Method Overloading - Compile Time Polymorphism

Method Overloading implements compile-time polymorhism. In a class more than one method can have same method name with different arguments. All the methods arguments distinct in three ways.

1. **The number of arguments should be distinct.**
2. **Types of argument should be distinct.**
3. **Sequence of argument should be distinct.**

Program

```
	class Overload
	{
		private int x,y,z,s;
		private double d1,d2,d3;
		public int sum(int a,int b)
		{	x=a;
			y=b;
			s=x+y;
			return(s);
		}
		public int sum(int a,int b,int c)
		{
			x=a;
			y=b;
			z=c
			s=x+y+z;
			return(s);
		}
		public double sum(double a, double b)
		{
			d1=a;
			d2=b;
			d3=d1+d2;
			return(d3);
		}
		public double sum(int a, double b)
		{
			x=a;
			d1=b;
			d3=x+d1;
			return(d3);
		}
		public double sum(double a, int b)
		{
			d1=a;
			x=b;
			d3=x+d1;
			return(d3);
		}
}
class MethodOverDemo
{
	public static void main(String args[])
	{
		Overload obj = new Overload();
		System.out.println("Sum of double and int "+obj.sum(20.4,30));
		System.out.println("Sum of two int "+obj.sum(10,20));
		System.out.println("Sum of three int "+obj.sum(10,20,30));
	}
}
```

OUTPUT

```
Sum of double and int 50.4
Sum of two int 20
Sum of three int 60
```

7.6 Static Modifier

The static modifier can be applied to **variables, methods** and **block of code** (that is not a part of a method). Static features are associated with a class rather than being associated with an individual instance of a class

7.6.1 Static Variables (Class Variables)

To access the static variable, can use class name instead of using its object, means that there is only one variable, no matter how many instances of class might exist at any particular moment.

Syntax `access-specifier static <datatype> <identifier>;`

There are two way to refer a static variable:

1. Via a reference to any instance(object) of the class
2. Via the classname.

Program

```
class Static
{
      public static int a;//class variable
}
class StaticDemo
{
      public static void main(String args[])
      {
            Static s1 = new Static();
            Static s1 = new Static();
            s1.a=10;
            s2.a=20;
            System.out.println("S1.a = "+s1.a);
            System.out.println("S2.a = "+s2.a);
            Static.a=100; //invoking through classname
            System.out.println("S1.a = "+s1.a);
            System.out.println("S2.a = "+s2.a);
            System.out.println("Static.a = "+Static.a);
      }
}
```

OUTPUT

```
S1.a = 20
S2.a = 20
S1.a = 100
S2.a = 100
Static = 100
```

7.6.2 Static Methods (Class Methods)

Class methods, like class variables, are available to instance of the class and can be made available to other classes. Class method can be used anywhere regardless of whether an instance of the class exist or not.

Syntax

```
static <return type> <methodname>( type arg1, type arg2, …....)
{
<set of statements>
}
```

Methods declared as static have several restrictions

- They can only call other **static** methods
- They must only access **static** data.
- They cannot refer to **this** or **super** in any way.

Program

```
class UseStatic
{
      static int a=3;
      static int b;
      static void display(int x)
      {
            System.out.println (" X = " + x);
            System.out.println (" a = " + a);
            System.out.println (" b = " + b);
      }
      static  //static block
      {
            System.out.println ("Static block initialized");
            b=a*4;
      }

      public static void main(String args[])
      {
            display(42);
      }
}
```

OUTPUT

```
Static block initialised
X = 42
a = 3
b = 12
```

As soon as the UseStatic class is loaded, all of the static statements are run. First **a** is set to **3** , then the static block executes (printing a message), and finally, b is initialized to **a*4 or 12**. the **main()** is called, which calls **display(),** passing **42 to x.**

The static keyword indicates that it is a class method and can be accessed without help of an object.

7.7 Class members vs Instance Members

Class members (static)	Instance members (non-static)
A class methods or class variables is associated with a particular class. The runtime system allocates a class variable once per class, no matter how many instances exist of that class. You access class variable and methods through the class name.	An instance methods or instance variables is associated with a particular object (an instance of a class). Every time you create an object, the new object gets a copy of every instance variable defined in its class. You access instance variable and methods through its objects.

