Polymorphism and Examples

-K.L.MADHAVI

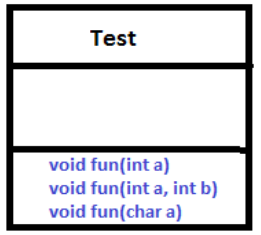
Polymorphism

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

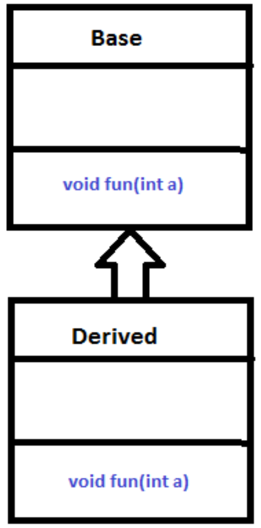
There are two types of polymorphism in Java:

1.compile-time polymorphism (Overloading)

2.runtime polymorphism. (Overriding)



Overloading



Overriding

Compile-time polymorphism(Overloading)

Compile-time polymorphism is a programming technique that involves calling methods during the compilation phase. It's also known as static polymorphism or early binding. Overloading of methods is called through the reference variable of a class. Compile-time polymorphism is achieved by **method overloading,constructor overloading and operator overloading.**

1.Method overloading

Method Overloading occurs when a class has many methods with the same name but they have different parameters or return type is known as method overloading.

Example:

```
class Overload
{
     void add()
      {
          //body
      }
     void add(int x,int y)
     {
        //body
     }
    int add(int x,int y)
   {
     //body
   }
```

Example Program

```
Class A
{
     void add()
     {
          int a=10,b=20;
          int c=a+b;
          System.out.println(“method without arguments”);
          System.out.println(“Sum=“+c);
      }
      Void add(int x,int y)
      {
          int z=x+y;
          System.out.println(“method with arguments”);
          System.out.println(“sum=“+z);
      }
   }
```

```
Class Moverload
{
       public static void main(String args[])
       {
              A obj=new A();
              obj.add();
              obj.add(20,20);
        }
}
```

2.Constructor overloading

The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Consider the following Java program, in which we have used different constructors in the class.

Example Program:

```
class A
{
     A()
     {
           System.out.println(“constructor without arguments”);
      }
     A(int a)
     {
           System.out.println(“constructor with arguments”);
      }
}
```

```
Class Coverload
{
     public static void main(String args[])
     {
           A obj=new A();
           A obj1=new A();
      }
}
```

3.Operator overloading

An operator is said to be overloaded if it can be used to perform more than one function other than the one its pre-defined for.It is a mechanism through which we can change the meaning of a pre-defined operator and make it work for user-defined objects.

Note: Operator Overloading is not supported in Java, in the following example we're trying to achieve the same functionality by the use of methods.

Example Program:

```
public class OOverload
{
   void add(int a,int b)
    {
        int sum=a+b;
        System.out.println("addition of 2 integers:"+sum);
    }
    void add(String s1,String s2)
    {
        String con_str=s1+s2;
        System.out.println("concatenated Strings:"+con_str);
    }
 }
```

```
Public static void main(String args[])
{
     OOverload obj=new OOverload();
     obj.add(10,10);
     obj.add("operator "," overloading");
}
```

Runtime polymorphism(Overriding)

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by **Method Overriding**.It occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

1.Should follow inheritance

2.both methods should have same name in sub and super class.

3.both methods must have same return type,parameters.

4.both methods should have same scope.

5.methods declared as final cannot be overridden.

6.static methods cant be overridden.

Example Program

```
class Parent
{
      void display()
       {
              System.out.println(“Parent class”);
        }
}
class Child1 extends Parent
{
      void display()
      {
             System.out.println(“Child1 class”);
      }
}
Class  Child2 extends Parent
{
       void display()
      {
             System.out.println(“Child2 class”);
      }
}
```

```
Class Moverride
{
        Public static void main(String args[])
        {
              Child1 obj1=new Child1();
              obj1.display();
              Child2 obj2=new Child2();
              obj2.display();
          }
}
```

Method overloading vs Method overriding

Method overloading	Method overriding
Compile-time polymorphism	Runtime polymorphism
Static Binding	Dynamic Binding
This can be implemented in a single class	This can be implemented in 2 classes
No need of inheritance concept	Uses inheritance concept
Method name should be same	Method name should be same
parameters or return type can be different	Both methods should have same parameters and return type
Static methods can be overloaded	Static methods cant be overridden