Training On Java

Lecture – 5
Methods & OOPS Concepts

Method In Java

- Methods are used to write the business logics of the project.
- Coding conversion of method is method name starts with lower case letter if method contains more than One word then every inner word starts with uppercase letter. Example:- post() , charAt() , toUpperCase() , compareToIgnoreCase()……etc
- There are two types of methods **1. Instance method 2. Static method**
- Inside the class it is possible to declare any number of instance methods & static methods based on the developer requirement.
- It will improve the reusability of the code and we can optimize the code.

Method In Java (cont..)

Instance Method:- The instance method is also called non-static method. These methods are declared without using static modifier. These methods can call by using the object of class. Without using object we can't call instance method.

Static Method:- The static methods are declared by using static keyword. These method are also called class methods. There is no need of object to call static methods

Syntax of declaration of method in Java:-

<modifier> <return_type> method_name(parameters)
{
//code
}
E.g.

```
static int add(int x, int y)
{
return (a+b);
}
```

Example Application -1

```
//Find the volume of cuboid using method
import java.util.*;
class MethodDemo1 {
public static void main(String [] args){
int l,b,h,v;
Scanner sc=new Scanner(System.in);
System.out.println(“Enter length, breadth and height of cuboid”);
l=sc.nextInt();
b=sc.nextInt();
h=sc.nextInt();
v=volume(l,b,h);
System.out.println(“Volume of cuboid=”+v);
}
static int volume(int x,int y,int z){
return (x*y*z);
}
}
```

Recursion

When a function call itself then it is called ’Recursion ’. For example To find the factorial of given number:-

5!=5*4*3*2*1

5!=5*4!

.

.

n!=n*(n-1)!

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

Example Application - 2

```
//Develop a program to find factorial of given number using ‘Recursion’.
import java.util.*;
class Test {
static long fact(int n) {
if (n==0 || n==1)
return 1;
else
return n*fact(n-1);
}
public static void main(String [] args){
Scanner sc=new Scanner(System.in);
System.out.print(“Enter the number to find factorial : ”);
int n=sc.nextInt();
System.out.println(“Factorial=”+fact(n));
}
}
```

O/P:- Enter the number to find factorial : 5
Factorial=120

.

Object Oriented Programming System

Object Oriented Programming System (cont..)

The Object Oriented Programming System is a mechanism of Software Development. It have four concepts:-

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Any Programming Language which follows these concepts is called Object Oriented Programming Language.

.

Abstraction In Java

Abstraction is selecting data from a larger pool to show only the relevant details to the object. It helps to reduce programming complexity and effort. In Java, abstraction is accomplished using Abstract classes and interfaces. It is one of the most important concepts of OOPs. Abstraction in Java can be achieved using Abstract Class and Abstract Method.

Abstract Class:- A class which is declared “abstract” is called as an abstract class. It can have abstract methods as well as concrete methods. A normal class cannot have abstract methods.

Abstract Methods:- A method without a body is known as an Abstract Method. It must be declared in an abstract class. The abstract method will never be final because the abstract class must implement all the abstract methods.

.

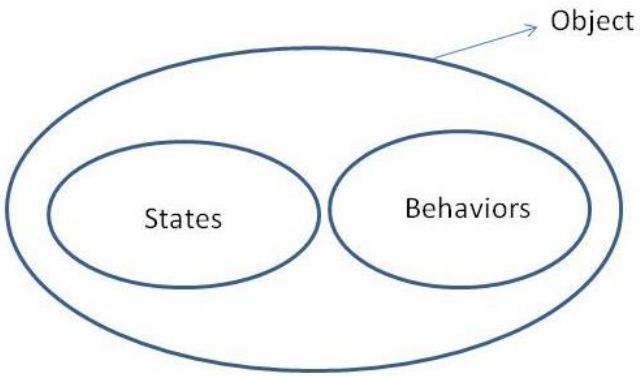
.

Encapsulation In Java

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

.

.

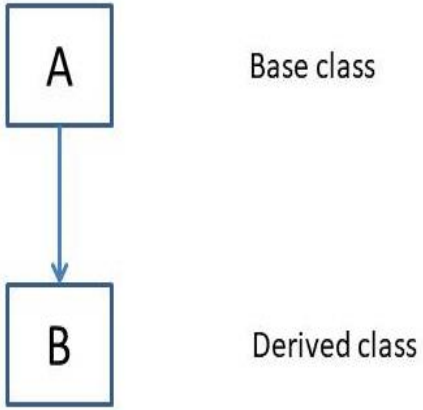


Inheritance In Java

The Inheritance is a very nice feature of Object oriented programming . In Inheritance we can create a new class by using existing class. The existing class is called base class and new created class is called derived class. The concept of Inheritance is also called “Reusability”.

