JAVA PROGRAMMING

UNIT I

Computer and its Languages

Around the world language is a source of communication among human beings. Similarly, in order to communicate with computers we need programming languages. A programming language is medium of communicating with computers.

Types of Programming Language

Low Level Languages

These are machine codes or close to it. Computer cannot understand instruction given in high level languages or in English. It can only understand and execute instructions given in the form of machine languages i.e. the binary number 0 and 1. There are two types of low level computer language.

Machine Language

The lowest and most elementary language and was the first type of programming language to be developed. Machine language is basically the only language which computer can understand. In fact, a manufacturer designs a computer to obey just one language, its machine code, which is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stand for the absence of an electric pulse and 1 for the presence of an electric pulse.

Advantages

1. It makes fast and efficient use of the computer
2. It requires no translator to translate the code i.e. directly understood by the computer.

Disadvantages

1. All operation codes have to be remembered
2. All memory addresses have to be remembered
3. It is hard to amend or find errors in a program written in the machine language
4. These languages are machine dependent.

Assembly Language

It was developed to overcome some of inconveniences of machine language. This is another low level in which operation codes and operands are given in the form of alphanumeric symbols. These alphanumeric symbols will be known as mnemonic codes and can have maximum up to 5 letter combinations e.g. ADD, SUB, START LABEL etc. because of this feature it is also known as “Symbolic Programming Language”.

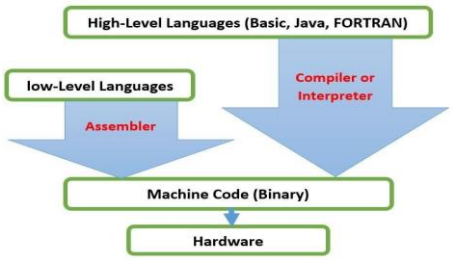
This language is very difficult and needs a lot of practice to master it because very small English support is given. The instructions of the assembly language will be converted to machine codes by language translator called assembler.

Advantages

1. It is easier to understand and use as compared to machine language
2. It is easy to locate and correct errors and modified easily

Disadvantages

1. Like machine language it is also machine dependent
2. Since it is machine dependent, there programmer should have the knowledge of hardware.



High Level Languages

High level computer languages give formats close to English language and the purpose of developing high level languages is to enable people to write programs easily and in their own native language. High-level languages are basically symbolic languages that use keywords and mathematical symbols. Each instruction in the high level language is translated into many machine language instructions by using either compiler or interpreter. Eg. C, C++, Java etc.

History of Java

Java language was developed **James Gosling**, **Mike Sheridan** and **Patrick Naughton** in 1991. Java team members also known as **Green Team.** Firstly java was called **"Greentalk"** by James Gosling and file extension was .gt. After that, it was called **Oak** and was developed as a part of the Green project.

Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc. In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. Java team members initiated this project to develop a language for digital devices such as set-top boxes, televisions etc. But, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

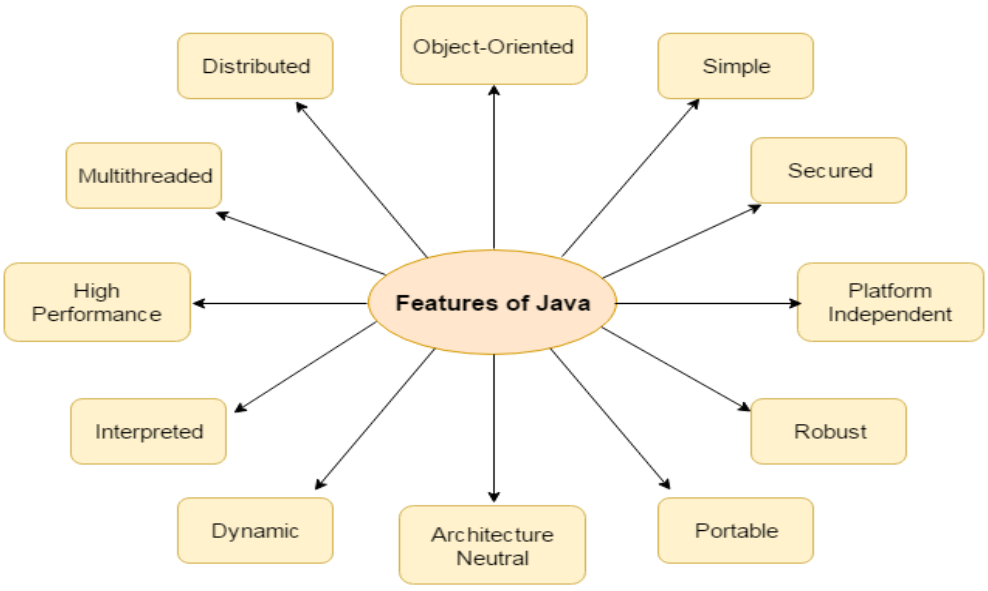
Why had they chosen java name for java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology.

According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java. Java is an island of Indonesia where first coffee was produced (called java coffee). In 1995, Time magazine called **Java one of the Ten Best Products of 1995**. JDK 1.0 released in(January 23, 1996).

Features of Java

The main objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some awesome features which play important role in the popularity of this language. The features of Java are also known as java buzzwords.

