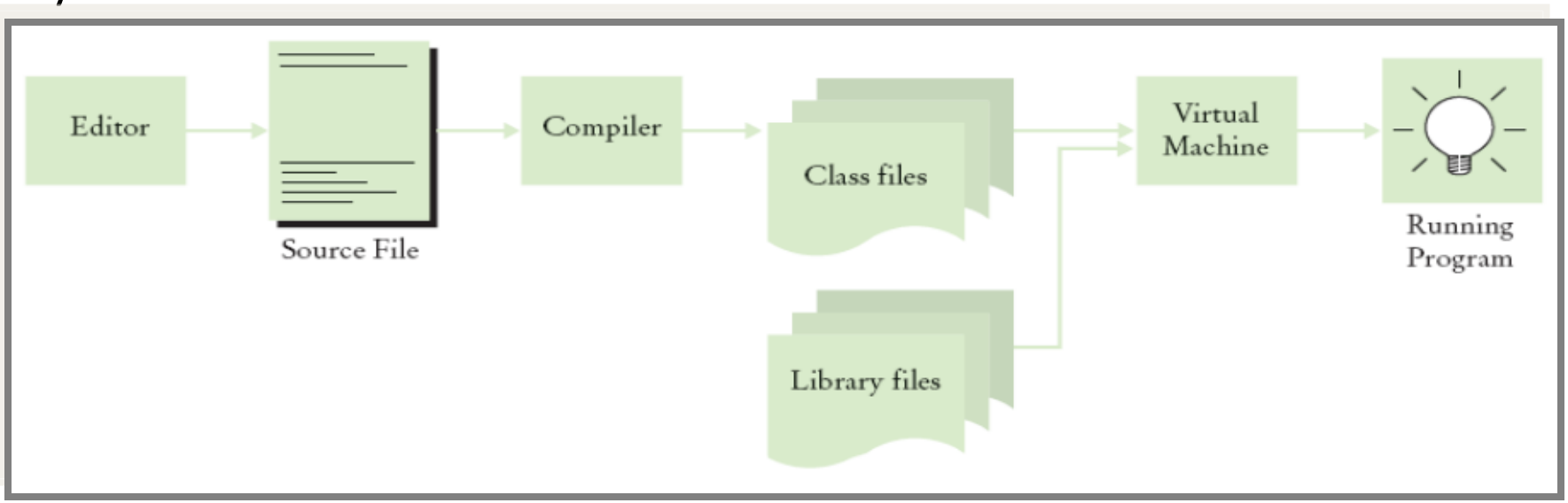
The Bytecode:

- The Bytecode allows both: Security And Portability.
- The problem is that the output of java program is not an executable code but it's a Bytecode.

"Bytecode is a highly optimized set of instruction designed to be executed by the Java runtime system, which is called Java Virtual Machine."

JVM : Java Virtual Machine

- JVM is interpreter for the Bytecode file.
- All language compiler translate source code into m/c level code for a specific computer.
- The Java compiler produce an intermediate code know as Bytecode for a m/c that is not in readable format.
- JVM is located inside the computer memory and Bytecode is executed by only JVM.



Compilation Process : From source code to running program

There are two Processes:

- Basically the Java program can be run in two process.