7.8 Constructors

A class is a blueprint, which stores a set of properties(variable and constants) and methods. If the properties and methods of a class are to be initailized, hence a class needs a method. These methods are known as constructors.

A Constructor is a special kind method that determines how an object is initialized when created.

Instantiating an object of a class does two things

- allocate memory
- calls the constructor of the class

Rules for constructing a Constructor

1. Constructor name and class name must be **identical.**
2. Constructor should be **not be private**
3. Constructor has **no return type**. But it may be have arguments.

When Constructors are called

When we create an object of the class, constructor will be invoke automatically.

When the keyword **new** is used to create an instance of class, java allocates memory for the object, initializes the instance variable and **calls the constructor methods.**

Note Constructors can also be **Overloaded.**

7.8.1 Default Constructors

Every class in java by default has a **default constructor** that does not take any argument and the body of it does not have any statements.

7.8.2 Constructor without argruments

Program

```
class Box
{
	private double width,height,depth; //instances variables
	public Box()//constructor without arguments
	{
			width=5;
			height=5;
			depth=5;
	}
	public double volume()//instance method
	{
		double vol= width*height*depth;
		return vol;
	}
}
class BoxDemo
{
	public static void main(String args[])
	{
```

```
            Box b = new Box();
            double vol=b.volume();
            System.out.println("Volume of box is "+vol);
      }
}
```

OUTPUT `Volume of box is 125`

7.8.3 Parameterised Constructors

The constructors that can take arguments are called parameterized constructors. We must pass the initial values as arguments to the constructors when an object is declared.

Program

```
class Box
{
      private double width,height,depth;
      public Box(double w, double h, double d)
      {
            width = w;
            height = h;
            depth = d;
       }
      public double volume()//instance method
      {
            double vol= width*height*depth;
            return vol;
      }
}
class BoxDemo
{
      public static void main(String args[])
      {
            Box b = new Box(3.5,4.5,5.5);
            double vol=b.volume();
            System.out.println("Volume of box is "+vol);

      }
}
```

OUTPUT `Volume of box is 125`

7.8.4 Overloading Constructors

Constructors can also take varying numbers and types of parameters. This enables creation of objects with the properties required.

Program

```
/* Here, Box defines three constructors to initialize
   the dimensions of a box various ways.
*/
class Box
{
  private double width;
  pirvate double height;
```

```
  private  double depth;


// constructor used when all dimensions specified

      public Box(double w, double h, double d)
      {
            width = w;
            height = h;
            depth = d;
      }

  // constructor used when no dimensions specified

      Box()
      {
          width = 5;
          height = 5;
          depth = 5;
      }

  // constructor used when cube is created

        Box(double len)
        {
          width = height = depth = len;
        }

  // compute and return volume
       double volume()
       {
          return width * height * depth;
       }
}

class BoxDemo
{
  public static void main(String args[])
{
    // create boxes using the various constructors
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box();
    Box mycube = new Box(7);

    double vol;

    // get volume of first box
    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);

    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);

    // get volume of cube
    vol = mycube.volume();
    System.out.println("Volume of mycube is " + vol);
  }
}
```

OUTPUT

```
Volume of mybox1 is 3000
```

```
Volume of mybox1 is 125
Volume of mycube is 343
```

7.9 Using object as Parameters

So far we have only been using simple data types as parameters to methods. However, it is both correct and common to pass object to methods. For example, consider the following simple program

Program

```
// Objects may be passed to methods.
class Test
{
  int a, b;

       Test(int i, int j)
       {
              a = i;
              b = j;
       }

  // return true if o is equal to the invoking object

       boolean equals(Test o)
       {
       System.out.println("o.a   = "+ o.a + "a   = " + a);
       System.out.println("o.b   = "+ o.b + "b   = " + b);

       if(o.a == a && o.b == b)
             return true;
       else
             return false;
  }
}

class PassOb
{
  public static void main(String args[])
 {
    Test ob1 = new Test(100, 22);
    Test ob2 = new Test(100, 22);
    Test ob3 = new Test(-1, 22);

    System.out.println("ob1 == ob2: " + ob1.equals(ob2));

    System.out.println("ob1 == ob3: " + ob1.equals(ob3));
  }
}
```

7.10 Returning object

A method can return any type of data, including class types that you create. For example, in the following program, the **incrByTen()** method returns an object in which the value of a is ten greater than it is in the invoking object.

