UNIT-1

Java is a class-based and object-oriented programming language. It is a high level language and also platform independent.

Where Java is used?
There are many devices where Java is currently used.

- Desktop applications such as Acrobat Reader, Media Player, Anti-Virus etc.
- Web applications such as irctc.co.in.
- Enterprise applications such as banking applications.
- Mobile applications.
- Embedded system.
- Smart card.
- Robotics.
- Games etc.

History of java:

- ➢ Java was developed by James Gosling in 1995, he is known as the father of java. Java team members are known as Green Team, they initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc. Currently, Java is used in internet programming, mobile devices, games, e-business applications etc.
  Firstly it was called "Greentalk" by James Gosling. Later, it is changed to “Oak”.

Why Oak name for Java language?

- ➢ Oak is a symbol of strength and choosen 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.

Features of Java (buzz words)

There is given many features of java. They are also known as java buzzwords.

1. Simple
2. Object-Oriented
3. Portable and Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Multithreaded
8. Distributed applications

Simple:

Java is simple language. It does not use pointer, goto statements, etc. It eliminates operator overloading and multiple inheritance.

Object-Oriented:

Everything in java is an object. Object oriented means we organise our software as a combination of different types of objects that incorporates both data and behaviour.

Portable and Platform independent:

Write once and run anywhere. Java programs are portable because of its ability to run the program on any platform and no dependency on the underlying hardware/operating system.

Secured:

Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet. The absence of pointers in Java ensures that programs cannot get right of entry to memory location without proper approval.

Robust:

Java is a most strong language which provides many securities to make certain reliable code. It is design as garbage –collected language, which helps the programmers virtually from all memory management problems. Java also includes the concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system.

Architecture neutral:

Irrespective of architecture the memory allocated to the variables will not vary. Whatever the architecture we are using that memory allocated for the variables is same.

Eg: In C language the size of the datatype is depends on the architecture of the compiler, So if we take integer variable in 16bit compiler it occupies 2bytes of memory, and if it is 32bit compiler it is 4bytes of memory. Here, the memory allocation depends on the architecture.

Multithreaded:

Concurrent execution of several parts of same program at same time. So this will improve CPU utilization.

Distributed applications:

It is software that runs on multiple computers connected to a network at the same time. [A java program is able to create this Distributed applications].

Difference between C and JAVA

	C		JAVA
1	It is procedure oriented language.	1	It is object oriented language.
2	C is divided into small parts called functions.	2	Java is divided into some parts called class and objects.
3	C is middle level language.	3	Java is high level language.
4	C is platform dependent.	4	Java is platform independent.
5	C doesn’t support Multithreading.	5	Java supports Multi-threading.
6	C supports pointers.	6	Java does not supports pointers.
7	In C garbage collection needs to managed manually.	7	In java it is automatically managed by a Garbage collector.
8	In C declaration variables are declared at the beginning.	8	In Java variables are declared any where.
9	free() is a function to release the memory in C.	9	A compiler will free up the memory by calling the Garbage collector.
10	C does not have a feature of overloading functionality.	10	Java supports method overloading.
11	Function oriented program follows top to down approach.	11	Object oriented programming follows bottom up approach.
12	C does not have access specifiers/modifiers.	12	It has 3-access specifiers public, private, protected and default.
13	Data can move freely from function to function in the system.	13	Objects can move and can communicate with each other.
14	No data hiding is possible hence, security is not possible.	14	In Java provides data hiding hence, security is possible.

<u>**Oops concepts**</u>

Object oriented programming (Oops) is a methodology that simplifies software development and maintenance by providing some rules.

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

Structure of java program:

- Documentation Section
- Package Statement
- Import Statements
- Interface Statements
- Class Definitions
- Main method definition

Documentation Section:

It is an important section but optional for a java program. It includes basic information about a java program. The information includes the description of the program. Whatever we write in the documentation section, the java compiler ignores the statements during execution of the program. To write the statements in the documentation section, we use comments. The comments may be single-line, multi-line and documentation comments.

Single-line comment: **//**Example for Single line comment
Multi-line comment: /*it is an example of multiline comment*/
Documentation comment: /**it is an example of documentation comment***/ (**This comment style is new in java. The main purpose of this type of comment is to automatically generate program documentation. The java doc tool reads these comments and uses them to prepare your program's documentation in HTML format.**)**

Package Declaration:

It is an optional declaration. Package name can be defined by programmer with any name. Package contains a group of classes. By default compiler consider as these classes, which are defined under it belongs to this package.