A list of most important features of Java language are given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

1. Simple

Java is very easy to learn and its syntax is simple and easy to understand. According to Sun, Java language is a simple programming language because:

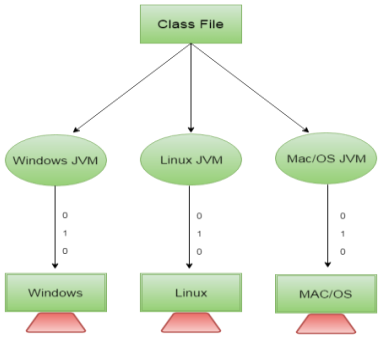
- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many rarely-used features e.g. explicit pointers, operator overloading etc.
- There is no need to remove unreferenced objects because there is Automatic Garbage Collection in java.

2. Object-oriented

Java is object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation



3. Platform Independent

Java is platform independent because it is different from other languages like C, C++etc. which are compiled into platform specific while Java is a write once, run anywhere language.

At the time of compilation, Java compiler produces bytecode. This bytecode runs on any Java Virtual Machine (JVM). JVM is responsible for converting the bytecode to machine code.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

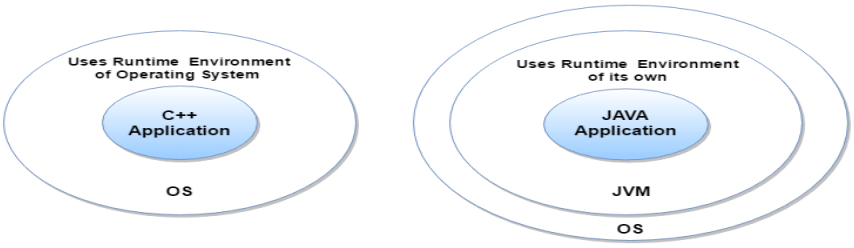
4. Portable

Java is portable because it facilitates you to carry the java bytecode to any platform. It doesn't require any type of implementation. It can be executed by any machine. The size of basic data type in java is compatible for all systems make java program highly portable.

5. Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**



Java does not allow programmers to interact with memory. The java run-time environment uses byte code verification process to ensure that code loaded in network does not violate Java security constraints.

6. Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There are lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There is exception handling and type checking mechanism in java. All these points makes java robust.

7. Architecture-neutral

Java is architecture neutral because there is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

8. Interpreted

Java is called as Interpreted language as it converts bytecode into processor readable binary code. This phase is called interpretation.

9. High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. Java is an interpreted language that is why it is slower than compiled languages e.g. C, C++ etc.

10. Distributed

Java is distributed because it facilitates users to create distributed applications in java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

11. Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

12. Dynamic

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages i.e. C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

Java OOPs Concepts

Object Oriented Programming is a paradigm based on the concept of objects, which contains data in the form of fields and code in the form of methods. **Smalltalk** is considered as the first truly object-oriented programming language. The OOP concepts are,

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any real life entity that has state and behavior is known as an object. It can be physical and logical. Object can be defined as an instance of a class and occupies some space in memory. Objects can communicate without knowing details of each other's data or code, the only necessary thing is that the type of message accepted and type of response returned by the objects.

Class

Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't store any space.

Inheritance

When one object acquires all the properties and behaviors of other object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. In java, we use method overloading and method overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines. A java class is the example of encapsulation.

Capsule

Java Development Kit (JDK)

The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. JDK is comprised of three basic components, as follows:

- Java Virtual Machine (JVM)
- Java Runtime Environment(JRE)
- Java Application Programming Interface (API)

The JDK contains a private Java Virtual Machine (JVM) which is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed.

The JRE (Java Runtime Environment) which is a set of software tools which are used for developing Java applications. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform

Application Programming Interface (API)

An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

In Java, most basic programming tasks are performed by the API's classes and packages, which are helpful in minimizing the number of lines written within pieces of code.

Literals in Java

Java Literals are the values assigned to the **variable**. It is also called a constant. Eg. int x = 100;
Here, 100 is literal. Java has five types of Literals.

1. Integral Literals in Java

a. Decimal Literals (Base 10)

Digits from 0-9 are allowed in this form.

Example: int x = 101;

b. Octal Literals (Base 8)

Digits from 0 – 7 are allowed. It should always have a prefix 0

Example: int x = 0146;

c. Hexa-Decimal Literals (Base 16): Digits 0-9 are allowed and also characters from a-f are allowed in this form. Furthermore, both uppercase and lowercase characters can be used, Java provides an exception here. Eg.int x = 0X123Face;

d. Binary Literals

A literal in this type should have a prefix 0b and 0B, from 1.7 one can also specify in binary literals, i.e. 0 and 1.

Example: int x = 0b1111;

2. Floating-Point Literals in Java

Here, **datatypes** can only be specified in decimal forms and not in octal or hexadecimal form.

a. Decimal literals (Base 10)

int c = 0x123.222; // Hexa-decimal form

3. Char Literals in Java

There are four ways in which we can specify char literals in Java –

i. Single Quote

Java Literal can be specified to char data type as a single character within a single quote.

Example: char ch = 'a';

4. String Literals in Java

String literals in Java are any sequence of characters with a double quote.

Example: String s = "Hello";

5. Boolean Literals in Java

Java Boolean literals allow only two values i.e. true and false.

Example: boolean b = true;

Variables in Java

Variables in Java are basically memory allocations, named storages which can be later manipulated by the program. Each java variable has a specific type, which determines its memory and the data that can be stored in that variable.