1. Process of Compilation

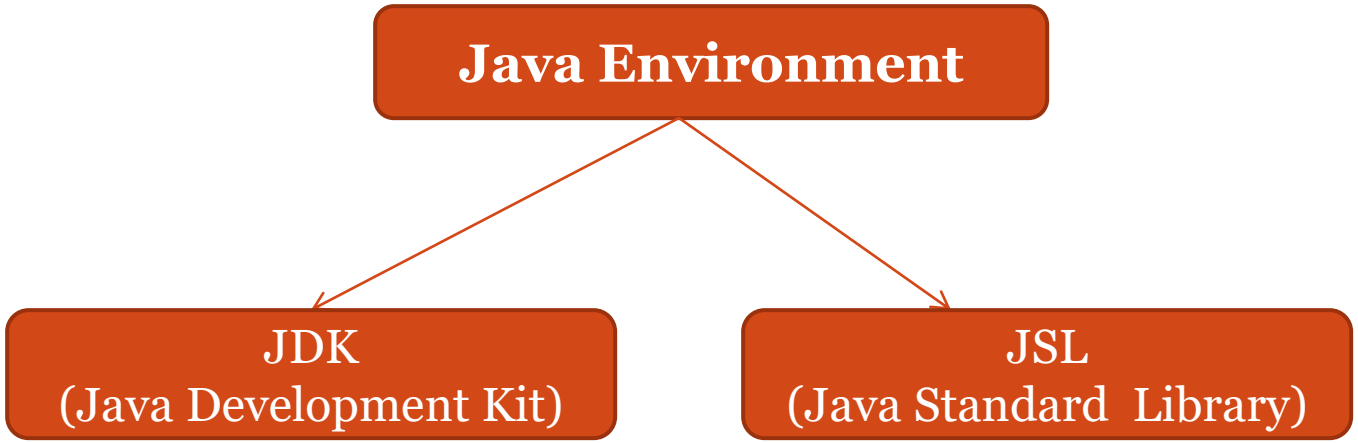
1. Process of converting Bytecode to m/c code

JVM cont…

- JVM also has JIT(Just-In-Time) compiler.
- JIT is a part of JVM.
- JIT execute the selected portions of Bytecode.
  i.e. JIT compiles code as needed during execution.
- Not all sequence of Bytecode are compiled but only those that will be benefit from compilation. The remaining code is simply interpreted.

Java Environment

- Java Environment includes a large number of development tools and hundreds of classes and methods.



JDK (Java Development Kit)

- JDK is a collection of tools that are used for developing and running Java Program.

1. **javac** - compiler used to compile Java source code.

   Syntax : **javac filename.java (**Source files end with extension .java

2. **Java -** interpreter used to execute Java byte codes.

   Syntax: **java filename**

3. **appletviewer** : Used to view and test applets.

   Syntax : appletviewer [options] url

4. **javadoc** : This is the Java documentation tool.

   Generates detailed documentation in HTML form for any .java source code or package

5. **jdb**: Java debugger which help to find errors in program.

Features of Java

- Sun Microsystems officially describe Java with the following features.

✓Simple

✓Robust

✓Compiled & Interpreter

✓Object -Oriented

✓Platform Independent and Portable

✓Multithreading

✓Distributed

✓High Performance

- **Simple:**

  If you are an experienced C++ programmer, moving to Java will be require very little effort as Java inherits the C/C++ syntax and many object-oriented features of C++.

- **Robust :**

  It proved two main things:

  1. Strictly compilation
  2. Memory Management

1. Strictly compilation:

   Strictly compilation means its checks **data types** and **exceptional conditions** and all **runtime errors**.

2. Memory Managements:

   As we all know that memory management is a very tedious and difficult task in traditional programming. And this sometimes get a problem, because programmers will either forgot to free memory that has been previously allocated. Java dynamically eliminates these problem by managing **garbage collection.**

- **Compile and Interpreter:**

  Usually a computer language is either compiled or interpreted. But Java combines both these features.

- **Object-Oriented:**

  Almost everything in Java is Object Oriented.

  All program code and data reside within classes and objects.

- **Platform Independent & Portable:**

  There is no guarantee that if you write a program today, it will run tomorrow – even on the same machine.

  There may be possibility to change OS, processor so your program not work with these new changes.

  To solve this problem Java language use JVM.

  Their goal was “write once: run anywhere, anytime, forever. And it was accomplished.

Most important feature of Java is portability over other languages.

Java ensure portability in two ways:

1. Compiler generates Bytecode file that is translate by JVM and give m/c readable output.
2. Size of data type are m/c dependent.

- **Multithreading:**

  - Multithreading means handle multiple tasks simultaneously.

    i.e. we need have to wait to finish one application before start another.

- **Distributed:**

  - Java is designed as a distributed language for creating application on Internet

  - It handle TCP/IP for Internet programming.

