**UNIT 1: Introduction to java**

1. **Overview of Java:**

GENESIS OF JAVA

* Java is a programming language developed by JAMES GOSLING at SUN MICROSYSTEMS and released in 1995.The language derives much of its syntax from C and C++; but has a similar object model and fewer low level fertility.
* Java application are typically compiled to BYTECODE (class file) that can run on any Java Virtual Machine (JVM).
* Java is a general purpose, class based, object oriented language that is designed to have few implementation dependencies as possible. It is intended to let application developers write once ,run anywhere it is one of the most popular language in use widely used from application software to web applications

**Why use java**

Java technology is an object oriented /platform independent , multithreaded programming environment .It enables:

* To efficiently deploy exciting new application services.
* To securely occur key system resources while protecting your data.
* Allow to do more with your application, from real time capabilities to high performance multitasking virtual machines to broad access to system capability.
* Runs across more than seven billion devices including mobile, desktop, tv etc.

**Java applets and applications**

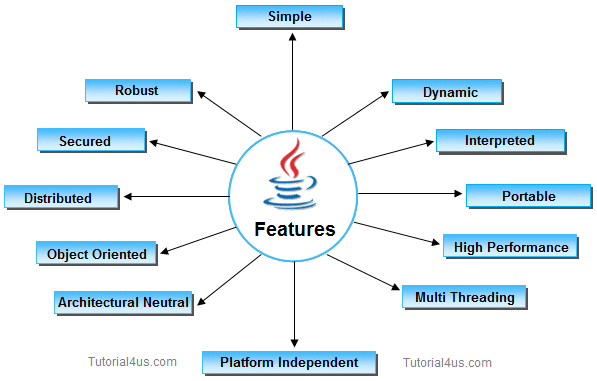
Java is used to create 2 types of programs:

1. Applets
2. Application

* Application is a program that runs on the computer, under its OS. It is more or less like one created using C/C++.
* Applet is an application designed to be transmitted over the internet and executed by a java compatible web browser. An applet is a tiny java program dynamically downloaded across the network like image or video clip.
* The difference between applet and application is that an applet is an intelligent program ,not an animation /media files.

**2.Bytecode:**

* The key that allows java to solve both security and portability problems first describe is that the output of java computer is not executable code/but bytecode. The java source code is converting into virtual code called as **byte code.**
* A bytecode is a set of highly optimized set of instructions which can be executed by JVM. JVM acts as an interpreter for bytecode .This help to solve the major problems associated with downloading programs over the internet .
* Once the run time package exists for a given system, any java program can run on it .Interpretation of bytecode is the easiest way to create truly portable programs .It also helps to make it secure.
* The use of bytecode enable JVM to execute programs much faster.
  1. **Java Features/Buzz words:**

****

* 1. **Simple**

Learning java will be much easier if we already understand the basic concept of object oriented program, then moving to java will need less effort.

**Java is simple because of the following factors:**

* Java is **free from pointer** due to this execution time of application is improve. [whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
* Java have **Rich set of API** (application protocol interface).
* Java have **Garbage Collector** which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
* Java contains user friendly syntax for developing Java applications.

**2. Object Oriented:**

Java was not designed to be source code compatible with any other language . One outcome of this was a clean, usable, pragmatic approach to objects. Objects model in simple/easy to extend.

**3. Robust:**

* To gain reliability, javas restrict you in few key areas, to force you to find your mistakes early in program development. It also frees you from having to worry about the most common causes of programming errors.
* Java is a strictly typed language, it checks user code at compile time and run time.   
  The main reasons for failure are,

1. Memory management mistakes
2. Mishandled exceptional conditions

* Java virtually eliminates these problem by managing memory allocation and deallocation. Exceptions are handled by providing object oriented exception handling .

**4.Portability:**

* Many types of computer and OS are in use throughout the world and many are connected to internet. For programs , to be dynamically downloaded , some means of generating portable executable code is needed.
* If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, Pascal are treated as non-portable language. JAVA is a portable language.



**5.Security:**

Java provides a firewall between a networked applet and your computer. When you use a java compatible web browser, you can safely download java applets without fear of viral infection.

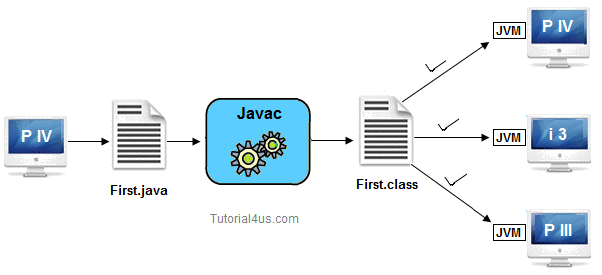
Java is more secured language compare to other language; All java code is covered into **byte code** after compilation which is not readable by human.

**6.Multithread:**

* It allows you to write an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems.
* A flow of control is known as thread. When any Language execute multiple thread at a time that language is known as multithreaded Language. java and .net are multithreaded Language.