Package packagename;

Import statements:

It contains many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. We use the import keyword to import the class.

import java.util.Scanner; //it imports the scanner class
import java.util.*; //it imports all the class of the java.

Interface Section:

An interface is similar to classes, which consist of group of method declaration. It is an optional section and can be used when programmers want to implement multiple inheritance.

```
Interface car
{
Void start();
Void stop();
}
```

Class Definition:

In this section, we define the class. Without the class, we cannot create any java program. We use the **class keyword** to define the class.

```
class  Example  //class definition
{
}
```

Main method:

In this section, we define the main() method. It is essential for all java programs. Because the execution of all java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class.

```
public static void main(String args[])
{
}
```

Java Example

Let's have a quick look at java programming example

1. class Simple {
2. public static void main(String args[]) {
3. System.out.println("Hello Java");
4. }
5. }

Understanding public static void main(String args[]) in java:

The point from where the program starts its execution or simply the entry point of java programs is the main() method.

Public:

It is an access modifier, which specifies from where the main() method public makes it globally available. It is made public, so that JVM (java virtual machine) can invoke it from outside the class as it is not present in the current class.

```
class Example

{

private static void main(String[] args)
```

```
{
System.out.print(“a”);
}
Output:
Error: method not found in class, please define the main method as:
public static void main(String[] args)
```

Static (Direct access without object):

It is a keyword when it is associated with a method, makes it a class related method. The main() method is static so that JVM can invoke it without instantiating the class(Without creating an object for a class).

```
class Ex
{
public void main(String[] args)
{
System.out.print(“a”);
}
Output: Error: method not found in class Ex, please define the main method as:
public static void main(String[] args).
```

Void:

Void is a **keyword** and used to specify that, a method doesn’t return anything. As main() method doesn’t return anything, its return type is void.

Main:

It is the name of java main method & it is the identifier that the JVM looks for as the starting point of the java program.

Note: main is not a keyword.

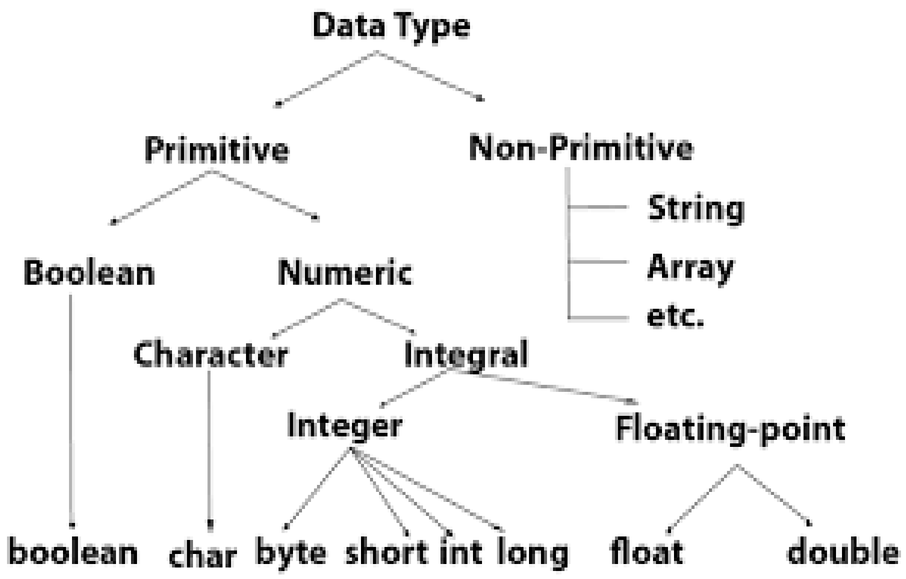
String[] args:

It stores java command line arguments and is an array of type. (When you run a java program with command prompt or want to give command line arguments, then “String[] args” is used) Here, the name of the string array is args but it is not fixed and user can use any name.

Data types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interface and Arrays.



Each data type have some memory size, whenever a variable is declared with a data type, the memory size is automatically defined by the JVM.