.

.



Polymorphism In Java

The term “Polymorphism” means one thing many forms. There are two types of Polymorphism.

Compile Time Polymorphism:- The compile time polymorphism contains the concept of overloading. In java there is a concept of method overloading. In method overloading the method name is same but parameters are different. Based on method parameters it is decided at compilation time that which method call from where.

Run Time Polymorphism:- The run time polymorphism contains the concept of overriding. In java there is a concept of method overriding. The re-writing of base class method to derived class is called ‘Method Overriding’.

.

.

Concept Of Class

The Class is a container of variables, methods and constructors. Or The Class contains data members and member functions. The class is declared by using **“class”** keyword followed by class name. The body of class is enclosed within braces and terminated by semicolon (Optional in Java).

Declaration of Class:-

```
class class_name
{
//Body of class
};
.
.
```

Example Application -3

```
//Use of public access specifier
class car {
public String make; //Data member
public String color; //Data member
public int price;      //Data member
}
class Test {
public static void main(String [] args) {
Car c;                  //Reference variable of class Car
c=new Car();          //Creation of object of class Car
c.make=”Tata”;
c.color=”Silver”;
c.price=650000;
System.out.println(“Car Make-> ”+c.make);
System.out.println(“Car Color-> ”+c.color);
System.out.println(“Car Price-> ”+c.price);
}
}
```

.

.

Example Application -4

```
//Use of private access specifier
class Car
{
private String make;                //private data member
private String color;//private data member
private int price;      //private data member
public void setCar (String make, String color, int price)
{
this.make=make;
this.color=color;
this.price=price;
}
public void display()
{
System.out.println(“Car make-> ”+this.make);
System.out.println(“Car color-> ”+this.color);
System.out.println(“Car price-> ”+this.price);
}
};
```

.

.

Example Application -4 (cont..)

```
class Test
{
public static void main(String [] args)
{
Car c=new Car();
c.setCar(“Tata”,”Silver”,650000);
c.display();
}
}
```

O/P:-

Car make->Tata
Car color->Silver
Car Price->650000

Note:- The private data members are not directly accessible outside of class. These are accessible via public member functions.

.

.

Concept Of Constructor

The Constructor is a special member function which is used to initialize final data members.

Rules to declare constructor:-

1) Constructor name class name must be same.
2) Constructor is able to take parameters.
3) Constructor not allowed explicit return type (return type declaration not possible).

There are two types of constructors:-

1) Default Constructor (provided by compiler).
2) User defined Constructor (provided by user) or parameterized constructor.

.

.

Default Constructor

1) If we are not write constructor for a class then compiler generates one constructor for you that constructor is called default constructor. And it is not visible in code.

2) Compiler generates Default constructor inside the class when we are not providing any type of constructor (0-arg or parameterized).

3) The compiler generated default constructor is always 0-argumnetconstructor with empty implementation (empty body).

.

.

Application Before Compilation

```
class Test
{
void m1()
{
System.out.println("m1 method");
}
public static void main(String[] args)
{
//at object creation time 0-arg constructor executed
Test t = new Test();   t.m1();
}
}
```

In above application when we create object by using new keyword **“Test t = new Test()”** then compiler is searching **“Test()”**constructor inside the since not there hence compiler generate default constructor at the time of compilation.

.

.

Application After Compilation

```
class Test
{
void m1()
{
System.out.println("m1 method");
}
//default constructor generated by compiler
Test()
{
}
public static void main(String[] args)
{
//object creation time 0-arg constructor executed
Test t = new Test();
 t.m1();
}
}
```

.

.

Example Application - 5

```
class Employee  {
//instance variables
int eid;
String ename;
double esal;
Employee(int eid,String ename,double esal) //local variables
{
//conversion (passing local values to instance values)
this.eid = eid;
this.ename = ename;
this.esal = esal;
}
void display()  {
//printing instance variables values
System.out.println("****Employee details****");
System.out.println("Employee name :-->"+ename);
System.out.println("Employee eid :-->"+eid);
System.out.println("Employee esal :-->"+esal);
}
```

.

.

Example Application – 5 (cont..)

```
public static void main(String[] args)
{
// during object creation parameterized  constructor executed
Employee e1 = new Employee(111,"Brijesh",60000);
e1.display();
Employee e2 = new Employee(222,"Rohit",70000);
e2.display();
Employee e3 = new Employee(333,"Yashi",80000);
e3.display();
}
}
```

Output

****Employee details****
Employee name :-->Brijesh
Employee eid :-->111
Employee esal :-->60000.0
****Employee details****
Employee name :-->Rohit
Employee eid :-->222
Employee esal :-->70000.0
****Employee details****
Employee name :-->Yashi
Employee eid :-->333
Employee esal :-->80000.0