**7.Architectural Neutral:**

* The goal of java designers to develop “**write once, run anywhere**, anytime, forever” so that a program can be independent of the architecture of the system in which it is running.
* A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering there architecture and vendor (providers) irrespect to its development and compilation



**8.High Performance:**

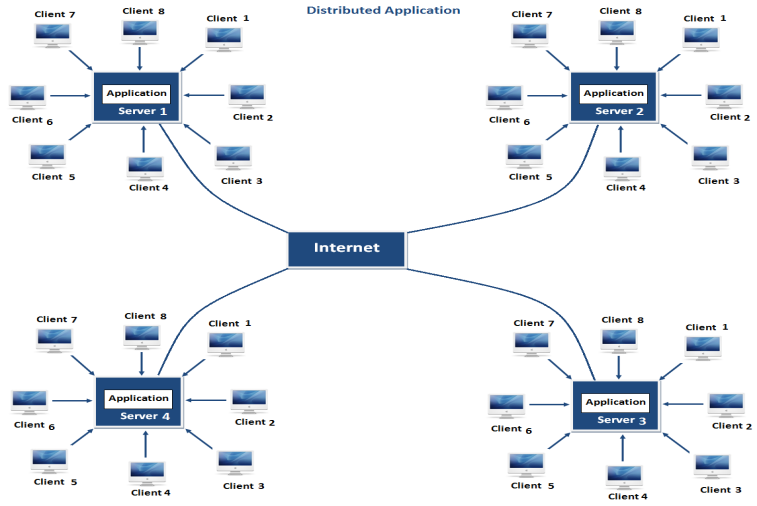
It enables the creation of cross-platform programs by compiling into an intermediate representation called java bytecode .It can be interpreted on any system that provide JVM.

Java have high performance because of following reasons;

* Java **uses Bytecode** which is more faster than ordinary pointer code so Performance of java is high.
* **Garbage collector**, collect the unused memory space and improve the performance of java application.
* Java have **no pointers** so that using java program we can develop an application very easily.
* It **support multithreading**, because of this time consuming process can be reduced to execute the program.

**9.Distributed:**

* Java is designed for distributed environment of internet ,since it handles TCP/IP protocol .The original version of java included features for intra address-space messaging. This allowed objects on two different computers to execute procedures remotely.
* We can create distributed applications in java. RMI and EJB are used for creating distributed applications. In distributed application multiple client system are depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.



**10.Dynamic:**

* Java programs carry with them substantial amount of run time type information that is used to verify and resolve accesses to objects at run time thus making it possible to dynamically link code in a safe and expedient manner
* Java programming support Dynamic memory allocation due to this memory wastage is reduce and improve performance of application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation, In java programming to allocate memory space by dynamically we use an operator called 'new' 'new' operator is known as dynamic memory allocation operator.

**4.Java Environment variable/JAVA runtime Environment(JRE):**

Java environment is a collection of tools and class, methods. The developments tools are part of system called as java development kit(JDK) and classes ,methods are part of the java standard library(JSL) also known as the Application programming interface(API).

1) Java Development kit (JDK).

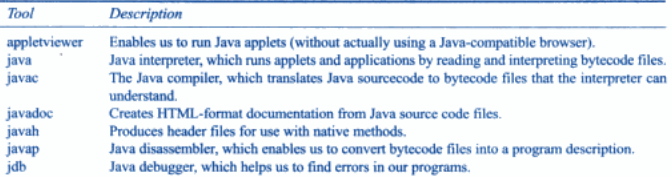
2) Application programming interface (API)/Java Standard library(JSL)

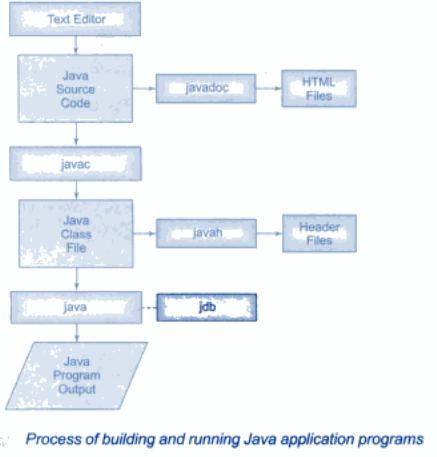
**JAVA DEVELOPMENT KIT – (JDK)**

As the name suggests refers to a collection of tools that are used for developing and running java programs. The jdk consists of a collection of tools as

* appletviewer (to launch applets)
* javac (java compiler)
* java (java interpreter)
* javap (java disassembler)
* javah (for C header files)
* javadoc (for HTML documents)
* jdb (java debugger)

A source program written in java is compiled using “javac” (java compiler) and executed using “java” (java interpreter). The “jdb” (java debugger) is used to locate errors if any in the source file.





**Application programming interface(API):**

API is a collection of classes and methods are grouped into several different packages.