Variables

Variable in java is a container that holds the value during the execution of program it is assigned with a data type, a variable is a name of the memory location for storing value.

There are three types of variables:

1. Local variable
2. Instance variable
3. Static/class variable.

Local variable:

A variable that is declared and used inside the body of methods, constructor or blocks is called local variable, it cannot be defined with "static" keyword.

Local variable must be assigned a value at the time of declaration.

No access modifiers can be used with local variables.

A local variable cannot be static.

Syntax:

```
class class_name
{
        return_type method_name()
        {
        data_type variable=value;
        }
}
```

```
//Example-1 for local variable
class Localvar
{
        void m(int x,int y)
        {
                int z=x+y;
                System.out.println(z);
        }
        public static void main(String[] args)
        {
                 Localvar v= new Localvar();
                v.m(10,20);
        }
}
```

OUTPUT:

```
30
```

```
//Example-2 program for local variable

public class Test
{
  public void displayAge()
  {
    int age = 0;
    age = age + 7;
    System.out.println("Age is : " + age);
  }

  public static void main(String args[])
  {
    Test test = new Test();
    test.displayAge();
  }
}
```

Output:

```
Age is : 7
```

Instance variable:

A variable that is declared inside the class but outside the body of a method, constructor or any block are called instance variable.

The instance variable is initialized at the time of the class loading or when an object of the class is created.
An instance variable can be declared using different access modifiers available in Java like default, private, public, and protected.

Syntax:

```
class class_name

{

        data_type  variable;

        return_type method_name()

        {

        --------
        ---------
        }
}
```

```
//Example program for instance variable
class Instancevar
{
        int a=10;
        public static void main(String[] args)
        {
                Instancevar var=new Instancevar();
                System.out.println(var.a);
        }
}
```

OUTPUT:

```
10
```

Static variable:

A variable which is declared with a static keyword is called a static variable. It is also called as a class variable, because it is associated with a class.

Static variables are always declared inside the class but outside of any methods, constructors or any blocks.

Static variable will get the memory only once. If we changes the value of static variable, it will replace the previous value and display the changed value, this is because it is constant for every object created.

Syntax:

```
class class_name
{
        static data_type variable;
        return_type method_name()
        {
```

```
        …………
        ………….
        }
}
//Example program for static variable
class Staticvar
{
        static int a=20;
        public static void main(String[] args)
        {
                Staticvar obj=new Staticvar();
                System.out.println(obj.a);
                System.out.println(Staticvar.a);
        }
}
```

OUTPUT:

```
10
10
```

```
//program to demonstrate local, instance, and static variables
class Example
{
        int a=10;
        static int b=20;
        void m()
        {
                int c=30;
                int d=a+b+c;
                System.out.println(d);
        }
        public static void main(String[] args)
        {
                Example e=new Example();
                System.out.println(Example.b);
                System.out.println(e.b);
                e.m();
                System.out.println(e.a);
        }
}
```

OUTPUT:

```
20
20
60
10
```

Scope and life time of variable:

	Scope	Lifetime of variable
Local variable:	Scope of local variable is within the block in which it is declared.	Execution time of block or Until the control leaves the block in which it is declared.
Instance variable:	Throughout the class except in Static methods.	Until the object stays in memory.
Static variable:	Throughout the class	Until the end of the program.

Type Conversion and type casting

Type conversion:

It is a process of converting a value from one data type to another data type. If the data types are compatible then java can perform the conversion automatically or implicitly, If not they need to be casting explicitly.

Types of conversion:

1. Widening/implicit type conversion.
2. Narrowing/explicit type conversion.

Widening:

Converting a lower data type to a higher data type is known as widening. In this case the conversion is done automatically also called as implicit type casting.

byte → short → int → long → float → double

```
//Program to demonstrate  implicit type conversion

  class TypeConversion

 {

        public static void main(String[] args)

        {

                int a=10;

                long b=a;

                float c=a;

                System.out.println("Integer value:" +a);

                System.out.println("long value:" +b);

                System.out.println("Float value:" +c);

        }

}
```

Output:

```
Integer value: 10

long value:10

Float value:10.00
```

Narrowing:

Converting a higher data type to the lower data type is narrowing. In this case the casting or conversion is not done automatically, user need to convert explicitly using the cast operator “( )”. Therefore it is known explicit type casting.

double ← long ← float ← int ← short ← byte.