Program

```
// Returning an object.
class Test
{
  int a;

  Test(int i)
  {
    a = i;
  }

  Test incrByTen()
 {
    Test temp = new Test(a+10);
    return temp;
  }
}

class RetOb
{
  public static void main(String args[])
 {
    Test ob1 = new Test(2);
    Test ob2;

    ob2 = ob1.incrByTen();
    System.out.println("ob1.a: " + ob1.a);
    System.out.println("ob2.a: " + ob2.a);

    ob2 = ob2.incrByTen();
    System.out.println("ob2.a after second increase: "
                       + ob2.a);
  }
}
```

8. Inheritance

Inheritance provides the idea of Reusability

Inheritance is a process by which objects of one class acquire the properties of objects from another class.

It is a process of creating new class from an existing class. The existing class is called as **Base class or super class.** The newly created class is called as **subclass or derived class.**

Declaring a Sub-class

Syntax

```
class <sub-class-name> extends <super-class-name>
{
      instance variable;
      +
      methods();
}
```

Types of Inheritance

1. Single Inheritance
2. MultiLevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance

8.1 Single Inheritance

Process of creating a subclass from a super class.

Program

```
class A
{
      public int x,y;
      public int sum(int a,int b)
      {
            x=a;
            y=b;
            return(x+y);
      }
}
class B extends A
{
      public int mul(int a, int b)
      {
            x=a;
            y=b;
            return(x*y);
      }
}
class Single
```

```
{
	public static void main(String args[])
	{
		B obj=new B();
		System.out.println("Sum of two nos. is "+obj.sum(20,10));
		System.out.println("Product of two nos. is "+obj.mul(20,10));
	}
}
```

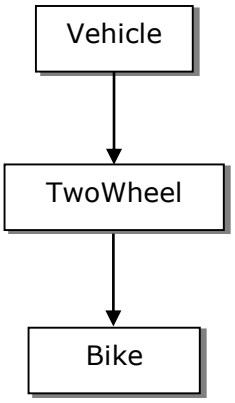
Using object of derived class can access all the properties of super class.

OUTPUT

```
Sum of two nos. is 30
Product of two nos. is 200
```

8.2 Multilevel Inheritance

It is a process of creating a sub class from another subclass.



Program

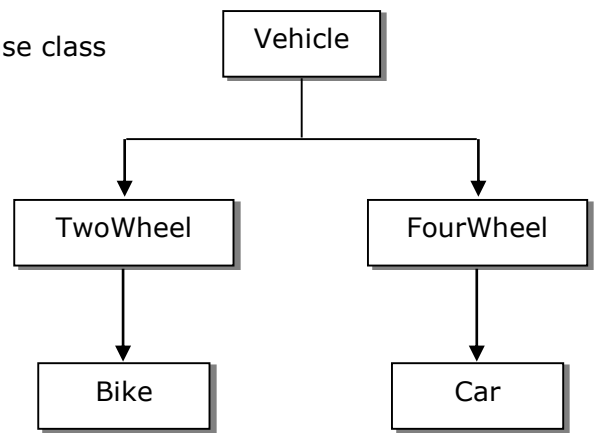
```
import java.util.Scanner;
class Vehicle
{
	String regNo;
	int model;
	Scanner s;
	void readVehicle()
	{
		System.out.println("\n Enter the RegNo and Model ");
		s =new Scanner(System.in);
		regNo=s.next();
		model=s.nextInt();
	}
	void printVehicle()
	{
		System.out.println("\n Registeration No.    : "+regNo);
		System.out.println("\n Model               : "+model);
	}
}
class TwoWheel extends Vehicle
{
	int noGear;
	int power;
	void readTwo()
	{
		readVehicle();
		System.out.println("\n Enter the No. of Gear and Power");
		noGear=s.nextInt();
```

```
            power=s.nextInt();
      }
    void printTwo()
     {
           printVehicle();
           System.out.println("\nNo. of Grear      :"+noGear);
           System.out.println("\nPower             :"+power);
     }
}
class Bike extends TwoWheel
{
      String manufacture;
      String owner;
      void readBike()
      {
            readTwo();
            System.out.println("\n Enter Manufacture and Owner Name\n");
            manufacture=s.next();
            owner=s.next();
      }
      void printBike()
      {
           printTwo();
           System.out.println("\nManufacturer  :"+manufacture);
           System.out.println("\nOwner         :"+owner);
      }
}

class Multilevel
{
      public static void main(String args[])
      {
            Bike s1 = new Bike();
            s1.readBike();
            s1.printBike();
      }
}
```