* Language Support package: Is a collection of classes and methods required for implementing basic features of java. EX: **import java.lang.\*;**
* Utilities package: Is a collection of classes to provide utility function such as data,time function Ex: **import java.util.\*;**
* Input/output package: Is a collection of classes required for input and output manipulation EX: **import java.io.\*;**
* Networking package: Is a collection of classes for communicating with other computer via internet. Ex: **import java.net.\*;**
* AWT package: The abstract window tool kit package contains classes that implements platform independent graphical user interface. EX: **import.java.awt.\*;**
* Applet package: This include a set of classes that allows us to create java applets. Ex: **import java.applets.\*;**

**5.JAVA IS INTERPRETED:**

Java as a language initially gained popularity mainly due to its platform independent architecture or portability feature. The reason for java to be portable is that it is interpreted.

**Java source code**

**(.java)**

javac filename.java java filename

**Native Code**

**JIT compiler**

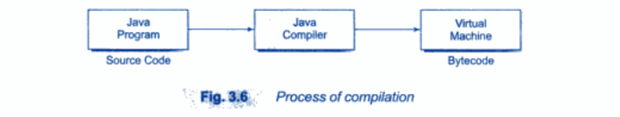
**Bytecode file (.class file)**

Just in time is added in jvm which makes the program execution much faster.

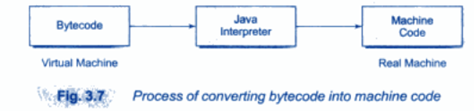
**6.The Java Virtual Machine (JVM):**

Java Virtual Machine (JVM) is the heart of entire Java program execution process.

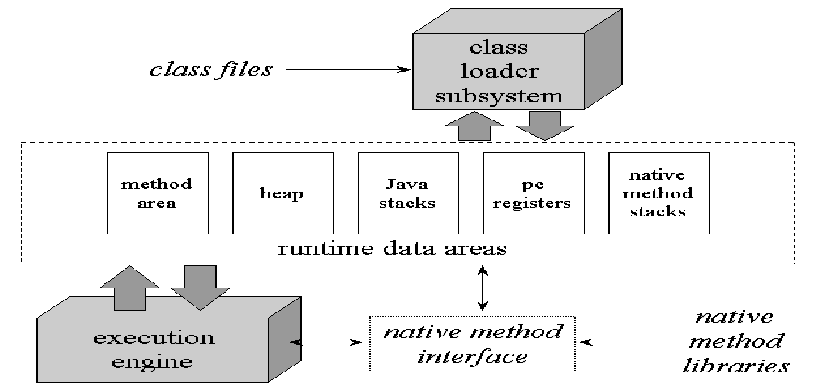
**The process of compiling the java source code into byte code is called as virtual machine code.**



The virtual machine code is not machine specific. The machine specific code also known as machine code is generated by the java interpreter by acting as an intermediary between the virtual machine and the real machine.



First of all, the .java program is converted into a .class file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This .class file is given to the JVM. Following figure shows the architecture of Java Virtual Machine.

****

**Figure:** The internal architecture of the Java virtual machine.

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:

* First of all, it loads the .class file into memory.
* Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.
* If the byte instructions are proper, then it allocates necessary memory to execute the program. This memory is divided into 5 parts, called run time data areas, which contain the data and results while running the program. These areas are as follows:
* **Method area:** Method area is the memory block, which stores the class code, code of the variables and code of the methods in the Java program. (Method means functions written in a class).
* **Heap:** This is the area where objects are created. Whenever JVM loads a class, method and heap areas are immediately created in it.
* **Java Stacks:** Method code is stored on Method area. But while running a method, it needs some more memory to store the data and results. This memory is allotted on Java Stacks. So, Java Stacks are memory area where Java methods are executed. While executing methods, a separate frame will be created in the Java Stack, where the method is executed. JVM uses a separate thread (or process) to execute each method.
* **PC (Program Counter) registers:** These are the registers (memory areas), which contain memory address of the instructions of the methods. If there are 3 methods, 3 PC registers will be used to track the instruction of the methods.
* **Native Method Stacks:** Java methods are executed on Java Stacks. Similarly, native methods (for example C/C++ functions) are executed on Native method stacks. To execute the native methods, generally native method libraries (for example C/C++ header files) are required. These header files are located and connected to JVM by a program, called **Native method interface**.
* **Execution Engine**  contains interpreter and JIT compiler which translates the byte code instructions into machine language which are executed by the microprocessor. Hot spot (loops/iterations) is the area in .class file i.e. executed by JIT compiler. JVM will identify the Hot spots in the .class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter.

**7.OOP Features:**

**Abstraction**

* It is the property of hiding the information and presenting only the necessary. For a real time example, a car is not considered as a set of thousands of individual parts but the object as a whole. Abstraction can be managed through the use of hierarchical classification, i.e., complex systems can be broken down into more manageable pieces.
* The data from a traditional process –oriented program can be transformed by abstraction into it component objects. A sequence of process steps can become a collection of steps between these objects .Each of these objects describe their unique properties.

**Encapsulation**

* It is a mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. It can be considered as a protective wrapper which helps in preventing the code and data from being accessed by other code defined outside the wrapper. Access is possible through the well defined interface .
* In java basis of encapsulation is class which defines the structure and behavior (data and code) that will be shared by a set of objects. Each object of a given class contains the structure and behavior defined by the class. Therefore objects are also called as *instance of class.*
* Data defined by the class is called member/instance variable. Code that operates on the data is refined to as (member) methods.