Syntax: ↱ target data type

```
        Variable_name(Type_name)variable;
```

```
//Program to demonstrate  explicit type conversion

class TypeConversion

{

        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
```

<u>Control statements</u>

Control statements is one of the fundamentals required for programming.

It determines which statement to execute, and it controls the flow of execution of the program.

Statements can be executed multiple times or only under a specific condition.

The ”selection” statements are used for testing conditions, the “ iteration” statements are used to create cycles, and the jump statements is used to alter a loop.

Control statements can be divided into the following categories:

1. Selection statements(Decision making statements)
2. Iteration statements(Loop statements)
3. Jump statements(Branching statements)

Selection Statements:

1. Simple if statement
2. if-else statement
3. if-else-if ladder statement
4. nested if statement
5. switch statement

Simple if statement:

It instructs the program to execute only certain part of the code if the condition is true.

The Boolean value is true then statement will be executed.

Syntax:

```
if(condition)

{

        statement; //executes when condition is true

}
```

// example program for Simple if condition statement

```
public class Student
{
public static void main(String[] args)
{
        int x = 10;
        int y = 12;
        if(x+y > 20)
        {
                System.out.println("x + y is greater than 20");
        }
}
}
```

if-else statement:

The if-else statement is an extension to the if statement, which uses another block of code, i.e., else block.

The else block is executed if the condition of the if block is evaluated as false.

Syntax:

```
if(condition)
{
        statement 1; //executes when condition is true
}
else
{
        statement 2; //executes when condition is false
}
```

//Write a java program for if-else condition statement

```
class ifelse
{
public static void main(String args[])
        {
                int a=15;
                if(a>20)
                {
                        System.out.println("a is greater than 10");
                }
                else
                {
                        System.out.println("a is less than 10");
                }
        }
}
```

if-else-if ladder statement:

The if-else-if statement contains the if statement followed by multiple else-if statements.

It is used to decide among multiple options.

Syntax:

```
if(condition 1)
 {
        statement 1; //executes when condition 1 is true
}
else if(condition 2)
{
        statement 2; //executes when condition 2 is true
}
else
 {
```

```
                statement 2; //executes when all the conditions are false
        }
```

//program for if-else-if ;adder condition statement

```
public class Student
{
public static void main(String[] args)
{
    String city = "Delhi";
    if(city == "Meerut")
    {
    System.out.println("city is meerut");
    }
        else if (city == "Noida")
        {
            System.out.println("city is noida");
        }
        else if(city == "Agra")
        {
            System.out.println("city is agra");
        }
    else
    {
        System.out.println(city);
    }
}
}
```

<u>Nested if statement:</u>

An if present inside an if block is known as a nested if.

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

<u>Syntax:</u>

```
    if(condition 1)
    {
        statement 1; //executes when condition 1 is true
    if(condition 2)
    {
        statement 2; //executes when condition 2 is true
    }
    else
    {
        statement 2; //executes when condition 2 is false
    }
```

```
        }
```

```
//Example program for nested if statement
class Test
{
public static void main(String args[])
{
        int x=20;
        int y=15;
        int z=30;
        if(x>y)
        {
                if(x>z)
                {
                        System.out.println("Largest no is:" +x);
                        else
                        System.out.println("Largest no is:" +z);
                }
                else
                {
                if(y>z)
                        System.out.println("Largest no is:"+y);
                        else
                        System.out.println("Largest no is:"+z);
                }
        }
}
}
```

Switch statement:

The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable or expression which is being switched.

- The case variables can be int, short, byte, char, enums and strings.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- Cases cannot be duplicate.
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.

Syntax:

```
switch(expression)
{
  case value1 :
    // Statements
    break; // break is optional
  case value2 :
    // Statements
    break; // break is optional
  default :
    // Statements
          // No break is needed in the default case.
}
//program for switch case

class Music
{
public static void main(String args[])
{
        int instrument=4;
        String musicInstrument;
        //switch statement with int data type
        switch(instrument)
{
        case 1:
        musicInstrument=”Guitar”;
        break;
        case 2:
        musicInstrument=”Piano”;
        break;
        case 3:
        musicInstrument=”Drums”;
        break;
        case 4:
        musicInstrument=”Flute”;
        break;
        default:
        musicInstrument=”invalid”;
}
```

Arrays

Array is a collection of homogeneous elements.

