**Class:-**

**In java class is a blue print or template or logical entity of an object.**

**Based on blue print of a class we developed an object.**

**Class is foundation of any java program.**

**In java program we write everything inside the class.**

**Defined class by using class keyword**

**Example:-**

**Class Name**

**keyword**

**class Student**

**{**

**//body of class**

**}**

**java class can contains:-**

1. **Object**
2. **Variable**
3. **Method**
4. **Constructors**
5. **Blocks(instance block and static block)**
6. **Interface and Nested class**

**Object:-**

**Object is physical entity which is created based on class properties.**

**Every object contains three characteristics:-**

**State:-In java programming state represents by instance variable**

**Behaviour:- In java programming behaviour represent by method.**

**Identity:-hashCode is unique identification number for each and every object.**

**Syntax of Object creation by using new keyword:-**

**Class\_Name reference\_variable=new Class\_Name();**

**Example:-**

**Test t=new Test();**

**Test.......class-name**

**t..........reference variable**

**=..........assignment Operator**

**new........keyword used for object creation**

**Test().....constructor call**

**;..........statement terminate**

**Java Variable:-**

**In java, variable means which hold some memory.**

**Variables are also known as fields of a class or properties of a class**

**All variables must be some types its can be primitive types, array types, class types, interface types etc.**

**Syntax Variable Declaration:- We declare a Variable with three components in this orders.**

1. **Zero or more Modifier(optional)**
2. **Variable types(compulsory)**
3. **Variable names(compulsory)**

**Example:-**

**public final int id=101;**

**public final :-Modifier**

**int:- data type**

**id:-variable name.**

**101:-literal(or constant value)**

**;:- statement termination**

**Types of Variable:-Variable divided into two parts.**

**Division-1**

**Based on types of value represented by a variable, all variables are divided into two types.**

1. **Primitive variable.**
2. **Reference variable**

**Primitive variable:-This type variable we have used to represent primitive data value.**

**Example:- int x=101;**

**Reference variable:-This type of variable we have used to represent Object.**

**Example:-**

**Customer cust=new Customer();**

**cust**

**cust is reference variable which is Pointing to the Customer object.**

**Division-2**

**Based on position of declaration and behaviour, all variables are divided into three parts.**

1. **instance variable**
2. **static variable**
3. **local variable**

**Instance Variable:-**

* **If the value of a variable is varied from object to object such types of variables we declared as instance variable.**
* **For every object a separate copy of instance variable will be created**
* **We declared instance variable directly inside the class but outside of the any method or constructor or block.**
* **For the instance variables memory allocated during object creation and memory released when object destroyed. Hence scope of instance variable is exactly same as scope of Object.**
* **It will be stored in heap memory area.**
* **We can access instance variable directly in instance area but can’t be access in static area directly but by using object reference we can access in Static area.**

**There are two types of area in java Language:-**

**A. Instance Area.**

**B. Static Area.**

**Instance Area:-The area which without static keyword such types of area known as instance area.**

**void m1()//instance method**

**{ //logic here //instance Area**

**}**

**Static Area:-The area which with static keyword such types of area known as static area.**

**static void m1()//static method**

**{**

**//logic here//static Area**

**}**

**Example:-**

**class Test**

**{**

**//instance variable**

**int a=20;**

**int b=30;**

**//static method**

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

**{**

**//static area**

**Test t=new Test();**

**System.out.println("static area value of a="+t.a);**

**System.out.println("static area value of a="+t.b);**

**t.m1();**

**}**

**//instance method**

**void m1()**

**{**

**//instance area**

**System.out.println("instance area value of a="+a);**

**System.out.println("instance area value of a="+b);**

**}**

**}**

**Result:-**

**static area value of a=20**

**static area value of a=30**

**instance area value of a=20**

**instance area value of a=30**

**For instance, variable JVM is always providing default value and we have not need to initialize explicitly.**

**Example:-**

**Test.java**

**class Test**

**{**

**int a;**

**double d;**

**boolean b;**

**String s;**

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

**{**

**Test t=new Test();**

**System.out.println(t.a);**

**System.out.println(t.d);**

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

**System.out.println(t.s);**

**}**

**}**

**Result:-**

**0**

**0.0**

**false**

**null**

**It is also known as object level variable attributes**

**Static variable:-**

**If value of the variable is not varied from object to object such types of variables declared as static variable.**

**For every object a single copy of static variable will be created at class level and shared by every object of the class.**