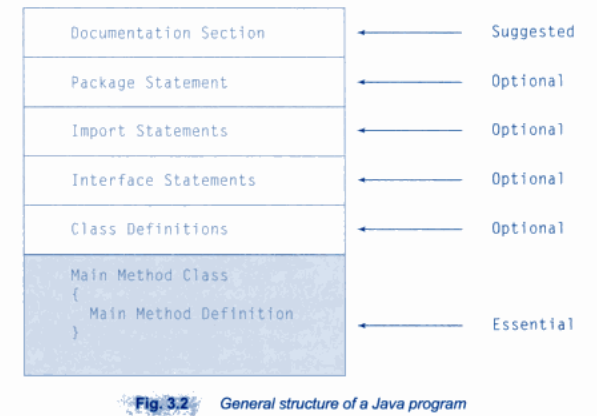
**Inheritance**

* It is the process by which one object acquires the properties of another object. This is important since it supports the concept of hierarchical classification. Without the use of hierarchies, each object would need to define all of its characters explicitly.
* By using inheritance ,an object need only define those qualities that make it unique within its class .It can inherit its general attributes from it parent .thus ,it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.
* Polymorphism It is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.

**polymorphism**

* The concept of polymorphism is “one interface, multiple methods”, i.e. , it is possible to design a generic interface to a group of related activities.
* This helps reduce the complexity by allowing the same interface to be used to specify a general class of actions.
* It is the compiler’s job to select the specific action as it applies to each situation. Programmers do not need to make this selection manually.

**10.Simple Java program/General structure of java:-**

****

* **Document section:** Gives the description about what type of program written, operation carried out, copy right of program within the comment line. Ex: /\* stmt \*/
* **Package statement:** Declare a name of package or folder name inform to the compiler that the class belongs to this package. EX: package folder\_name;
* **Import statement:** This statement inform to the interpreter about the corresponding class or methods to be loaded in the program. Ex: import java.util.\*; or import foldername.\*;
* **Interface statement:** This section contains the interface or function declaration that belongs to class.
* **Main line:** the program execution always begin with the main methods which takes arguments as a string type.

class myjava   
{  
public static void main (String arg[])  
{  
System.out.println (“My 1st program!");  
}  
}

* The first line of code is:

**public static void main(String arg[])**

1. This line begins the **main()** method. This is the line at which the program will begin executing. Now let us briefly examine the other parts of the above statement.
2. The **‘public’** keyword is an *access specifier*, that allows the programmer to control the accessibility of the class members. When specified as public, it can be accessed by code outside the class in which it is declared. **’private’** is its complement which does not allow the access to the code outside the class in which it is declared. Here main() must be declared as public, as it must be called by code outside of its class when the program is started.
3. The **‘static’** keyword allows the **main()** to be called by the java interpreter before any objects are made. No need to instantiate a particular instance of a class.
4. The keyword **‘void’** tells the compiler that main() does not have any return value
5. Java is case sensitive. **’Main’** and **‘main’** are different. Java compiler will compile classes that do not have a main() method. But java interpreter has no way to run these classes. So if misspelled or case changed, interpreter would report an error as it could not find the main() method.
6. Main() has only one parameter. **String args[]** declares a string array parameter, of name args. They store character strings. Here, args receives any command-line arguments present when the program is executed. The above program made no use of this information!

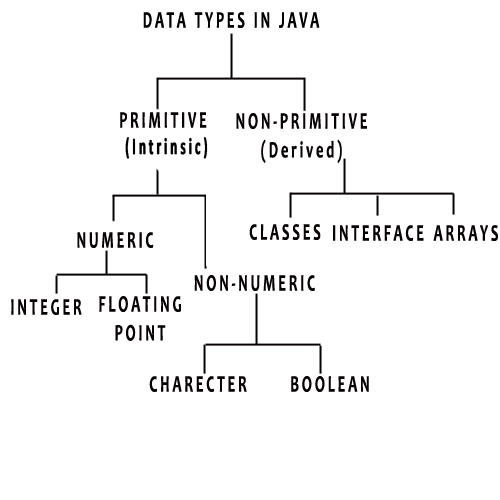
**System.out.println("My 1st program!");**

is shown. This outputs the string my 1st program followed by a new line on the screen. It is accomplished by the built-in **println()** method.

*println()* displays any string that is passed to it. The line begins with System.out, where System is a predefined class that provides access to the system and out is the output stream that is connected to the console. We’ll deal about this in detail in the later parts.

**11.DATATYPES:**

Every variable in your java has a data type, which specifies the size, and type of values. Every variable, every expression has a type and the types are strictly defined. The java compiler checks all expressions for type compatibility.