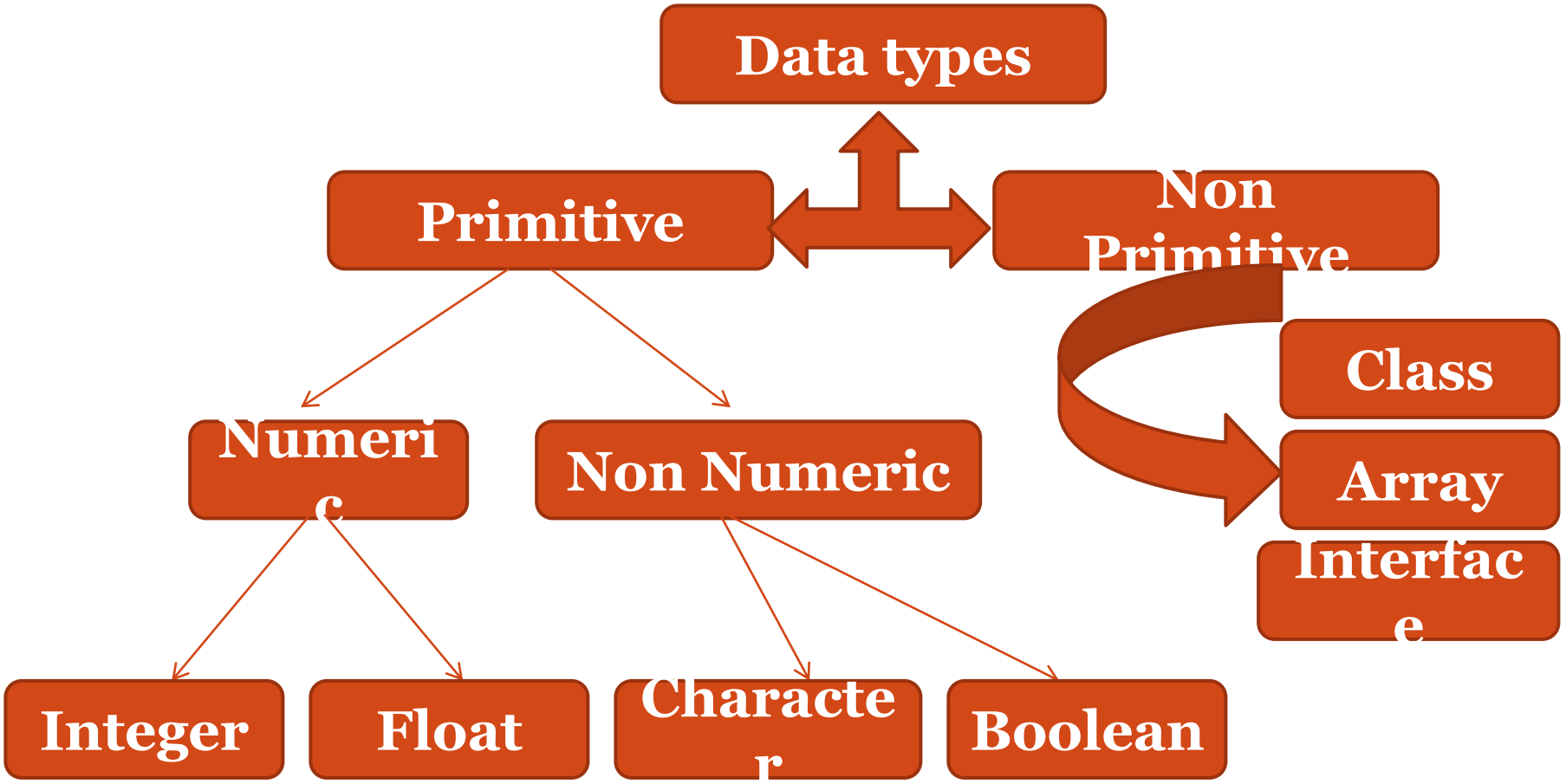
  - Java also support RMI.

- **High Performance:**

  Java performance is impressive for an interpreted language because of intermediate bytecode. It reduce overheads during runtime.

Data types

- Java supports two types of data types.