**Static variable will be declared inside the class but outside of the method or block or constructor with static modifier.**

**Static variables memory allocated during .class file loading and memory released at .class file unloading time. So, scope of static variable is exactly same as .class file.**

**Static variable are stored in method area(non-heap memory).**

**We can access static variable either by object reference or by class name but recommended to use to class name. within in the same class it is not required to class name and we can access directly by name.**

**Exercise:-**

**class Test**

**{**

**//static variable**

**static int x=10;**

**//static method**

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

**{**

**//static Area**

**Test t=new Test();**

**System.out.println(t.x);**

**System.out.println(Test.x);**

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

**//all are valid but Test.x is recommended to call static variable**

**//because its reflect the properties of static variable.**

**}**

**}**

**Result:-**

**10**

**10**

**10**

**We can access static variable from both instance and static area directly.**

**Exercise:-**

**class Test**

**{**

**//static variable**

**static int x=10;**

**static int y=20;**

**//static method**

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

**{**

**//static area**

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

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

**Test t=new Test();**

**t.m1();**

**}**

**//instance method**

**void m1()**

**{**

**//instance area**

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

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

**}**

**}**

**Result:-**

**10**

**20**

**10**

**20**

**For static variable JVM provide default value so we have not need to initialize explicitly.**

**Exercise:-**

**class Test**

**{**

**//static variable**

**static int a;**

**static char ch;**

**static double d;**

**static boolean b;**

**public static void main(String[]args)//static method**

**{**

**//static area**

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

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

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

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

**}**

**}**

**Result:-**

**0**

**0.0**

**false**

**It is also known as class level variable**

**Local variable:-**

**Sometime temporary or locally requirement of the variable. such types of variable we declared as local variable.**

**We declared local variable inside a method or block or constructor.**

**Example:-**

**class Test**

**{**

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

**{**

**int a=10;//local variable**

**int b=20; //local variable**

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

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

**}**

**}**

**Result:-**

**10**

**20**

**Local variable:-**

**Local variable will be created while executing the block in which we declare it, once block execution is completes, local variable will be destroyed automatically hence the scope of local variable is the same as block in which we declare it.**

**It is not possible to access local variable out of the block in which we declare it, otherwise we will get compile time error**

**Example:-**

**class Test**

**{**

**void m1()**

**{**

**int x=10;**

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

**}**

**void m2()**

**{**

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

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**t.m2();**

**}**

**}**

**Result:-**

**Test.java:10: error: cannot find symbol**

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

**symbol: variable x**

**location: class Test**

**Local variable will be stored in stack area. hence it is also known as temporary or stack or automatically variables**

**For local variable JVM won’t provide default value, so we should compulsory to performed initialization explicitly, before using that variable, if we are not used that variable not need to initialize explicitly.**

**Exercise:-case-1**

**class Test**

**{**

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

**{**

**//Local area of main method**

**int x;//local variable**

**System.out.println("Hello");**

**}**

**}**

**Result:-Hello**

**Exercise:-case-2**

**class Test**

**{**

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

**{**

**//Local area of main method**

**int x;**

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

**}**

**}**

**Result:-**

**Test.java:6: error: variable x might not have been initialized**

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

**METHOD**

**Method:-**

**Inside the class we can’t write any business logic directly. Inside a class we declare the method and inside that method we write business logic of programming requirement.**