**Datatypes**   


Java includes mainly 4 categories of data types:

* [Integers](http://www.codeteasers.com/java/data/dat.php#int)
* [Characters](http://www.codeteasers.com/java/data/dat.php#char)
* [Floating Point numbers](http://www.codeteasers.com/java/data/dat.php#float)
* [Boolean](http://www.codeteasers.com/java/data/dat.php#bool)

**Integer Types:**

* This can hold whole numbers such as 123,-105, 5399 etc...The size depends on the integer data type we choose. As said earlier java supports four elemental data types
* Java does not support the concept of unsigned types. Therefore all java values are signed i.e. they can be positive or negative

**Floating Point Types:**

Used to hold numbers containing fractional parts such as 40.5, 23.7 etc. also called as floating point constants. There are two kinds of floating point storage in java. They are float and double.

**Character type:**

* The keyword is char. It is not as same as that of C/C++. In C/C++ char is an integer type i.e. 8-bit wide, whereas in java it is 16-bit, as java uses Unicode to represent characters.
* Unicode defines a fully international character set that can represent all characters found in all human languages. The range is 0-65,536.

**Boolean type:**

Used when we want to test a condition. It can have only two values i.e. True or False. This is the type returned by all relational operators, also required by the conditional expressions.

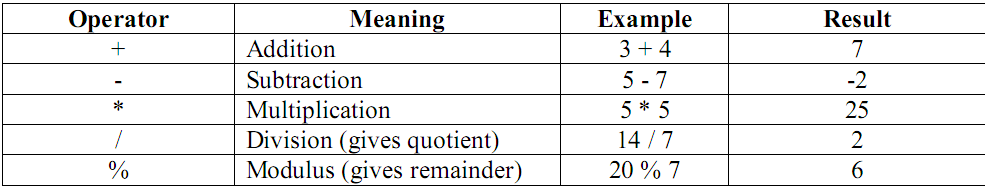
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group | Data types | Size | Range | Default value |
| Integer | Byte | 1 byte | -27 to 2 7-1(signed)  Unsigned does not support | 0 |
| Short | 2 byte | -215 to 2 15-1 | 0 |
| Int | 4 byte | -231 to 2 31-1 | 0 |
| Long | 8 byte | -263 to 2 63-1 | 0 |
| Float | Float | 4 byte | 3.4e-038 to 3.4 e+038 | 0.0 |
| Double | 8 byte | 1.7e-308 to 1.7e+308 | 0.0 |
| Boolean | Boolean | 1 bit | True or false | False |
| Character | char | 2 byte | A single character | null |

**12.OPERATORS:**

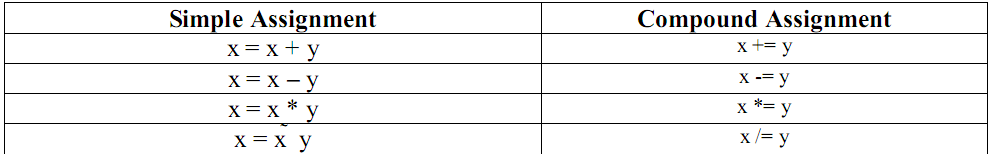
Java provides a rich operator environment. Most of its operators can be divided into the following four groups: arithmetic, bitwise, relational, and logical. Java also defines some additional operators that handle certain special situations.

**Operators:** An operator is a symbol that performs an operation. An operator acts on variables called operands.

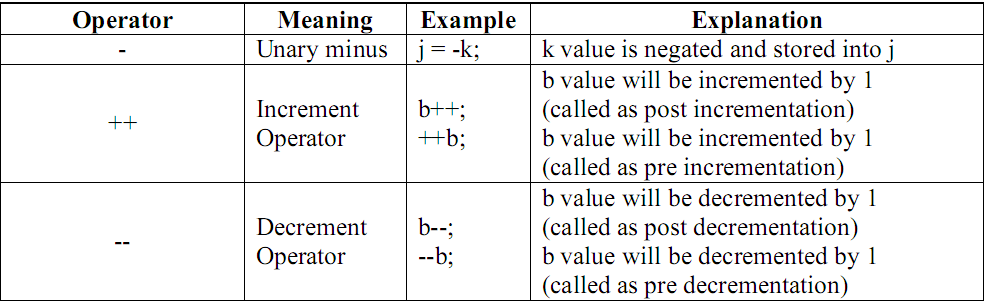
* **Arithmetic operators:** These operators are used to perform fundamental operations like addition, subtraction, multiplication etc.



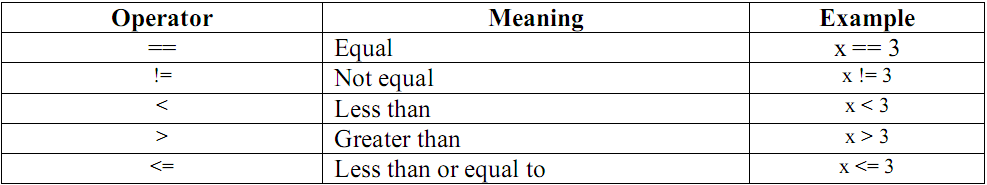
* **Assignment operator:** This operator (=) is used to store some value into a variable.