There are three types of Java variables, they are

a. Local variables in Java

Local variables are defined within a java method or constructor and their scope remains limited to java blocks only. These Java variables are created when a function or block is called and deleted once they exit the block or call is returned.

b. Instance variables in Java

Java Instance variables are declared in a class outside any method, constructor, or block, it is of static type. These variables in Java are created when a object is created and destroyed once the object is destroyed.

c. Static variables in Java

We declare static keyword in java same as the instance variables, i.e. outside any object, block, or constructor, the difference being in java syntax that it uses the keyword static with it. It can also has as many copies which means it can be used for many classes. They create at start of program execution and destroy once the program ends.

Java Data types

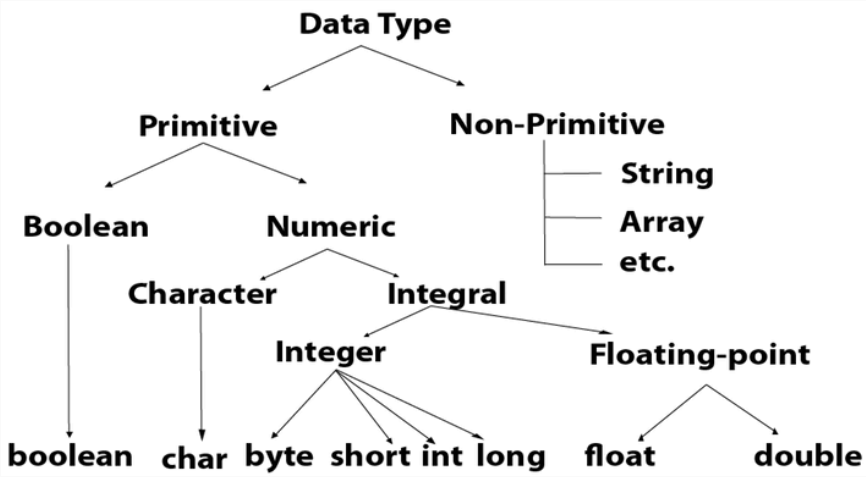
Java is a statically typed language, i.e. it does not allow to change the data type once declared, they cannot be changed and also **Java variable** and Java keyword should be assigned a data type, they are predefined.

There are basically two types of data types in Java-

a. Primitive Data type in Java
b. Non- Primitive Data type in Java

a. Primitive data type in Java

Java Primitive data types are the simple type, i.e. they don't have any special capabilities and are only single values. There are 8 primitive data types in Java, they are-



i. Java boolean

In a java boolean data type, only one bit of information can be represented, true or false. They cannot be changed either explicitly or implicitly.

Example: boolean b = true;

ii. Java byte

Byte data type in java is useful in saving memory for large arrays, it is a signed two's complement integer. It has a size of 8 bit with a range from -128 to 127.

Example: byte a = 126;

iii. Character (char)

A char data type in java is also a single character with a memory of 16 Bit, unicode character. It has a range from '\u0000' (or 0) to '\uffff' 65535.

Example: char a = 'G';

iv. Java short

The short data type in Java is similar to byte and is used to save memory in large arrays. It is also a two's compliment with a bit size of 16 bit and a range of -32,768 to 32,767 (inclusive).

Example: short a = 56;

v. int

Int data type in Java is two's compliment with a memory allocation of 32 bits and range lies between -2,147,483,648.

Example-

Example: int a = 56;

vi. long

The long data type in Java is two's compliment with a memory allocation of 64 bit and a range of -2^63 to 2^63-1.

Example: long a = 100000L;

vii. float

Float data type is single precision 32-bit floating point and cannot used for very precise values.

Example: float a = 4.5541132f;

viii. double

The double data type is a default choice when, it comes to decimal as it is a double precision 64 bit floating point variable.

Example: double a = 1.33526252726;

b. Non Primitive Data Types

The non-primitive data types in Java are created by the programmer during the coding process, they are known as the "reference variables" or "object variables" as they refer to a location where data is stored.

i. String

Strings are basically a collection of characters, they cannot be changed once they are created.

Example: String a="Java";

ii. Array and object

An array is a group of values that are similar in nature, they are dynamically allocated and can contain both primitive or objects depending on the definition. Their size has to be specified in integer. Their indexing always starts with zero, and they are ordered.

Example: int a[]=new int[5];

String Class

String is a sequence of characters. In java, string is an immutable object which means it is constant and can cannot be changed once it has been created.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";
String str2 = "Welcome";
```

Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

In this case compiler would create two different object in memory having the same string.

A Simple Java String Example

```
public class Example{
  public static void main(String args[]){
        String str = "Beginnersbook";
        char arrch[]={'h','e','l','l','o'};
        String str2 = new String(arrch);
        String str3 = new String("Java String Example");
        System.out.println(str);
        System.out.println(str2);
        System.out.println(str3);
  }
}
```

Output:

```
Beginnersbook
hello
Java String Example
```

Java String Class methods

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string.

1. toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

Example:

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
```

Output:

```
SACHIN
sachin
```

2. trim() method

The string trim() method eliminates white spaces before and after string.

3. charAt() method

The charAt() method returns a character at specified index.

Example:

```
System.out.println(s.charAt(0));
```

Output:

S

4. length() method

The string length() method returns length of the string.