**If we write any business logic directly inside the class then we will get compile time error.**

**Example:-**

**class Test**

**{**

**int x=10;**

**int y=20;**

**System.out.println(x+y);//business logic**

**}**

**To Solve this problem we write all business logic inside the method**

**class Test**

**{**

**int x=10;**

**int y=20;**

**public static void main(String[]args)//static method**

**{**

**Test t=new Test();**

**System.out.println(t.x+t.y);//business logic**

**}**

**}**

**Every method contain three parts**

**1)method declaration.**

**2)method implementations(business logic).**

**3)method calling.**

**Example:-**

**class Test**

**{**

**public void m1()//method declaration**

**{**

**System.out.println("m1()method");//method implementation(business logic)**

**}**

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

**{**

**Test t=new Test();**

**t.m1();//method calling**

**}**

**}**

**Method Syntax:-**

**[Modifier-List] Return-Type Method-Name(parameter List)throws Exception**

**Modifier-List---> Represent access permission------->[optional]**

**Return-Type--->functionality return value------------>[mandatory]**

**Method-Name--->functionality name()---------------->[mandatory]**

**Parameter-List--->input to functionality-------------->[optional]**

**Throws Exception-->represent Exception Handling-->[optional]**

**Example:-**

**Modifier**

**Return-type**

**public void m1(int x)**

**Method name**

**Parameter**

**Method signature:-In java method name and parameter-list considered as method signature.**

**A class can’t contain two method with same signature otherwise we will get compile time error.**

**Method signature:- method name(parameter-list).**

**m1(int a)**

**Method name are same but method signature are different**

**m1(int a,int b)**

**There are two types of method in java**

**1)Instance Method**

**2)Static method**

**Instance Method:-**

**Inside a method if we use at least one instance member directly then that method must be declared as instance method.**

**Instance Method will be invoked at the time of object creation, If we want to access Instance method inside the instance area by using name of the method directly and inside static area by using object reference. it is also known as non-static method.**

**void m1()//instance method/non-static method**

**{**

**//method Body //instance area//non static area**

**}**

**Example:-Access instance method inside the instance area and static area.**

**class Test**

**{**

**public void m1()//instance method**

**{**

**//instance area**

**System.out.println("m1()instance method");**

**}**

**public void m2()//instance method**

**{**

**//instance area**

**m1();**

**}**

**}**

**Example:-Access instance method inside the static area.**

**class Test**

**{**

**public void m1()//instance method**

**{**

**//instance area**

**System.out.println("m1()instance method");**

**}**

**public void m2()//instance method**

**{**

**//instance area**

**m1();**

**}**

**public static void main(String[]args)//static method**

**{**

**//static area.**

**Test t=new Test();**

**t.m2();**

**}**

**}**

**Result:-m1()instance method**

**Static method:-**

**Inside a method if we do not access any instance member directly then that method we declared as static method.**

**Static method will be invoked at the time of .class file load. If we want to access static method inside the instance area or static area by using method name directly but this rule is applicable only for within the class but outside of the class we access by using class name.**