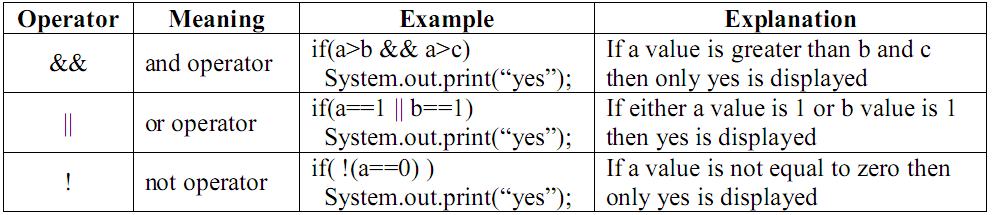
* **Unary operators:** As the name indicates unary operator’s act only on one operand.



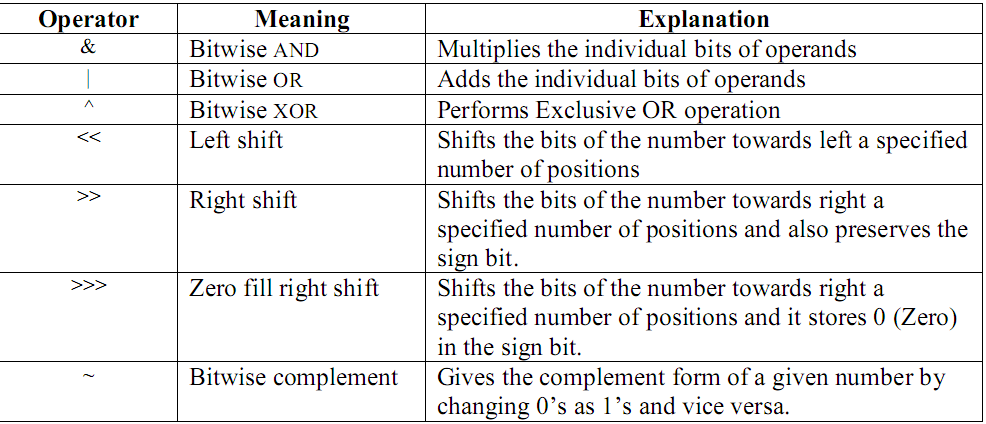
* **Relational operators:** These operators are used for comparison purpose.



* **Logical operators:** Logical operators are used to construct compound conditions. A compound condition is a combination of several simple conditions.



* **Bitwise operators:** These operators act on individual bits (0 and 1) of the operands. They act only on integer data types, i.e. byte, short, long and int.



* **Ternary Operator or Conditional Operator (? :):** This operator is called ternary because it acts on 3 variables.

The syntax for this operator is:

Variable = Expression1? Expression2: Expression3;

First Expression1 is evaluated. If it is true, then Expression2 value is stored into variable otherwise Expression3 value is stored into the variable.

e.g.: max = (a>b) ? a: b;

|  |  |
| --- | --- |
| **Operator Precedence** | |
| **Operators** | **Precedence** |
| Postfix | expr++ expr-- |
| Unary | ++expr --expr +expr -expr ~ ! |
| Multiplicative | \* / % |
| Additive | + - |
| Shift | << >> >>> |
| Relational | < > <= >= instanceof |
| Equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| Ternary | ? : |
| Assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**13.TYPE CONVERSION AND CASTING:**

Process of converting one data type to another data type is called type casting. There are two types of casting

**1) Automatic/implicit/widening/promotion type casting**

**2) Explicit/narrowing type casting**

**Java’s Automatic Conversions/implicit/widening/promotion type casting:**

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

• **The two types are compatible.**

**• The destination type is larger than the source type.**

When these two conditions are met, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required.

When these two conditions are met, a widening conversion takes place.

Eg:   
int a;  
byte b=10;  
a=b;

**2) Explicit/narrowing type casting**

* This kind of conversion is called narrowing conversion as here the value is made explicitly narrower to fit into target.
* The process of converting larger data type to smaller data type , here data may loss.
* For this, you must use cast.  
  (target type) value, target type= the desired type to convert the specified value to.

Eg:

int a;  
byte b;  
b=(byte )a;

**15.ARRAYS:**

Arrays: An array represents a group of elements of same data type. Arrays are generally categorized into two types:

* Single Dimensional arrays (or 1 Dimensional arrays)
* Multi-Dimensional arrays (or 2 Dimensional arrays, 3 Dimensional arrays, …)

**Single Dimensional Arrays:** A one-dimensional array is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is

Syntax:-

type var\_name [];

Eg: int a[];

a=new int[10];

**or**

int a[]=new int[10];

The steps involved are:-

* Declare a variable of desired array type.  
  int a[];
* Allocate memory that will hold the array using new and assign it to the array var.

a=new int[10];  
 **Also it can be declared as:** int a[]=new int[10];  
 **Also array can be initialized when they are declared.**  
 int a[]={1,2,3,4,5,6,7,8,9,10};  
The array will automatically be created large enough to hold the number of elements you specify. No need of new key word.

* Java strictly checks to be sure that all array indices are in the correct range.
* **Two Dimensional Arrays (2d array):** A two dimensional array represents several rows and columns of data. To represent a two dimensional array, we should use two pairs of square braces [ ] [ ] after the array name. For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array.