Example:

System.out.println(s.length());

Output:

6

5. concat() method

The concat() method combines specified string at the end of this string. It returns combined string. It is like appending another string.

Example:

```
String s1=”java is”;
String s2=”best”;
System.out.println(s1.concat(s2));
```

Output:

Java is best

6. indexOf() method

The indexOf() method returns index of given character value or substring.

Example:

System.out.println(s.indexOf(‘c’));

Output:

2

7. replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

Example:

```
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String s2=s1.replace ("Java","Lava"); //replaces all occurrences of "Java" to "Kava"
System.out.println (s2);
```

Output:

Lava is a programming language. Lava is a platform. Lava is an Island.

8. substring() method

This function return a substring from original string by specifying starting and ending index.

Example:

```
String s1=”Programming”;
System.out.println(s1.substring(0,6));
```

Output:

Program

Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators

5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators

1. Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

- *** :** Multiplication
- **/ :** Division
- **% :** Modulo
- **+ :** Addition
- **– :** Subtraction

```
public class operators
{
   public static void main(String[] args)
   {
      int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
      System.out.println("a + b = "+(a + b));
      System.out.println("a - b = "+(a - b));
       System.out.println("a * b = "+(a * b));
      System.out.println("a / b = "+(a / b));
       System.out.println("a % b = "+(a % b));
    }
}
```

Output:

```
a+b = 30
a-b = 10
a*b = 200
a/b = 2
a%b = 0
```

2. **Unary Operators:** Unary operators needs only one operand. They are used to increment, decrement or negate a value.
   - **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
     - **Post-Increment :** Value is first used for computing the result and then incremented.
     - **Pre-Increment :** Value is incremented first and then result is computed.
   - **-- : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
     - **Post-decrement :** Value is first used for computing the result and then decremented.
     - **Pre-Decrement :** Value is decremented first and then result is computed.
   - **! : Logical not operator**, used for inverting a boolean value.

Example:

```
public class operators
{
   public static void main(String[] args)
   {
      int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
      boolean condition = true;
       c = ++a;
      System.out.println("Value of c (++a) = " + c);
```

```
        c = b++;
        System.out.println("Value of c (b++) = " + c);
        c = --d;
        System.out.println("Value of c (--d) = " + c);
        c = --e;
        System.out.println("Value of c (--e) = " + c);
        System.out.println("Value of !condition =" + !condition);
    }
}
```

Output:

```
Value of c (++a) = 21
Value of c (b++) = 10
Value of c (--d) = 19
Value of c (--e) = 39
Value of !condition =false
```

3. **Assignment Operator('=') :** Assignment operator is used to assign a value to any variable. The value given on right hand side of operator is assigned to the variable on the left and therefore right hand side value must be declared before using it or should be a constant. General format of assignment operator is,

```
variable = value;
```

In many cases assignment operator can be combined with other operators named as shorthand assignment operator. For example, instead of a = a+5 , we can write a += 5.

Example

```
int a = 5;
a += 5; //a = a+5;
```

4. **Relational Operator:** These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison.

Some of the relational operators are-

- **== (Equal to):** returns true of left hand side is equal to right hand side.
- **!= (Not Equal to):** returns true of left hand side is not equal to right hand side.
- **< (less than):** returns true of left hand side is less than right hand side.
- **<= (less than or equal to):** returns true of left hand side is less than or equal to right hand side.
- **> (Greater than):** returns true of left hand side is greater than right hand side.
- **>= (Greater than or equal to):** returns true of left hand side is greater than or equal to right hand side.

5. **Logical Operators:** These operators are used to perform "logical AND" and "logical OR" operation. One thing to keep in mind is the second condition is not evaluated if the first one is false. Used extensively to test for several conditions for Conditional operators are-
   - **&&(Logical AND):** returns true when both conditions are true.
   - **|| (Logical OR):** returns true if at least one condition is true.
6. **Conditional operator:** conditional operator also called Ternary operator. It has three operands and hence the name ternary. General format is-

```
condition ? if true : if false
```

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

7. **Bitwise Operators :** These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types
   - **& (Bitwise AND operator):** returns bit by bit AND of input values.
   - **| (Bitwise OR operator):** returns bit by bit OR of input values.
   - **^ (Bitwise XOR operator):** returns bit by bit XOR of input values.
   - **~ (Bitwise Complement Operator):** returns the one's compliment of input value.

Type conversion

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion (Implicit Conversion) and if not then they need to be converted explicitly. For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Example:

```
int i = 100;
long l = i;        //automatic type conversion
float f = i;       //automatic type conversion
System.out.println("Int value "+i);
System.out.println("Long value "+l);
System.out.println("Float value "+f);
```

Output:
Int value 100
Long value 100
Float value 100.0

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type, then it is said to be explicit conversion or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

char and number are not compatible with each other. Let's see when we try to convert one into other.

Example:

```
class Test
{
   public static void main(String[] args)
   {
      double d = 100.04;
```

```
        long l = (long)d;
        int i = (int)l;
       System.out.println("Double value "+d);
       System.out.println("Long value "+l);
       System.out.println("Int value "+i);
   }
}
```

Output:

```
Double value 100.04
Long value 100
Int value 100
```

Constant in Java

A constant is fixed value which does not changes during the execution of the program. Constants are used in programming to make code a bit more robust and human readable.