Type	Bits	Lowest Value	Highest Value
boolean	(n/a)	false	true
char	16	'\u0000' [0]	'\uffff' [2^{16}-1]
byte	8	-128 [-2^{7}]	+127 [2^{7}-1]
short	16	-32,768 [-2^{15}]	+32,767 [2^{15}-1]
int	32	-2,147,483,648 [-2^{31}]	+2,147,483,647 [2^{31}-1]
long	64	-9,223,372,036,854,775,808 [-2^{63}]	+9,223,372,036,854,775,807 [2^{63}-1]
float	32	±1.4012984632481707e-45	±3.4028234663852860e+38
double	64	±4.94065645841246544e-324	±1.79769313486231570e+308

Basic Operators

- Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –
- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

General Structure of Java program

Structure of Java Program		
Document Section	-	Suggestion
Package statement	-	Optional
Import Statement	-	Optional
Interface Statement	-	Optional
Class Definition	-	Necessary
Main Method class	-	Necessary
Main method definition	-	Necessary

Sample Program

```
/*
 * Created on Jul 14, 2005
 *
 * First Java Program
 */
import java.util.*;

/**
 * @author JDS
 */
public class JavaMain {

public static void main(String[] args) {
        // print a message
System.out.println("Welcome to Java!");
   }
}
```

Comments in Java

1. Single Line Comment

```
// insert comments here
```

2. Block Comment

```
/*
 * insert comments here
 */
```

3. Documentation Comment

```
/**
 * insert documentation
 */
```

Object 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 Passing

Objects in Java

- Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
- If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.
- Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.
- So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

- A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

The General Form of a class

- class classname {
  type instance-variable1;
  type instance-variable2;
  .....
  type instance-variableN;
  type methodname1(parameter-list){
  //body of method }
  type methodname2(parameter-list){
  //body of method }
  .....
  type methodnameN(parameter-list){
  //body of method }
  }