Arrays are objects in java, we can find their length using the object.

There are three main features of an array.

1) Dynamic allocation:
   In arrays, the memory is created dynamically, which reduces the amount of storage required for the code.
2) Elements stored under a single name:
   All the elements are stored under one name. This name is used any time we use an array.
3) Occupies contiguous location:
   The elements in the arrays are stored at adjacent positions. This makes it easy for the user to find locations of its elements.

Advantages:

- Arrays enable you to access any element randomly with the help of indexes.
- It is easy to store and manipulate large data sets.

Disadvantages:

- The size of the array cannot be increased or decreased once it is declared.(Arrays have a fixed size).
- It cannot store heterogeneous data. It can only store a single type of primitives.

Syntax:

```
Type array-name[];

        (Or)

Type[] array-name;
```

- ➢ To mention the size of the array

```
Array ref = new type[size];
```

- ➢ "Length property" is to know the size of the array.

```
Array_name.length;   //arr.length;
```

- ➢ The declaration of array is:

```
int arr[];             //Declaring array.
arr = new int[20];     // allocating memory to array.
        (Or)
int[] arr=new int[20]; // Combining both statements in one
```

Types of arrays:

1) One-Dimensional Array.
2) Multi-Dimensional Array.

<u>One-Dimensional Array:</u>

One Dimensional Array is always used with only one subscript([]). A one-dimensional array behaves likes a list of variables.

/* Program to illustrate creating an array of integers, put some values in the array and prints each value to standard output*/

```
class ArrayEx

{

        public static void main(String[] args)

{

        int[] arr;        //declares an array

        arr=new int[5];         // allocating memory for 5 integers.

        arr[0]=10;      // initialise the first elements of the array

        arr[1]=20;

        arr[2]=30;

        arr[3]=40;

        arr[4]=50;

for(int i=0;i<arr.length;i++)   //accessing the elements of the specified array

System.out.println("Element at index" +i+  ":" +arr[i]);

}

}
```

Output:

```
Elements at index 0: 10

Elements at index 1: 20

Elements at index 2: 30
Elements at index 3: 40

Elements at index 4: 50
```

Multi-Dimensional Array.

Multi Dimensional Array is used with two or more subscript([]).

```
//Example program for multi dimensional array
        class MultiDimensional
        {
                public static void main(String[] args)
                {
                int arr[][]={{1,2,3},{4,5,6},{7,8,9}};
                for(int i=0;i<3;i++)
                {
                        System.out.println(arr[i][j]+" ");
                        System.out.println();
                }
                }
        }
```

Output:

```
1 2 3

4 5 6

7 8 9
```

Jagged array:

An Array which is collection of arrays or an array which is holding arrays are known as Jagged arrays.

```
public class Testarray3{
```

```
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5,6}};
//printing 2D array
for(int i=0;i<arr.length;i++){
 for(int j=0;j<arr[i].length;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Static fields and methods:

Static is a keyword, it is used with class, variables, methods and block.

It is mainly used for memory management.

Static keyword belongs to the class instead of a specific instance of the class.

Static can be accessed without creating object.

The static can be:

- Variable
- Method
- Block

Static variable:

The static variable can be used to refer to the common property of all objects (which is not unique for each object).

Advantages:

- It makes your program memory efficient (i.e., it saves memory).

//Java Program to demonstrate the use of static variable

```
class Student{
int rollno;                    //instance variable
String name;
static String college ="RGM";                    //static variable
Student(int r, String n)
{
rollno = r;
name = n;
}
        void display ()
        {
        System.out.println(rollno+" "+name+" "+college);
        }
        }
        public class TestStaticVariable1        //Test class to show the values of objects
        {
        public static void main(String args[])
        {
        Student s1 = new Student(111,"abc");
        Student s2 = new Student(222,"xyz");
        s1.display();
        s2.display();
         }
        }
```

Output:

```
111 abc RGM
222 xyz RGM
```

static method:

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without creating an instance(object) of a class.
- A static method can access static data member and can change the value of it.

//Java Program to demonstrate the use of a static method.