In java a constant can be declared either using final or static keyword.

The final Keyword

When you declare a variable to be final we are telling Java that we will not allow the variable's "pointer" to the value to be changed.

The final keyword means is that once the value has been assigned, it cannot be re-assigned.

Example

```
Class fdemo{
final int price90;
Void run()
{
 price = 400; //Cannot assign value to final variable
}
public static void main(String args[])
{
fdemo xx=new fdemo();
f1.run();
System.out.println(price);
}
}
```

Scope

Scope refers to the lifetime and accessibility of a variable. How large the scope is depends on where a variable is declared. For example, if a variable is declared at the top of a class then it will accessible to all of the class methods. If it's declared in a method then it can only be used in that method.

Comments in Java

In a program, comments take part in making the program become more human readable by placing the detail of code involved and proper use of comments makes maintenance easier and finding bugs easily. Comments are ignored by the compiler while compiling a code.

In Java there are three types of comments:

1. Single – line comments.
2. Multi – line comments.
3. Documentation comments.

1. Single-line Comments

A beginner level programmer uses mostly single-line comments for describing the code functionality. Its the most easiest typed comments.

Syntax:

```
//Comments here( Text in this line only is considered as comment )
```

2. Multi-line Comments

To describe a full method in a code or a complex snippet single line comments can be tedious to write. So to overcome this multiline comments can be used. The multiline comment starts with '/*' and ends with '*/' notation.

Syntax:

```
/*Comment starts
continues
continues
.Commnent ends*/
```

}

We can also accomplish single line comments by using the above syntax as shown below:

```
/*Comment line 1*/
```

Documentation Comments

This type of comments are used generally when writing code for a project/software package, since it helps to generate a documentation page for reference, which can be used for getting information about methods present, its parameters, etc.

documentation page which is generated by using a javadoc tool for processing the comments.

Syntax:

```
/**Comment start
*
*tags are used in order to specify a parameter
*or method or heading
*HTML tags can also be used
*such as <h1>
*
*comment ends*/
```

Keyboard Input

There are various ways to read input from the keyboard during the execution of the program. Accepting keyboard input in Java is done using a Scanner object.

Consider the following statement

```
Scanner console = new Scanner (System.in)
```

This statement declares a reference variable named console. The Scanner object is associated with standard input device (System.in).

To get input from keyboard, you can call methods of Scanner class. For example in following statement. The nextInt() method of Scanner takes an integer and returns to variable x :

Some other useful methods of Scanner class

Method	Returns
int nextInt()	Returns the next token as an int.
float nextFloat()	Returns the next token as a float.
double nextDouble()	Returns the next token as a long.
String next()	Finds and returns the next complete token as a string ended by a blank.
String nextLine()	Returns the rest of the current line, excluding any line separator at the end.

Example:

```
import java.util.Scanner;    // Needed for Scanner class
public class RectangleArea
{
  public static void main(String[] args)
  {
    int length, width;
    int area;
   Scanner console = new Scanner(System.in);
   System.out.print("Enter length ");
    length = console.nextInt();
    System.out.print("Enter width ");
    width = console.nextInt();
    area = length * width;
    System.out.println("The area of rectangle is " + area);
  }
}
```

Output :

```
Enter length 5
Enter width 8
The area of rectangle is 40
```

Example :

```
import java.util.Scanner;
public class ReadEmployee
{
   public static void main(String[] args)
   {
      String name; // To hold the employee's name
      int age; // To hold the employee's age
      char gender; // To hold the employee's gender
      double salary; // To hold the employee's salary
      Scanner console = new Scanner(System.in);
      System.out.print("Enter name: ");
      name = console.nextLine();
      System.out.print("Enter age: ");
      age = console.nextInt();
      System.out.print("Enter gender: ");
      gender = console.next().charAt(0);
```

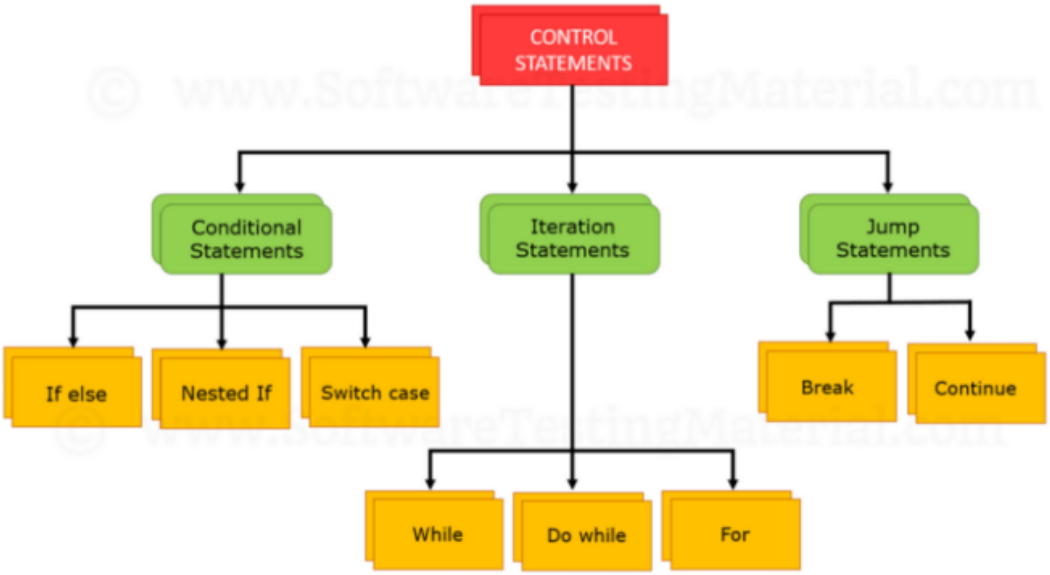
```
        System.out.print("Enter salary: ");
        salary = console.nextDouble();
        System.out.println("Name: " + name + " Age: " + age + " Gender: "
                + gender + " Salary: " + salary);

    }
}
```