They are actually array of arrays.  
Syntax:-  
int a[][]=new int[3][3];  
This allocates 3x3 array and assigns to ‘a’

**//sorting of names in ascending order**

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class Sorting

{

public static void main(String[] args)

{

String k[]={a,c,b};

String temp=new String();

String names[]=new String[k+1];

for(int i=0;i<k;i++)

{

names[i]=args[i];

}

for(int i=0;i<k;i++)

for(int j=i+1;j<k;j++)

{

if(names[i].compareTo(names[j])<0)

{

temp=names[i];

names[i]=names[j];

names[j]=temp;

}

}

System.out.println("Sorted order is");

for(int i=k-1;i>=0;i--)

{

System.out.println(names[i]);

}

}

}

**Output: a,b,c**

**16.LITERALS:**

A literal is a value that can be written directly into a Java program. The compiler can calculate the value it represents because most [primitive data types](http://java.about.com/od/understandingdatatypes/a/primitivetypes.htm) have their own literal pattern.

Literal patterns:

* **whole numbers** are int values (e.g., 123).
* **whole numbers followed by an "L" (or "l")** are long values (e.g., 6720000000L). Note, long variables can be assigned a value using an int literal - a long literal is only needed when it is greater than the value an int data type can hold.
* **decimal numbers** are double values (e.g., 1.2) Note, you can also follow a decimal number with a "D" or "d" to explicitly define it as a double value.
* **decimal numbers followed by an "F" or "f"** are float values (e.g., 0.2F).
* **truth values** are boolean values (i.e., true, false).
* **single characters in single quotes** are character values (e.g., 'a').
* **characters in double quotes** are string values (e.g., "abc").
* **whole numbers preceded by "0x"** are [hexadecimal](http://java.about.com/od/h/g/hexadecimal.htm) numbers (e.g., 0xFF).
* **whole numbers preceded by "0"** are [octal](http://java.about.com/od/o/g/octal.htm) numbers (e.g., 0647).

**17.CONTROL STATEMENTS:**

Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution. In Java control statements are categorized into selection control statements, iteration control statements and jump control statements.

* **Java’s Selection Statements:**

These statements allow us to control the flow of program execution based on condition.

Types of control statements are:

1)Simple if Statement

2)if else

3)else if

4)switch

**1) Simple if statements:**

If condition is true then part will print.

Syntex:

**If(condition)**

**Print statement**

**Ex:**

class A

{ public static void main(String args[])

{ int a=10,b=20;

if (a<b)

System.out.print (a);

}

}

2**.if else :** if statement performs a task depending on whether a condition is true or false.

**Syntax:** if (condition)

statement1;

else

statement2;

ex:

class A

{ public static void main(String args[])

{ int a=10,b=20;

if (a<b)

**System.out.print (a);**

else

**System.out.print (b);**

}

}

**3.else if:** This statement perform a task depending on whether a condition is true or false.

Syntex: if(condition )

Statement

Else if(condition)

Statement

Else

Statement

Ex:

class A

{ public static void main(String args[])

{ int a=10,b=20,c=30;

if (a<b)

**System.out.print (a);**

Else if(b<c)

**System.out.print (b);**

**Else**

**System.out.print (c);**

}

}

**4.Switch Statement:**

When there are several options and we have to choose only one option from the available ones, we can use switch statement.

**Syntax:** switch (expression)

{ case value1: //statement sequence

break;

case value2: //statement sequence

break;

case valueN: //statement sequence

break;

default: //default statement sequence

}

**2) Java’s Iteration Statements**: Java’s iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops

Types of looping statements are:

1)while

2)do while

3)for

4)for each(Enhanced for loop)

1) **while Loop:** while loop repeats a group of statements as long as condition is true. Once

the condition is false, the loop is terminated. In while loop, the condition is tested first; if

it is true, then only the statements are executed. while loop is called as entry control loop.

**Syntax:** while (condition)

{

statements;

}

Ex:

class A

{ public static void main(String args[])

{ int i=1;

**while (i <= 20)**

**{ System.out.print (i );**

**i++;**

**}**

} }

2) **do…while Loop**: do…while loop repeats a group of statements as long as condition is

true. In do...while loop, the statements are executed first and then the condition is tested.

do…while loop is also called as exit control loop.

**Syntax:** do

{

statements;

} while (condition);

Ex:

class A

{ public static void main(String args[])

{ int i=1;

**do**

**{ System.out.print (i + “\t”);**

**i++;**

**} while (i <= 20);**

}

}

3) **for Loop**: The for loop is also same as do…while or while loop, but it is more compact syntactically. The for loop executes a group of statements as long as a condition is true.

**Syntax:** for (expression1; expression2; expression3)

{ statements;

}

Here, expression1 is used to initialize the variables, expression2 is used for condition

checking and expression3 is used for increment or decrement variable value.

Ex:

class A

{

public static void main(String args[])

{ int i;

**for (i=1; i<=20; i++)**

**System.out.print (i );**

}

}

4)**For Each**: loop will work for array elements, this loop is much faster than the for loop statements.