```
class Student
{
        int rollno;
        String name;
        static String college = “RGM";
        static void change()
{
```

```
                college = "SREC";
        }
                Student(int r, String n)
        {
                rollno = r;
                name = n;
        }

                void display()
        {
                System.out.println(rollno+" "+name+" "+college);
        }
        public class TestStaticMethod
        {
        public static void main(String args[])
        {
        Student.change();                    //calling change method
        Student s1 = new Student(1,"abc");
        Student s2 = new Student(2,"xyz");
        Student s3 = new Student(3,"def");
        s1.display();
        s2.display();
        s3.display();
}
}
```

<u>OUTPUT:</u>

```
1 abc
2 xyz
3 def
```

<u>Static Block:</u>

It is used to initialize the static member.
It is executed before the main method at the time of classloading.

```
// example program for static block

class A
{
        static
        {
        System.out.println("static block is invoked");
        public static void main(String args[])
        {
         System.out.println("main method");
        }
}
```

<u>OUTPUT:</u>

```
        static block is invoked
        main method
```

Access control:

The access modifiers specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of access modifiers:

1. Public
2. Private
3. Protected
4. Default

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

1. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

// Example of public access modifier

```
        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

2. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

//Simple example of private access modifier

```
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

  }

  }
```

3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

//Simple example of protected access modifier

```
package pack;
        public class A
        {
                protected void msg()
        {
                System.out.println("Hello");
        }
        }

        package mypack;
                 import pack.*;

                class B extends A
                {
                        public static void main(String args[])
                {
                        B obj = new B();
                        obj.msg();
                }

                }
```

Output: Hello

4. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

//Simple example of default access modifier

```
package pack;
        class A
        {
                void msg()
        {
                System.out.println("Hello");
        }

        }

 package mypack;

        import pack.*;
        class B
        {
                public static void main(String args[])
        {
                A obj = new A();        //Compile Time Error
                obj.msg();       //Compile Time Error
        }

        }
```

<u>this</u>

'this' keyword is a reference variable that refers to the current object of a method or a constructor.

The main purpose of using this keyword is to remove the confusion between class attributes and parameters that have same names.

uses of 'this' keyword:

- this can be used to refer current class instance variable. // this.variable_name;
- To Invoke current class constructor //this();
- To Invoke current class method //this.method_name();

➢ this: to refer current class instance variable:

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

//Example program for this keyword that refers current class 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(1,"abc",5000f);

Student s2=new Student(2,"def",6000f);
s1.display();

s2.display();

}}
```

OUTPUT:

```
   1 abc 5000.0
   2 def 6000.0
```

//Example program to invoke current class method using this

```
class A

{

        void m()

        {

                System.out.println("hello m");

        }

        void n()

        {

                System.out.println("hello n");

                this.m();

        }

        }

class Test

{

        public static void main(String args[])
{

    A a=new A();

    a.n();

}

}
```

OUTPUT:

```
hello n
hello m
```

// invoke current class constructor using this()

```
class Test
{
   int a;
   int b;

   Test()    //Default constructor
   {
      this(10, 20);
      System.out.println("Inside  default constructor \n");
   }

   Test(int a, int b)    //Parameterized constructor
   {
      this.a = a;
      this.b = b;
      System.out.println("Inside parameterized constructor");
   }

   public static void main(String[] args)
   {
      Test object = new Test();
   }
}
```

OUTPUT:

```
            Inside parameterized constructor
            Inside  default constructor
```

<u>Overloading methods and constructors:</u>

Method overloading is a concept that allows to declare multiple methods with same name but different parameters in the same class.

There are two different ways of method overloading:

1. Different datatype of arguments
2. Different number of arguments

// Example program for Method overloading by changing data type of arguments.

```
class Calculate
{
  void sum (int a, int b)
  {
    System.out.println("sum is"+(a+b)) ;
  }
  void sum (float a, float b)
  {
    System.out.println("sum is"+(a+b));
  }
  Public static void main (String[] args)
  {
    Calculate  cal = new Calculate();
    cal.sum (8,5);                    //sum(int a, int b) method is called.
    cal.sum (4.6f, 3.8f);          //sum(float a, float b) method is called.
  }
}
```

<u>OUTPUT:</u>

```
Sum is 13
Sum is 8.4
```

// Example program for Method overloading by changing no. of argument.