Output :

```
Enter name: Alex Joseph
Enter age: 24
Enter gender: M
Enter salary: 8000
Name: Alex Joseph Age: 24 Gender: M Salary: 8000.0
```

Control State,emts

The different types of control statements in java includes



1.Conditional statement

If-else Statement

The Java if statement is used to test the condition. It checks condition and return Boolean value : true or false. There are various types of if statement in java.

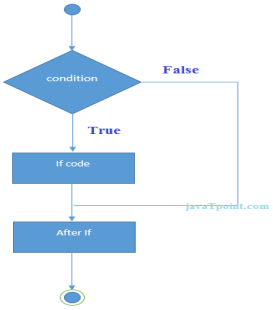
- if statement
- if-else statement
- if-else-if ladder
- nested if statement

If Statement

The Java if statement tests the condition. It executes the if block if condition is true.

Syntax:

```
if(condition)
{
//code to be executed
}
```

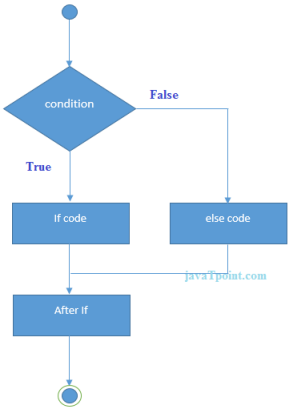


If-else Statement

The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

Syntax:

```
if(condition)
{
//code if condition is true
}
else
{
//code if condition is false
}
```



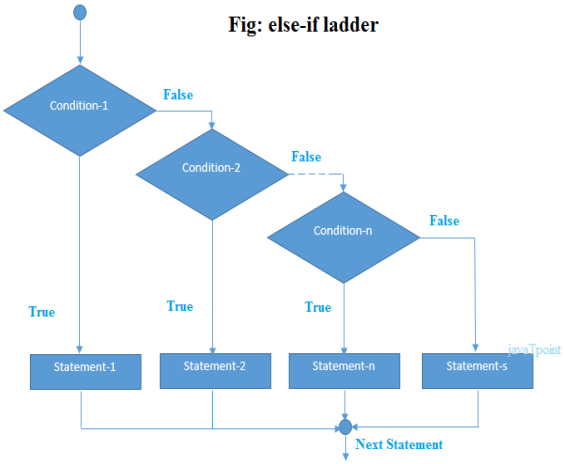
If-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
        //code to be executed if condition1 is true
}else if(condition2){
        //code to be executed if condition2 is true
}
else if(condition3){
        //code to be executed if condition3 is true
}
else{
        //code to be executed if all the conditions are false
}
```

Fig: else-if ladder



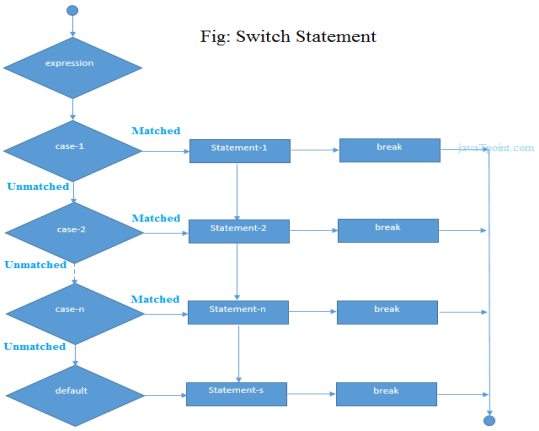
Switch Statement

switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
default:
 code to be executed if all cases are not matched;
}
```

Fig: Switch Statement



2. Iteration/Loop statements

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

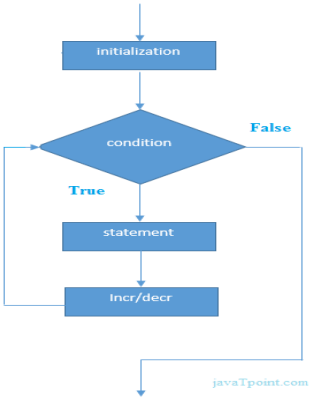
- for loop
- while loop
- do-while loop

For Loop Statement

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loops in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop



Syntax:

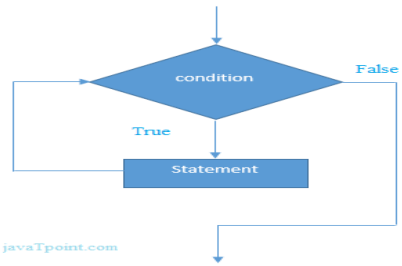
```
for(initialization;condition;incr/decr)
{
//code to be executed
}
```