Constructors

- 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

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

Creating an Object

- A class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.
- There are three steps when creating an object from a class –
- **Declaration** – A variable declaration with a variable name with an object type.
- **Instantiation** – The 'new' keyword is used to create the object.
- **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example

```
public class Puppy
   {
        public Puppy(String name)
        {System.out.println("Passed Name is :" + name ); }

  public static void main(String []args)
   {
  // Following statement would create an object myPuppy
   Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

Introduction to Packages

- In Java, a package is a combination of classes, interfaces and sub-packages

  e.g.: java.awt package has a sub-package called *event*

- Package act as “container” for classes that keeps the list of classes.

- List of classes name created it your own package will not collide with other class name list stored somewhere.

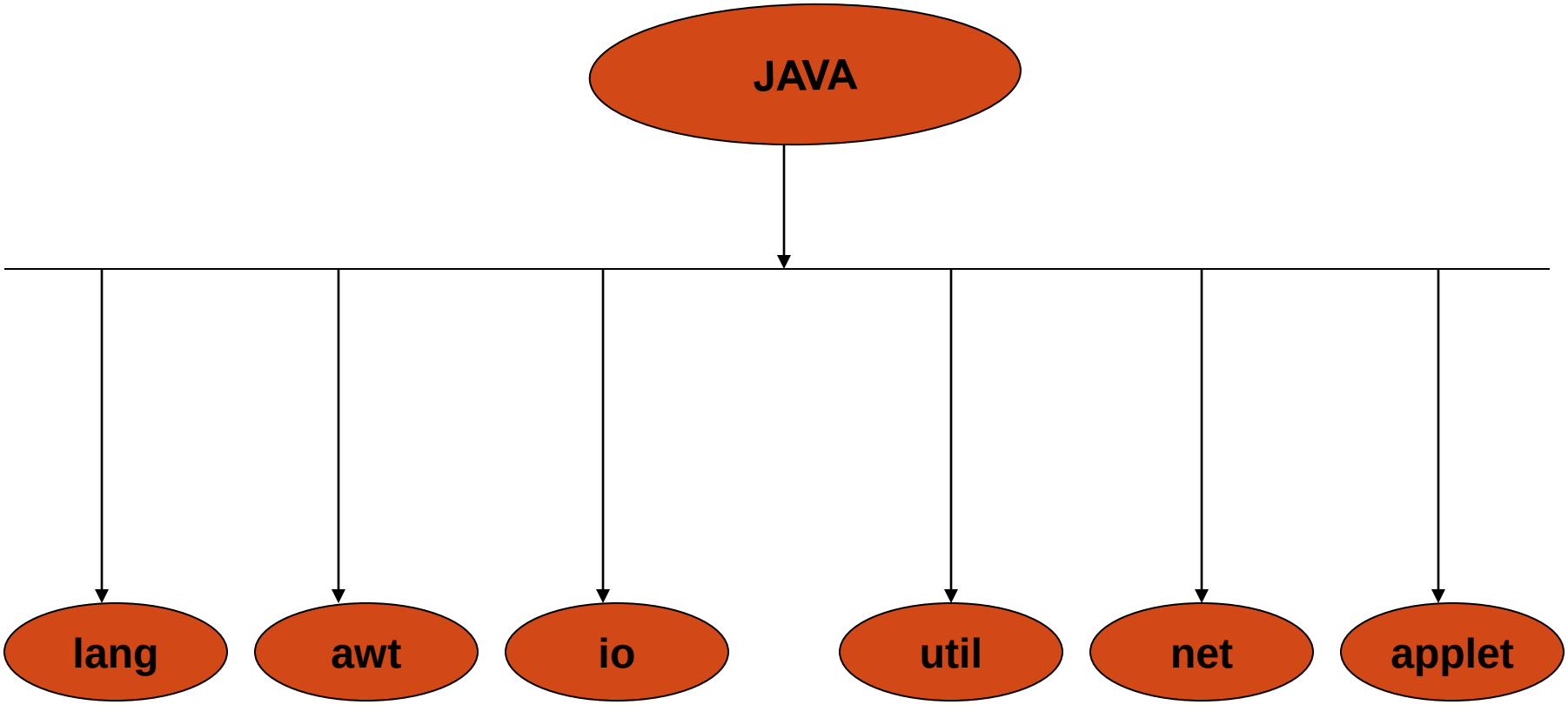
Classification

- A java package is classified into two types.

  Java API Packages

  User define Packages

JAVA API PACKAGES



Packages	Description
java.lang	Language support classes. And imported automatically
java.util	Language utility class such as date.
java.io	Input/output support class.
java.awt	Classes for graphical user interface.
java.net	Classes for networking.
Java.applet	Classes for creating and implementing applets.

Creating a package

- For declare the name of the package use keyword **"package"** then define class name.

  package *pkgname*;

  e.g.

  Public class MyClass { …}

  The listing would be saved as file called MyClass.java and located in a directory named mypackage.

- Java uses file system directories to store packages.

  e.g. .class file of any class must be stored in a **directory that has same name as package name.**

- You can also create hierarchy of package.

  e.g. package pkg1.pkg2.pkg3;

- When a Java program is executed, the JVM searches for the classes used within the program on the file system.

- Uses one of **two** elements to find a class –
  - The package name
  - The directories listed in the CLASSPATH environment variable