**We can declare static method by using static keyword.**

**static void m1()//static method**

**{**

**//method Body //static area**

**}**

**Example:-Access static variable within the class.**

**class Test**

**{**

**public static void m1()//static method**

**{**

**//static area**

**System.out.println("m1()static method");**

**}**

**public void m2()//instance method**

**{**

**//instance area**

**m1();**

**}**

**public static void main(String[]args)//static method**

**{**

**Test t=new Test();**

**t.m2();**

**m1();**

**}**

**}**

**Result:-**

**m1()static method**

**m1()static method**

**Example:-Access static variable outside of the class.**

**class Test1**

**{**

**static void m1()**

**{**

**System.out.println("m1() static method of Test1 class");**

**}**

**}**

**class Test2**

**{**

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

**{**

**//m1();//12 line**

**Test1.m1();**

**}**

**}**

**Result:-**

**m1() static method of Test1 class**

**Note:-if we are not commenting line 12 then we will get compile time error saying**

**Test.java:12: error: cannot find symbol**

**m1();**

**symbol: method m1()**

**location: class Test2**

**Case1:-**

**Instance and static methods without parameter and at the time of method calling we can’t pass any argument**

**Example:-**

**class Test**

**{**

**public void m1()**

**{**

**System.out.println("m1()instance method");**

**}**

**public static void m2()**

**{**

**System.out.println("m2() static method");**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**Test.m2();**

**}**

**}**

**Result:-**

**m1()instance method**

**m2() static method**

**Case2:-**

**Instance method and static method with parameter**

**At the time of method calling, we should compulsory pass the argument then corresponding method will be execute.**

**While passing parameters, number of argument and order of arguments important.**

**void m1(int a)............t.m1(52);............................valid**

**void m2(int i,char ch,float f)......t.m2(12,’a’,12.32);......invalid**

**void m3(int i,char ch,float f)..........t.m3(10,’v’,12.32f);...valid**

**void m4(int i char ch,float f).......t.m4(10,’v’);...........invalid**

**Example:-**

**class Test**

**{**

**public void m1(int a,char ch)**

**{**

**System.out.println("m1 instance method");**

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

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

**}**

**public static void m2(double d,boolean b)**

**{**

**System.out.println("m2 static method");**

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

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

**}**

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

**{**

**Test t=new Test();**

**t.m1(10,'v');**

**Test.m2(15.31,true);**

**}**

**}**

**Result:-**

**m1 instance method**

**10**

**v**

**m2 static method**

**15.31**

**true**

**Case3:-**

**While calling method we can pass variable as argument**

**Example:-**

**class Test**

**{**

**void m1(int a,boolean b,char ch)**

**{**

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

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

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

**}**

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

**{**

**int a=10;**

**boolean b=true;**

**char ch='v';**

**Test t=new Test();**

**t.m1(a,b,ch);**

**}**

**}**

**Result:-**

**10**

**true**

**v**

**Case4:-**

**we can provide Object as parameter in the method.**

**Example:-**

**class Dog**

**{}**

**class Cat**

**{}**

**class Rat**

**{}**

**class Test**

**{**

**public void m1(Dog d,Cat c)**

**{**

**System.out.println("m1()instance method");**

**}**

**static void m2(int a,Rat r)**

**{**

**System.out.println("m2() static method");**

**}**

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

**{**

**Dog d=new Dog();**

**Cat c=new Cat();**

**Rat r=new Rat();**

**Test t=new Test();**

**new Test().m1(new Dog(),new Cat());**

**t.m1(d,c);**

**t.m2(10,r);**

**Test.m2(10,new Rat());**

**}**

**}**

**Result:-**

**m1()instance method**

**m1()instance method**

**m2() static method**

**m2() static method**

**Case5:-**

**Inside a class we can’t declare two methods with same signature otherwise we will get compile time error.**

**Example:-**

**class Test**

**{**

**void m1(int a)**

**{**

**System.out.println("m1 instance method");**

**}**

**static void m1(int b)**

**{**

**System.out.println("m1 static method");**

**}**

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

**{**

**Test t=new Test();**

**t.m1(10);**

**}**

**}**

**Result:-**

**Test.java:7: error: method m1(int) is already defined in class Test**

**static void m1(int b)**

**Case6:-**

**Declaring a method inside another method is not possible in java method inner concept is not applicable.**

**Example:-**

**class Test**

**{**

**public void m1()**

**{**

**void m2()**

**{**

**System.out.println("m2()inner method");**

**}**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:-**

**Test.java:5: error: illegal start of expressionvoid m2()**

**Test.java:5: error: ';'**

**Expected void m2()**

**Case8:-**

**A method can call any number of method but once method execution is completed the control returns to the caller method.**

**Example:-**

**class Test**

**{**

**public void m1()**

**{**

**m2();**

**System.out.println("m1()mehtod");**

**m2();**

**}**

**public void m2()**

**{**

**m3(100);**

**System.out.println("m2()method");**

**m3(100);**

**}**

**public void m3(int x)**

**{**

**System.out.println("m3()method");**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:-**

**m3()method**

**m2()method**

**m3()method**

**m1()mehtod**

**m3()method**

**m2()method**

**m3()method**

**Method Vs Return type**

**In java method return type is compulsory and void represent nothing return .if we declare a method without return type then we will get compile time error.**

**Example:-**

**class Test**

**{**

**public m1()**

**{**

**System.out.println("m1()method not contain return type");**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:-**

**Test.java:3: error: invalid method declaration; return type required**

**public m1()**

**Method return type can be any type like**

**-primitive types**

**-Array types**

**-class types**

**-Interface types**

**-Enum types**

**If the method return type other than void then method body must contain return the value by using return keyword otherwise we will get compile time error**

**Case1:- A method declare with float return type.**

**Example:-**

**class Test**

**{**

**float m1()**

**{**

**System.out.println("vikas");**

**return 11;**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:-vikas**

**Case2:- A method declare without return type.**

**Example:-**

**class Test**

**{**

**float m1()**

**{**

**System.out.println("vikas");**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:- Test.java:6: error: missing return statement**

**Inside a method we can declare only one return statement and that statement should be last statement otherwise we will get compile time error.**