While Loop Statement

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition)
{
  //code to be executed
}
```



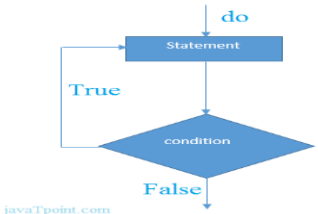
Do-while Loop

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and to execute the loop at least once, it is recommended to use do-while loop.

The Java do-while loop is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed
}while(condition);
```



3. Jump Statements

Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The break is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

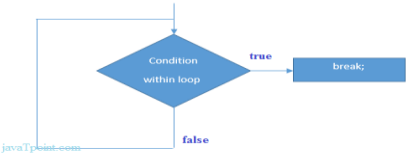


Figure: Flowchart of break statement

Continue Statement

The continue statement is used in loop control structure when you need to immediately jump to the next iteration of the loop. It can be used with for loop or while loop.

The Java continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

Object and Class in Java

In object-oriented programming technique, we design a program using objects and classes. Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.
An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. A class is a user defined data type.

A class in Java can contain:

- Data(variable)
- methods
- constructors
- blocks
- nested class and interface

Syntax to declare a class:

```
class <class_name>
{
   datatype variable1;
   datatype variable2;
return type method_name(parameter list)
{
// body of the method
}
}
```

Instance variable in Java

A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object is created. That is why, it is known as instance variable.

Method in Java

In java, a method is block of statements to perform a particular task as like function i.e. used to expose behavior of an object.

new keyword in Java

The new keyword is used to allocate memory for the objects at run time. All objects get memory in Heap memory area. This keyword also calls the constructor of the object.

```
class Student
{
    int id;
    String name;
    public static void main(String args[])
    {
    Student s1=new Student();//creating an object of Student
    System.out.println(s1.id);//accessing member through reference variable
    System.out.println(s1.name);
     }
}
```

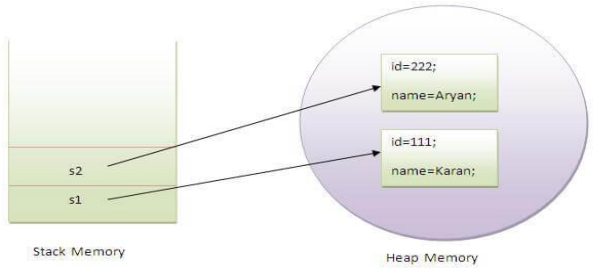
Output:

```
0
null
```

Object and Class Example: main outside class

In real time development, we create classes and use it from another class having main function. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```



Constructor in Java

In Java, constructor is a block of codes similar to method. It is called when an instance of object is created and memory is allocated for the object. It is a special type of method which is used to initialize the object.

A constructor is called every time an object is created using new() keyword, atleast one constructor is called. It is called a default constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors in java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

1. Default Constructor

When the user does not explicitly define a constructor, Java creates a Default Constructor. A constructor is called "Default Constructor" when it doesn't have any parameter and it initializes all instace variables to 0 or null.

```
class Bike1
{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

2. Parameterized constructor

A constructor which has a specific number of parameters is called parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

```
class Student4{
int id;
String name;
Student4(int i,String n){
id = i;
name = n;
}
void display(){System.out.println(id+" "+name);
}
public static void main(String args[]){
Student4 s1 = new Student4(111,"Karan");
Student4 s2 = new Student4(222,"Aryan");
s1.display();
s2.display();
} }
```

Output:

```
111 Karan
222 Aryan
```

Method Overloading in Java

If a class has multiple methods having same name but different parameter list, it is known as **Method Overloading**. If we have to perform only one operation, having same name of the methods increases the readability of the program.

To perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it is difficult for other programmers to understand the behavior of the method because its name differs.

Example:

```
public class Sum {
    public int sum(int x, int y)
   {
      return (x + y);
   }
   public int sum(int x, int y, int z)
   {
      return (x + y + z);
   }
   public double sum(double x, double y)
   {
      return (x + y);
   }
   public static void main(String args[])
   {
      Sum s = new Sum();
      System.out.println(s.sum(10, 20));
      System.out.println(s.sum(10, 20, 30));
      System.out.println(s.sum(10.5, 20.5));
   } }
```

Output :

```
30
60
31.0
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like methods. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task.

```
class Student5{
int id, age;
String name;
Student5(int i,String n){
id = i;
name = n;
}
```

```
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){ System.out.println(id+" "+name+" "+age);
}
    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Static keyword

The **static keyword** in java is applied to variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

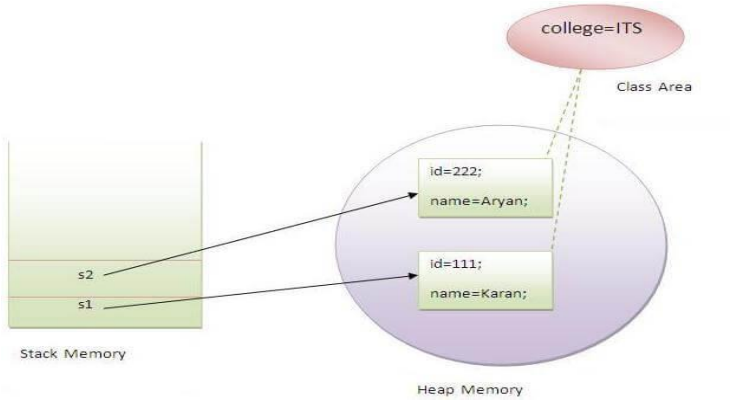
1) Java static variable

The static variable can be used to refer the common property of all objects The static variable gets memory only once in class area at the time of class loading. It makes your program **memory efficient** (i.e it saves memory).

