Object Oriented Programming

Method Overloading & Overriding, Polymorphism

Compiled By: Umair Yaqub

Method Overloading in Java:

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. **Method overloading** is also known as **Static Polymorphism**.

If we have to perform only one operation, having same name of the methods increases the readability of the program. Advantage of method overloading

Method overloading *increases the readability of the program.*

1. **Example 1: Overloading – Different Number of parameters in argument list**
   When methods name are same but number of arguments are different.

```
class DisplayOverloading
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
class Sample
{
  public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     obj.disp('a');
     obj.disp('a',10);
  }
}
```

Output: a

a 10

Example 2: Overloading – Difference in data type of arguments

In this example, method disp() is overloaded based on the data type of arguments – Like example 1 here also, we have two definition of method disp(), one with char argument and another with int argument.

```
class DisplayOverloading2
{
   public void disp(char c)
   {
      System.out.println(c);
   }
   public void disp(int c)
   {
     System.out.println(c );
   }
}

class Sample2
{
   public static void main(String args[])
   {
      DisplayOverloading2 obj = new DisplayOverloading2();
      obj.disp('a');
      obj.disp(5);
   }
}
```

Output: a

5

Example3: Overloading – Sequence of data type of arguments

Here method disp() is overloaded based on sequence of data type of arguments – Both the methods have different sequence of data type in argument list. First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

```
class DisplayOverloading3
{
  public void disp(char c, int num)
  {
     System.out.println("I'm the first definition of method disp");
  }
  public void disp(int num, char c)
```

```
  {
     System.out.println("I'm the second definition of method disp" );
  }
}
class Sample3
{
  public static void main(String args[])
  {
     DisplayOverloading3 obj = new DisplayOverloading3();
     obj.disp('x', 51 );
     obj.disp(52, 'y');
  }
}
```

Output: I'm the first definition of method disp
I'm the second definition of method disp

//

Method Overriding in Java:

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**. In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Example:

One of the simplest example – Here Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the method eat().

```
class Human{
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
```

```
class Boy extends Human{
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    obj.eat();
  }
}
```

Output: Boy is eating

Method Overriding in dynamic method dispatch

Dynamic method dispatch is a technique which enables us to assign the base class reference to a child class object. As you can see in the below example that the base class reference is assigned to child class object.

```
class ABC{
  public void disp()
  {
    System.out.println("disp() method of parent class");
  }
  public void abc()
  {
    System.out.println("abc() method of parent class");
  }
}
class Test extends ABC{
  public void disp(){
    System.out.println("disp() method of Child class");
  }
  public void xyz(){
    System.out.println("xyz() method of Child class");
  }
  public static void main( String args[]) {
    //Parent class reference to child class object
    ABC obj = new Test();
    obj.disp();
    obj.abc();
  }
}
```

Output: disp() method of Child class
abc() method of parent class

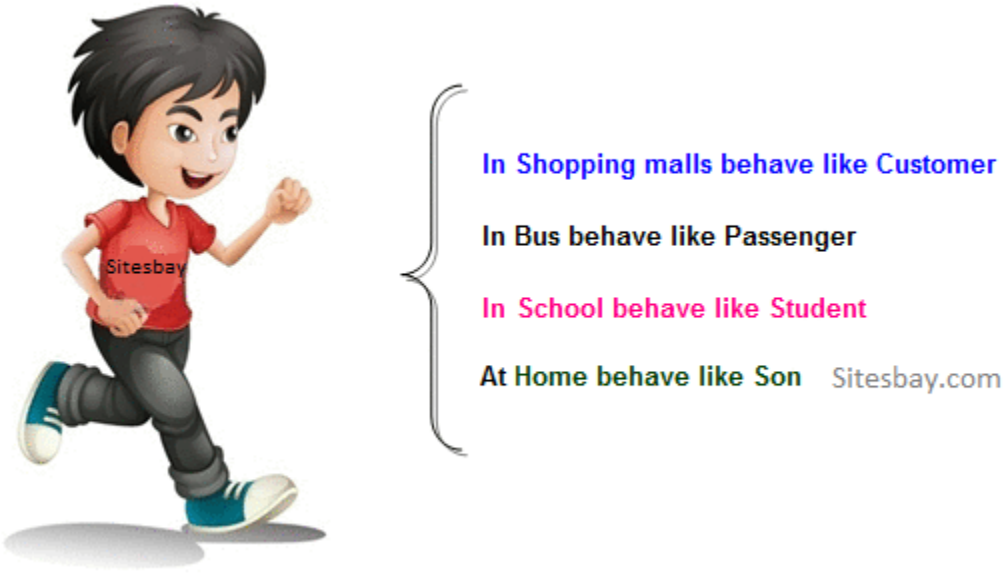
//

Polymorphism in Java:

The process of representing one form in multiple forms is known as **Polymorphism**.

Real life example of polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like on or daughter, Here one person present in different-different behaviors.



How to achieve Polymorphism in Java ?

Polymorphism principal is divided into two sub principal they are:

- Static or Compile time polymorphism
- Dynamic or Runtime polymorphism

Note: Java programming does not support static polymorphism because of its limitations and java always supports dynamic polymorphism.

Example of Runtime Polymorphism in Java

In below example we create two class Person an Employee, Employee class extends Person class feature and override walk() method. We are calling the walk() method by the reference

variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime. Here method invocation is determined by the JVM not compiler, So it is known as runtime polymorphism.

```
class Person
{
        void walk()
        {
        System.out.println("Can Run....");
        }
}
class Employee extends Person
{
        void walk()
        {
        System.out.println("Running Fast...");
        }
public static void main(String arg[])
{
Person p=new Employee(); //upcasting
p.walk();
}
}
```

Output: Running Fast…

Dynamic Binding

Dynamic binding always says create an object of base class but do not create the object of derived classes. Dynamic binding principal is always used for executing polymorphic applications. Advantages of dynamic binding along with polymorphism with method overriding are.

- Less memory space
- Less execution time

- More performance

Static polymorphism

The process of binding the overloaded method within object at compile time is known as **Static polymorphism** due to static polymorphism utilization of resources (main memory space) is poor because for each and every overloaded method a memory space is created at compile time when it binds with an object.

In C++ environment the above problem can be solve by using dynamic polymorphism by implementing with virtual and pure virtual function so most of the C++ developer in real worlds follows only dynamic polymorphism.

Dynamic polymorphism

In dynamic polymorphism method of the program binds with an object at runtime the advantage of dynamic polymorphism is allocating the memory space for the method (either for overloaded method or for override method) at run time.