```
class MethodOverloading
 {
    void display(int a)
{
        System.out.println("Arguments: " + a);
    }

    void display(int a, int b)
{
        System.out.println("Arguments: " + a + " and " + b);
    }

    public static void main(String[] args)
 {
        display(1);
        display(1, 4);
    }
}
```

OUTPUT:

```
Arguments: 1
Arguments: 1 and 4
```

CONSTRUCTORS

constructor is a block of codes similar to the method. It is a special type of method which is used to initialize the object.

- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- Every time an object is created using the new() keyword, at least one constructor is called.

Rules for creating Java constructor:

1. Constructor name must be the same as its class name.
2. A Constructor must have no explicit return type.
3. A Java constructor cannot be abstract, static, final, and synchronized.

Types of Java constructors:

There are two types of constructors in Java:

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

Default Constructor:
A default constructor is a 0 argument constructor.

Syntax:

```
class Test

{
Test()

{
// constructor body
}
}
```

//Example program for default constructor

```
public class Main
{
  int x;

    public Main()
{
    x = 5;                  // Set the initial value for the class attribute x
  }

  public static void main(String[] args)
{
    Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
```

Output: 5

Constructor Parameters:

- The parameterized constructors are the constructors having a specific number of arguments to be passed.
- A parameterized constructor is written explicitly by a programmer.
- A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors.

Syntax:

```
class Test

{
Test(parameters)

{
// constructor body
}
}
```

//Example program for parameterized constructor

```
public class Main
{
  int x;

  public Main(int y)
{
    x = y;
  }

  public static void main(String[] args)
{
    Main myObj = new Main(5);
    System.out.println(myObj.x);
  }
}
```

Output: 5

Constructor Overloading:

- 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 Student

{

int id;
String name;
int age;

Student(int i,String n)

{
id = i;
name = n;
}

Student(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[])

{
Student s1 = new Student(1,"abc");
Student s2 = new Student(2,"xyz",25);
s1.display();
s2.display();
}
}
```

OUTPUT:

```
1 abc

2 xyz 25
```

Recursion

Recursion is the attribute that allows a method to call itself.
A method that calls itself is said to be recursive.

Syntax:

```
        returntype methodname()

        {

        //code to be executed

        methodname();          //calling same method

        }
```

// The classic example of recursion is the computation of the factorial of a number.

// A simple example of recursion.

```
class Factorial

{


int fact(int n)   // this is a recursive method

{
int result;
if(n==1)

return 1;
result = fact(n-1) * n;

return result;

}
}
class Recursion

{
public static void main(String args[])

{
Factorial f = new Factorial();
System.out.println("Factorial of 3 is " + f.fact(3));
System.out.println("Factorial of 4 is " + f.fact(4));

System.out.println("Factorial of 5 is " + f.fact(5));
}
}
```

<u>OUTPUT:</u>

```
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120
```

// Another example that uses recursion.

```
    public class RecursionExample2

    {

    static int count=0;
    static void p()

    {

    count++;

    if(count<=5)

    {

    System.out.println("hello "+count);

    p();

    }

    }

    public static void main(String[] args)

    {

    p();

    }

    }
```

<u>OUTPUT:</u>

```
hello 1
hello 2
hello 3
hello 4
hello 5
```

<u>GARBAGE COLLECTION</u>

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

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) so we don't need to

make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous 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
```

3) By anonymous object:

```
new Employee();
```

finalize( )

- 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(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc()

- 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()
{
}
```

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Oops concepts

Object oriented programming (Oops) is a methodology that simplifies software development and maintenance by providing some rules.

7. Class
8. Object
9. Inheritance
10. Polymorphism
11. Abstraction
12. Encapsulation

Class:

Class is the collection of objects, class is a user defined blueprint or prototype or template for creating objects, it represents the set of methods, constructors, variables.

<u>Syntax:</u>

```
Accessmodifier class className

{

    Methods

    Constructors

    Fields & variables

}
```

Object:

Object is an instance of class, Object is real world entity, and it occupies memory.

Object consists of

1. State/Attribute
2. Behaviour

How to create an object?

By using new instance() method.

<u>Syntax:</u>

Classname obj/ref variable(that is used to hold the reference of created obj) = new Constructor();