- If no CLASSPATH is defined, then JVM looks for the default java\lib directory and the current working directory

Packages and Access control

- There are main three access specifires, public, private and protected.
  - **A public member** of a class can be accessed from anywhere; within the package, outside the package, within a subclass, as well as within a non-subclass
  - A member of a class that is declared **private** can be accessed only within the class but nowhere outside the class.
  - If no access specifier is given, the member would be accessible within any class in the same package but not outside the package
  - If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element **protected.**

Accessing Packages

- fully qualified class name that we want to use and that is done by using dot operator.

  e.g. java.awt.Color;

  Here **awt** is package within the package **java**

- **import Java.awt.*;**

  Import statement must appear at the top of the file and before any class and packages declaration.

  Note that statement must be ends with

  (;)

Packages and .jar files

- Packages are convenient , but large directories are not.
- Java allows you to bundle the entire directories into a single file, called a **jar [Java Archive] file.**
- There are two main uses for JAR files which I shall explain here.
  - ✓ The first use is to compress (make a smaller size) a number of files into one file (archiving).
  - ✓ The second use is to make a *Java executable* JAR file.

Packages and Classpath

- You may have many different packages in your system.
- How does JAVA knows where they are?

- Answer is Classpath

- A classpath is a system environment variable that gives a list of directories and jar files where Java should look up of its classes.

Set Classpath

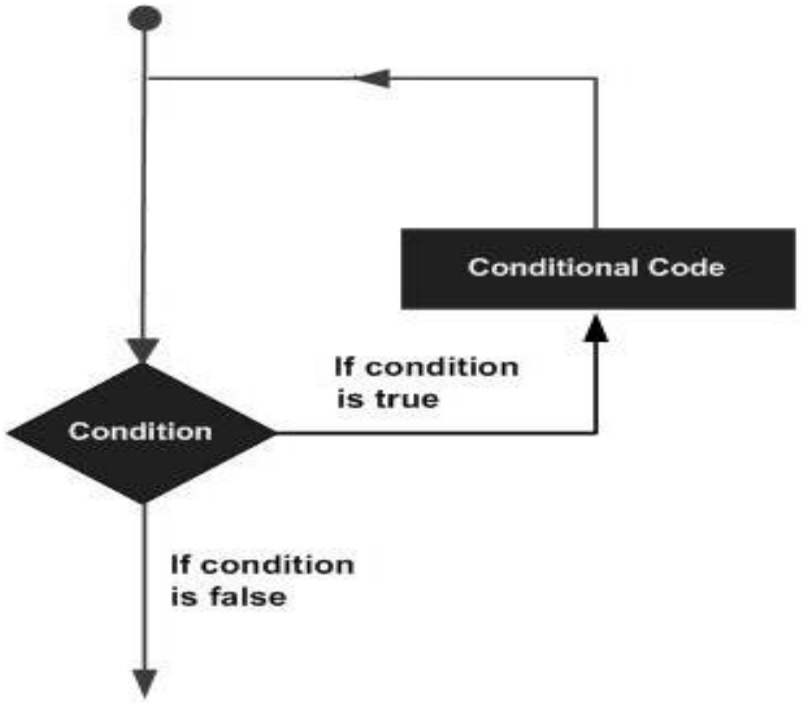
CLASSPATH in Command prompt:

c:\set CLASSPATH = .;[path where your classes/package/jar file located]

- List of directories separated by semicolon.
- "." represent "current directory".
- Jar file must be listed separately.

Loop Control

- A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –



while Loop

- A **while** loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

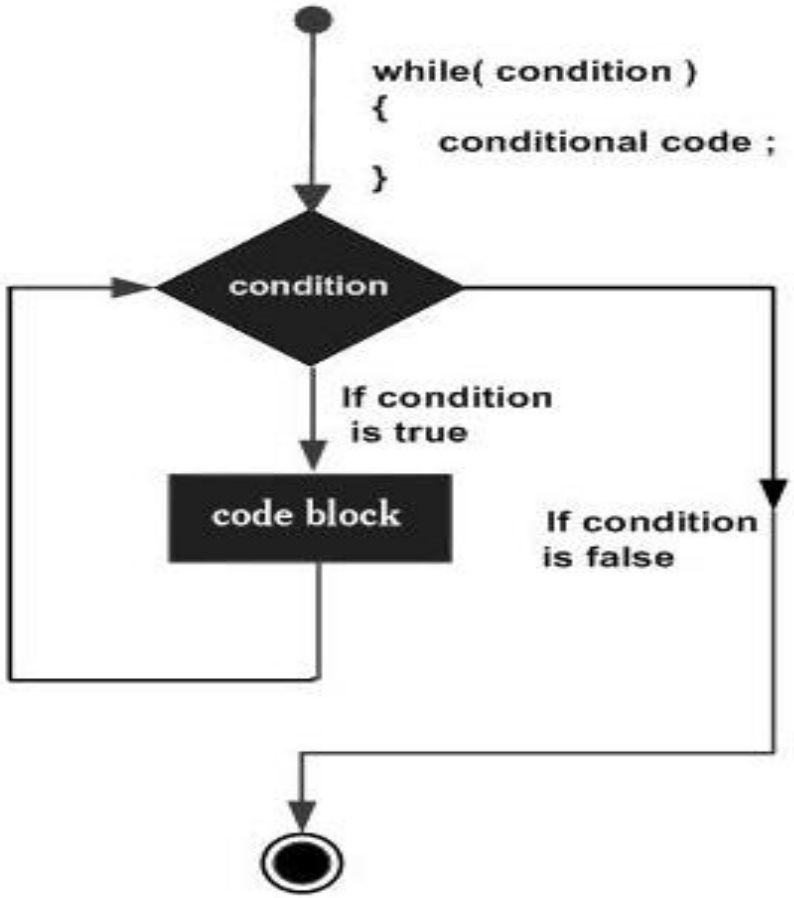
The syntax of a while loop is –