Systex:

for(type var\_name: Target Array\_var\_name)

{

Ststements;

}

Ex:

Class A{

Public static void main(String ar[]) {

Int a[]={1,2,3,4}

**For(int b:a)**

**{**

**System.out.println(b);**

**}**

}

}

**3) Java’s Jump Statements**: These statements transfer control to another part of the program.

Java supports three jump statements: **break, continue and return.**

1. **break:**

* break can be used inside a loop to come out of it.
* break can be used inside the switch block to come out of the switch block.
* break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.

Ex:

class A

{

public static void main(String args[])

{ int i;

for (i=1; i<=20; i++)

**If(i==5)break;**

System.out.print (i );

}

} **output: 1 2 3 4**

**2.continue:** This statement is useful to continue the next repetition of a loop/ iteration.

When continue is executed, subsequent statements inside the loop are not executed.

Ex:

class A

{

public static void main(String args[])

{ int i;

for (i=1; i<=10; i++)

**If(i==5)continue;**

System.out.print (i );

}

}

Outrput: 1 2 3 6 7 8 9 10

**3.return statement:**

* return statement is useful to terminate a method and come back to the calling method.
* return statement in main method terminates the application.
* return statement can be used to return some value from a method to a calling method.

Ex:

class A

{

int fun()

{

int a=10,b=20;

**return a+b;**

}

public static void main(String ar[])

{

A a1=new A();

System.out.println(“sum is=”+ a1.Fun());

}

}

**18.Creating and Destroying object:**

**Creating object:**

* In java objects are created using the new keyword
* new operator creates the object of specified class and returns reference to that object of that class.
* Syntex:

Class\_name object\_name=new Class\_name();

Ex:

A a1=new A();

**Destroying objects:**

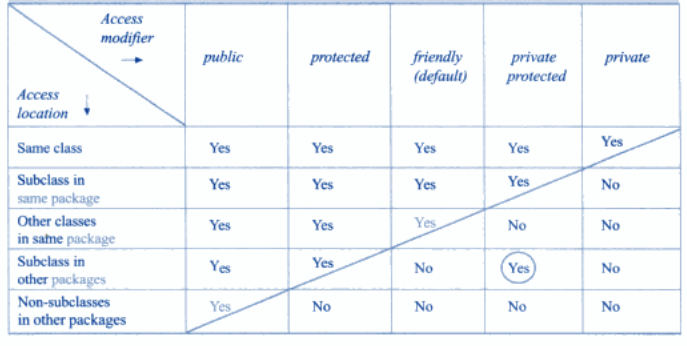
* In java objects are destroyed automatically with the help of garbage collator, if objects are no longer used in the program .
* Object can also destroyed by pointing to NULL

Ex: A a1;

Al=null;

* Objects can also deleted using finalizing method, But problem with finalizing method is that it will not guaranty the object destroyed successful, its time consuming process.

**19.Access Specifiers /visibility control/access control :**

**Note : Refer the example of class notes as program.**

**Private:** If any class or method made private access specifiers it can access in same class. Private access specifeir restricts the inheritance of method and class.

**Public:** If any class or method made public access specifiers it can access throughout the program. It can accessible in same class, sub class and other class in same package and also in other package.

**Protected**: If any class or method made Protected access specifiers It can accessible in same class, sub class and other class in same package and but also in sub class of other package.

**Default/friendly:** If any class or method made **Default/friendly** access specifiers It can accessible in same class, sub class and other class in same package.

**Private protected**: Private protected is applicable to only variables. If any variable made a Private protected access specifier it can access in same and sub class of same package.

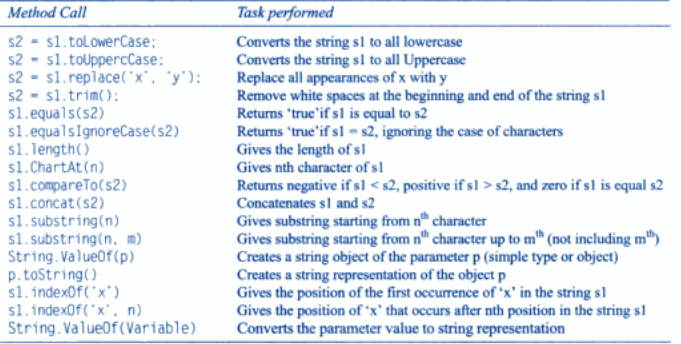
**19. Strings:**

* String is a collection of character enclosed in a double quotes.
* String is not a data type it is class in java, it always work on string class object.

Ex:

String a; Or String a=”hello”

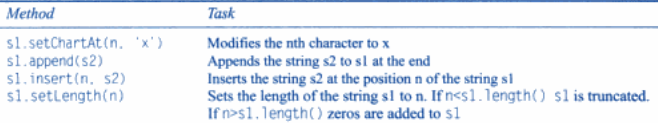
* Different string class function are as follows:



Note: Refer class notes for example as a program.

**StringBuffer class:**

* StringBuffer class creates a string of flexible length that can be modified in terms of both length and contents.
* Where as string class creates a string of fixed length.
* Different StringBuffer Class function are as follows:



Note: Refer class notes as example for program.