**Case1:-Return statement is not last statement**

**Example:-**

**class Test**

**{**

**public int m1()**

**{**

**return 10;**

**System.out.println("m1()method");**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:-**

**Test.java:6: error: unreachable statement**

**System.out.println("m1()method");**

**Case2:-Return statement is last the statement of the method body otherwise we will get compile time error.**

**class Test**

**{**

**public int m1()**

**{**

**System.out.println("m1()method");**

**return 10;**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**

**Result:- m1()method**

**Every method is able to return the value but holding that return value is optional, but it is recommended to hold the return value check the status of the method.**

**Example:-**

**class Test**

**{**

**public boolean m1(int x,char ch)**

**{**

**System.out.println("m2() method");**

**System.out.println(x+".."+ch);**

**return true;**

**}**

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

**{**

**Test t=new Test();**

**boolean b=t.m1(10,'v');**

**System.out.println("m1()method return values="+b);**

**}**

**}**

**Result:-**

**m2() method**

**10..v**

**m1()method return values=true**

**Print the return type values of the method by two ways.**

1. **Hold the value and then print that value**
2. **Directly print call method by using System.out.println() ,this approach is only applicable if method return type must be not void otherwise we will get compile time error.**

**Example:-**

**class Test**

**{**

**int m1()**

**{**

**System.out.println("m1()method");**

**return 7;**

**}**

**void m2()**

**{**

**System.out.println("m2()method");**

**}**

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

**{**

**Test t=new Test();**

**int x=t.m1();**

**System.out.println("return value of m1()method is="+x);**

**System.out.println("return value of m1()method is="+t.m1());**

**t.m2();**

**//System.out.println(t.m2());//line:-19**

**}**

**}**

**Result:-**

**m1()method**

**return value of m1()method is=7**

**m1()method**

**return value of m1()method is=7**

**m2()method**

**Note:-If we remove comment form line 19 then we will get compile time error Saying:-**

**Test.java:19: error: 'void' type not allowed here**

**System.out.println(t.m2());**

**class Test**

**{**

**void m1(int age)**

**{**

**if(age>=18)**

**{**

**System.out.println("you are eligible for voting");**

**}**

**else**

**{**

**System.out.println("you are not eligible for gc");**

**}**

**}**

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

**{**

**Test t=new Test();**

**t.m1(18);**

**}**

**}**

**Result:-**

**you are eligible for voting**

**A java class is able to return user defined class as a return type.**

**Example:-**

**class X**

**{}**

**class Emp**

**{}**

**class Test**

**{**

**public X m1()**

**{**

**System.out.println("m1()method");**

**X x=new X();**

**return x;**

**}**

**public Emp m2()**

**{**

**System.out.println("m2()method");**

**Emp e=new Emp();**

**return e;**

**}**

**public static String m3()**

**{**

**System.out.println("m3()method");**

**return "vikas";**

**}**

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

**{**

**Test t=new Test();**

**X x1=t.m1();**

**System.out.println(x1);**

**Emp e=t.m2();**

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

**String s=Test.m3();**

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

**}**

**}**

**Result:-**

**m1()method**

**X@15db9742**

**m2()method**

**Emp@6d06d69c**

**m3()method**

**vikas**

**Template Method:-**

**In general, after complete the task we required to perform all the method calling, so at the time of method calling we should remember two things.**

1. **Number of method and name of method.**
2. **Order of method(which one is first and which one is later).**

**Overcome this problem we used Template Method, Template Method is normal method inside this we perform calling all method and based on our requirement we calling only Template Method instead all method one by one.**

**Example:-**

**//Template method**

**class Test**

**{**

**void customer()**

**{**

**System.out.println("customer");**

**}**

**void product()**

**{**

**System.out.println("product");**

**}**

**void selection()**

**{**

**System.out.println("selection");**

**}**

**void billing()**

**{**

**System.out.println("billing");**

**}**

**void deliveryManager()**

**{**

**customer();**

**product();**

**selection();**

**billing();**

**}**

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

**{**

**Test t=new Test();**

**t.deliveryManager();**

**}**

**}**

**Result:-**

**customer**

**product**

**selection**

**billing**

**Method recursion :-A method calling itself during execution is called recursion.**

**Example:-**

**class Test**

**{**

**public void m1()**

**{**

**System.out.println("m1() method");**

**m1();**

**}**

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

**{**

**Test t=new Test();**

**t.m1();**

**}**

**}**