```
while(Boolean_expression)
        { // Statements }
```

Example:

```
while( x < 20 )
  {    System.out.print("value of x : " + x );
       x++;
       System.out.print("\n");
}
```

Flow Diagram



for loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.
- A **for** loop is useful when you know how many times a task is to be repeated.

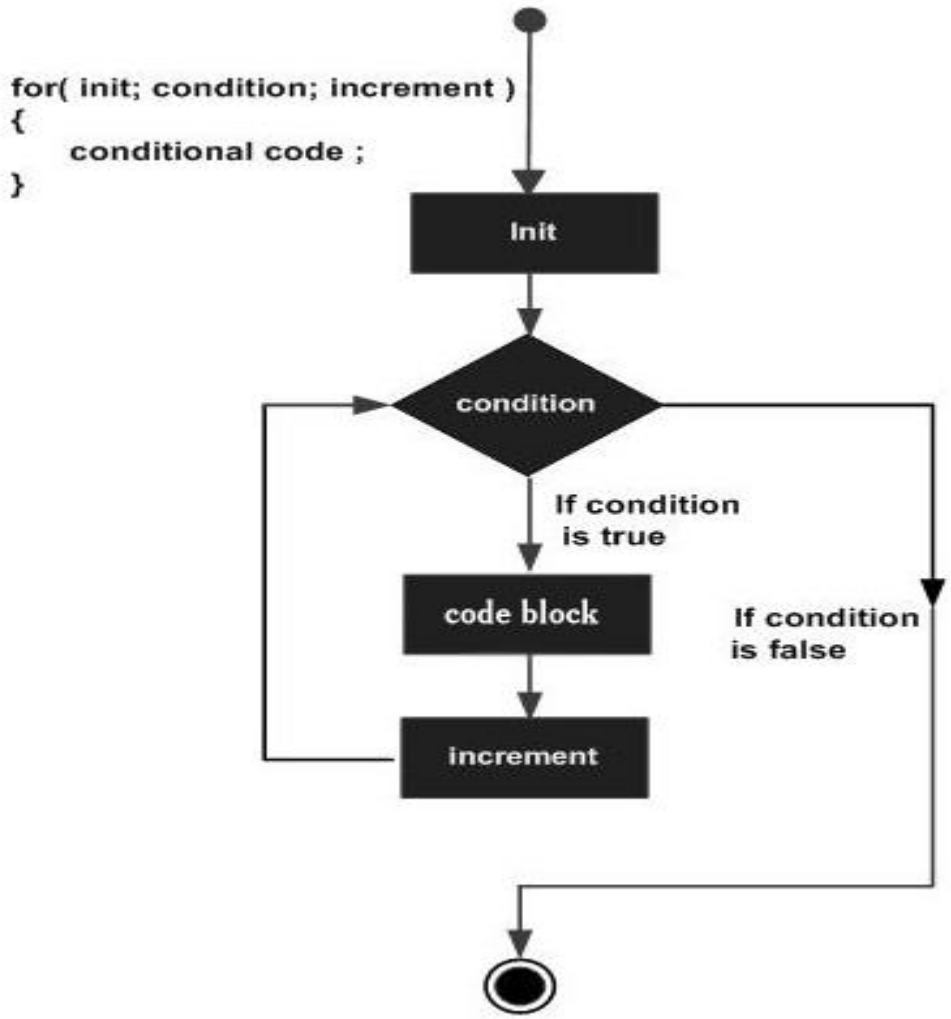
Syntax

```
for(initialization; Boolean_expression; update)
   { // Statements }
```

Example:

```
   for(int x = 10; x < 20; x = x + 1)
         { System.out.print("value of x : " + x );
         System.out.print("\n");
         }
```

Flow Diagram



do while loop

- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

```
do {
        // Statements
        }while(Boolean_expression);
```

Example:

```
do {
        System.out.print("value of x : " + x );
        x++;
        System.out.print("\n");
        }while( x < 20 );
```

Flow Diagram