8.3 Hierarchical Inheritance

Creating several derived classes from one base class



Program

```
import java.util.Scanner;
class Vehicle
```

```
{
	String regNo;
	int model;
	Scanner s;
	void readVehicle()
	{
		System.out.println("\nEnter the Register No. and Model ");
		s =new Scanner(System.in);
		regNo=s.next();
		model=s.nextInt();
	}
	void printVehicle()
	{
		System.out.println("\n Registration No.	: "+regNo);
		System.out.println("\n Model		: "+model);
	}
}
class TwoWheel extends Vehicle
{
	int noGear;
	int power;
	void readTwo()
	{
		readVehicle();
		System.out.println("\n Enter the Number of Gear and Power");
		noGear=s.nextInt();
		power=s.nextInt();
	}
	void printTwo()
	{
	     printVehicle();
	     System.out.println("\nNo.of Grear	:"+noGear);
	     System.out.println("\nPower		:"+power);
	}
}
class Bike extends TwoWheel
{
	String manufacture;
	String owner;
	void readBike()
	{
```

```
            readTwo();
            System.out.println("\n Enter Manufacture and Owner Name\n");
            manufacture=s.next();
            owner=s.next();
      }
      void printBike()
      {
            printTwo();
            System.out.println("\nManufacturer  :"+manufacture);
            System.out.println("\nOwner         :"+owner);
      }
}
class FourWheel extends  Vehicle
{
      String fuel;
      int noOfCylinder;
      void readFour()
      {
            readVehicle();
            System.out.println("\nGive the type of Fuel and No. of Cylinder ");
            fuel=s.next();
            noOfCylinder=s.nextInt();
      }
      void printFour()
      {
            printVehicle();
            System.out.println("\nFuel type =       : "+fuel);
            System.out.println("\nNo. of Cylinder   : "+noOfCylinder);
      }
}
class Car extends FourWheel
{
      String name;
      String owner;
      void readCar()
      {
      readFour();
      System.out.println("\nEnter name of the Car and Owner Name ");
      name=s.next();
      owner=s.next();
      }
```

```
	void printCar()
	{
	printFour();
	System.out.println("\n Name of the car      " +name);
	System.out.println("\n Name of the Owner    " +owner);
	}
}
class Hierarchy
{
	public static void main(String arg[])
	{
		Scanner s= new Scanner(System.in);
		System.out.println("\n\tMain Menu");
		System.out.println("\n\t*********");
		System.out.println("\n 1. Two Wheeler");
		System.out.println("\n 2. Four wheeler ");
		System.out.println("\n Enter your choice : ") ;
		int ch=s.nextInt();
		if(ch==1)
		{
		Bike s1 = new Bike();
		s1.readBike();
		s1.printBike();
		}
		else if(ch==2)
		{
		Car c1=new Car();
		c1.readCar();
		c1.printCar();
		}
		else
		System.out.println("Enter Correct Choice ....." );
	}
}
```

Output

```
		Main Menu
		*********
1. Two Wheeler
2. Four wheeler
Enter your choice : 1
Enter the Register No. and Model    TN23A1234 2010
Enter the Number of Gear and Power  5     150
Enter Manufacture and Owner Name    Honda Raj
```

```
Registration No.        : TN23A1234
Model                   : 2010
No.of Grear             : 5
Power                   : 150
Manufacturer            : Honda
Owner                   : Raj
```

Note

Java does not support Multiple Inheritance through class itself
Java offers the advantages of Multiple Inheritance through a feature called Interfaces

8.3 Constructors in Derived Classes

Constructors are invoke in the order of Inheritances

Program

```
class A
{
      A()
      {
      System.out.println("A is called ");
      }
}
class B extends A
{
      B()
      {

      System.out.println("B is called ");
      }
}

class C extends B
{
      C()
      {
      System.out.println("C is called ");
      }
}
class CallConstructor
{
      public static void main(String arg[])
      {
      C c = new C();
      }
}
```

OUTPUT

```
A is called
B is called
C is called
```

8.4 `Super` Keyword

It allows a subclass to refer to its immediate superclass members.