```
class Student8{
  int rollno;
  String name;
  static String college ="ITS";
  Student8(int r,String n){
  rollno = r;
  name = n;
  }
 void display (){System.out.println(rollno+" "+name+" "+college);
  }
 public static void main(String args[]){
 Student8 s1 = new Student8(111,"Karan");
 Student8 s2 = new Student8(222,"Aryan");
 s1.display();
 s2.display();
 } }
```

Output:

```
111 Karan ITS
222 Aryan ITS
```



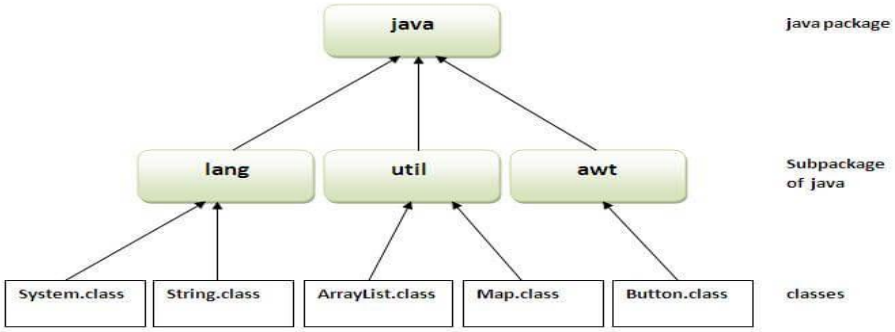
Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package.

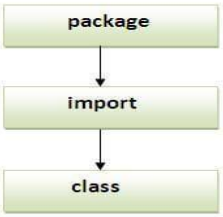
There are many built-in packages such as java.lang, awt, javax, swing, net, io, util, sql etc.
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.




```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  }
}
```



Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.
The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

1) **private access modifier**

The private access modifier is accessible only within class.

Example:

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
public static void main(String args[]){
A obj=new A();
System.out.println(obj.data);//Compile Time Error
obj.msg();//Compile Time Error
}
}
```

2) **Protected access modifier**

The **protected access modifier** is accessible within package and outside the package but through inheritance only. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

```
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
  class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Output:Hello

3) public access modifier

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

```
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
    A obj = new A();
    obj.msg();
  }  }
```

Output:Hello

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. You can pass **any** numbers of arguments from the command prompt.

```
class CommandLineExample
{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

compile by > javac CommandLineExample.java

run by > java CommandLineExample sonoo

Output: Your first argument is: sonoo

toString() method

If you want to represent any object as a string, **toString() method** comes into existence. The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object, values of the object. Depends on your implementation.

Example without toString() method

```
class Student{
int rollno;
String name;
String city;
Student(int rollno, String name, String city){
this.rollno=rollno;
this.name=name;
this.city=city;
}
public static void main(String args[]){
Student s1=new Student(101,"Raj","lucknow");
Student s2=new Student(102,"Vijay","ghaziabad");
System.out.println(s1);//compiler writes here s1.toString()
System.out.println(s2);//compiler writes here s2.toString()
}
}
```

Output:Student@1fee6fc
Student@1eed786

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects instead values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values.

Example of Java toString() method

```
class Student{
int rollno;
String name;
String city;
Student(int rollno, String name, String city){
this.rollno=rollno;
```

```
this.name=name;
this.city=city;
}
public String toString(){//overriding the toString() method
return rollno+" "+name+" "+city;
}
public static void main(String args[]){
Student s1=new Student(101,"Raj","lucknow");
Student s2=new Student(102,"Vijay","ghaziabad");
System.out.println(s1);//compiler writes here s1.toString()
System.out.println(s2);//compiler writes here s2.toString()
}
}
```

Output:

```
101 Raj lucknow
102 Vijay Ghaziabad
```

this keyword in java

In java this keyword **reference variable** that refers to the current object. Usage of this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.

this keyword resolves the problem of ambiguity between the instance variables and parameters,
Understanding the problem without this keyword

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

Enumeration(enum)

enum is a data type that contains fixed set of constants.It is mainly used to define user data type.

It can be used for days of the week, directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly.

Points to remember for Java Enum

- enum improves type safety and easily used in switch
- enum can have variables, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Simple example of java enum

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }
public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);
}
}
```

Output:

```
WINTER
SPRING
SUMMER
FALL
```

Garbage Collection

In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

  It is **automatically done** by the garbage collector(a part of JVM).

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

Employee e=**new** Employee();

e=**null**;

2) By assigning a reference to another:

Employee e1=**new** Employee();

Employee e2=**new** Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){}

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
Simple Example of garbage collection in java
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

Output:

```
object is garbage collected
object is garbage collected
```

References:

1. Herbert Schildt, Java 2: Complete Reference, fifth edition, Tata McGraw Hill Publishers.

2. P. Rizwan Ahmed, Java Programming, Second Edition, Margham Publications.

Web References:

1. **URL**: http://www.javatpoint.com

2. **URL**: http://docs.oracle.com/javase/

3. **URL**: http://www.tutorialspoint.com