The `super` keyword is used to call variables, methods and constructors of superclass

Syntax

```
super.data          //call superclass variable / constants
super.method();     //call superclass method
super();            //call superclass constructor
```

1. Super Class Variable

Program

```
//Invoking super class variable from sub class
```

```
class Main
{
	int i=12;
}
class Sub extends Main
{
	int i=17;

	void display()
	{
	System.out.println("Super I = "+super.i);
	System.out.println("Sub I = "+i);
	}
}
class SuperVariable
{
	public static void main(String arg[])
	{
		Sub s = new Sub();
		s.display();
	}
}
```

Output

```
Super I = 12
Sub I = 17
```

2. Super Class Method

Program

```
//Invoking super class method from sub class

class Main
{
	int i=12;

	void display()
	{
		System.out.println("Super I = "+i);
	}
}

class Sub extends Main
{
int i=17;


	void display()
	{
	super.display();
	System.out.println("Sub I = "+i);
	}
}
class SuperMethod
{
	public static void main(String arg[])
	{
		Sub s = new Sub();
		s.display();
}
```

```
}
```

Output

```
Super I = 12
Sub I = 17
```

3. Super Class Constructor

Program

```
//Invoking Super class Constructor using super keyword
class Main
{
int i;
int j;
      Main(int i, int j) //super class constructor
      {
      i=i;
      j=j;
      }
}

class Sub extends Main
{
int k;
      Sub(int x, int y, int z) //sub class constructor
      {
      k=z;
      super(x,y);  //invokes Main(x,y) constructor

      }
      void display()
      {
      System.out.println("I = "+i+"\nJ = "+j);
      System.out.println("K = "+k);
      }
}
class SuperConstructor
{
      public static void main(String arg[])
      {
            Sub s = new Sub(10,20,30);
            s.display();
      }
}
```

Output

```
I = 10
J = 20
K = 30
```

8.5 `this` keyword

this **refers to current object**

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the **this** keyword. **this** can be used inside any method to **refer to the *current* object.** That is, **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class' type is permitted.

Instance Variable Hiding

As you know, it is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes. Interestingly, you can have local variables, including formal parameters to methods, which overlap with the names of the class' instance variables.

However, when a local variable has the same name as an instance variable, the local variable *hides* the instance variable.

Note `this` cannot be refered by static methods

Program

```
class This
{
      private int a,b,c;
      public void sum(int a, int b)
      {
            this.a=a;
            this.b=b;
            this.c=this.a+this.b
            this.display();
      }
      public void display()
      {
            System.out.println("Value of A is "+a);
            System.out.println("Value of B is "+b);
            System.out.println("Sum of two nos. is "+c);
      }
}
class ThisDemo
{
      public static void main(String args[])
      {
            This t1= new This( ) ;
            This t2= new This( ) ;
            t1.sum(10,20);
            t2.sum(50,60);
      }
}
```

OUTPUT

```
Value of A is 10
Value of B is 20
Sum of two nos. is 30
Value of A is 50
Value of B is 60
Sum of two nos. is 110
```

8.6 Method Overriding - RunTime Polymorhism

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden.

Rules

- The prototypes of the base class version method and all the derived class versions must be identical.
- They are accessed by using object reference.
- They cannot be static members.

```
class A
{
```

```
        public int arith(int a,int b);
        {
              return(a+b);
        }

  }
  class B extends A
  {
        public int arith(int a,int b)
        {
              return(a-b);
        }
  }
  class C extends B
  {
        public int arith(int a, int b)
        {
              return(a*b);
        }
  }
  class D extends C
  {
        public int arith(int a,int b)
        {
              return(a/b);
        }
  }

class Overriding
{
      public static void main(String args[])
      {
      A p1;  // reference variable for base class
      p1=new B();
      System.out.println("Subtraction of two number is "+p1.arith(20,10));
      p1=new C();
      System.out.println("Product of two nos is "+p1.arith(20,10));
      p1=new D();
      System.out.println("Division of two nos is "+p1.arith(20,10));
      p1=new A();
      System.out.println("Addition of two nos is "+p1.arith(20,10));
      }
}
```

OUTPUT

```
Subtraction of two number is 10
Product of two number is 200
Division of two number is 2
Addition of two number is 30
```