

Chapter 6

POLYMORPHISM

Polymorphism

The word polymorphism means having many forms (Poly means many and morph means forms). In simple words, we can define polymorphism as the ability of an entity to exhibit multiple behaviors or definitions depending upon the working environment. Polymorphism is of two types:

- **Static polymorphism/Early Binding/Compile-time:** If the polymorphism is achieved during program compilation, it is known as static polymorphism. It is therefore also known as compile time polymorphism or early binding.
- **Dynamic polymorphism/Late Binding/run-time:** If the polymorphism is achieved during program execution, it is known as dynamic polymorphism. It is therefore also known as run time polymorphism or late binding.

Method Overloading

If a class has multiple methods having same name but different signatures i.e. different no and types of arguments, it is known as Method Overloading. A method in java can be overloaded in two ways:

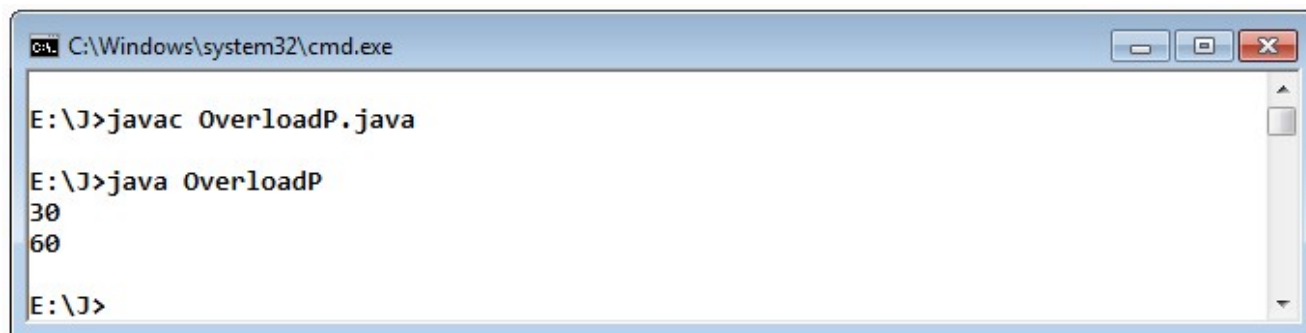
- By changing number of arguments
- By changing the data type

Program 1 Write a program to demonstrate method overloading in java by changing the no of arguments.

Solution:

```
class Adder
{
    static int add(int a,int b)
    {
        return a+b;
    }
    static int add(int a,int b,int c)
    {
        return a+b+c;
    }
}
```

```
class OverloadP
{
    public static void main(String[] args)
    {
        System.out.println(Adder.add(10,20));
        System.out.println(Adder.add(10,20,30));
    }
}
```

Output:

```
C:\Windows\system32\cmd.exe

E:\J>javac OverloadP.java

E:\J>java OverloadP
30
60

E:\J>
```

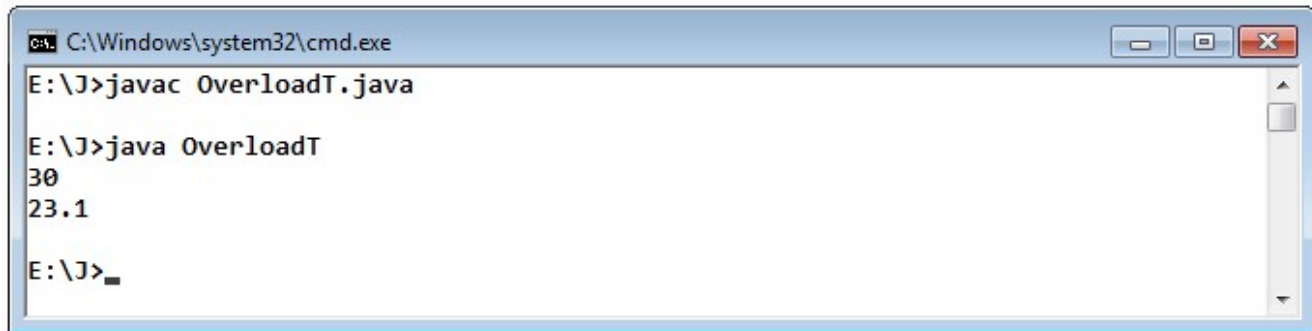
Program 2 Write a program to demonstrate method overloading in java by changing the types of arguments.

Solution:

```
class Adder
{
    static int add(int a, int b)
    {
        return a+b;
    }
    static double add(double a, double b)
    {
        return a+b;
    }
}

class OverloadT
{
    public static void main(String[] args)
    {
```

```
        System.out.println(Adder.add(10,20));  
        System.out.println(Adder.add(10.3,12.8));  
    }  
}
```

Output:

```
C:\Windows\system32\cmd.exe  
E:\J>javac OverloadT.java  
  
E:\J>java OverloadT  
30  
23.1  
  
E:\J>
```

Note: It is not possible to overload methods by simply changing their return types as it leads to compilation error due to ambiguity problem. It is also possible to overload main() method in a java program, but compiler call that function which takes string array as its arguments.

Method Overriding

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to override the method in the super class.

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 super class will be hidden.

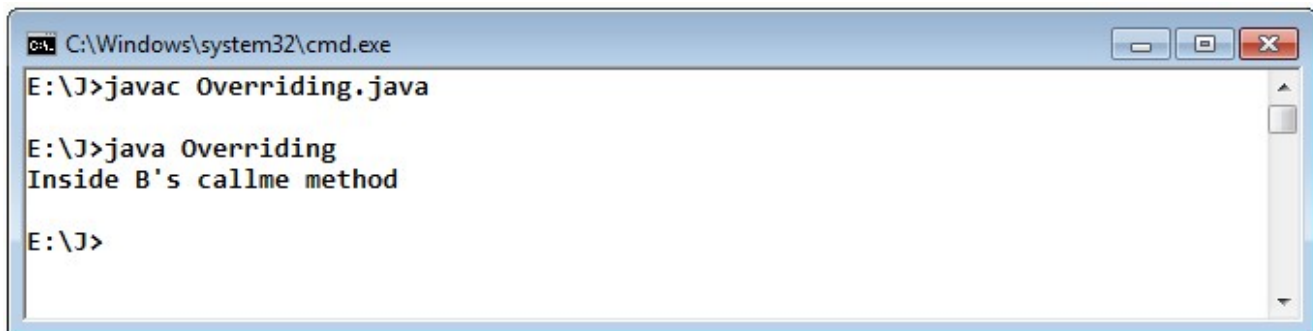
A real example of Java Method Overriding: Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.

Program 3 Write a program to demonstrate method overriding in java.

Solution:

```
class A  
{  
    void callme()  
    {  
        System.out.println("Inside A's callme method");  
    }  
}
```

```
    }  
}  
class B extends A  
{  
    void callme()  
    {  
        System.out.println("Inside B's callme method");  
    }  
}  
class Overriding  
{  
    public static void main(String args[])  
    {  
        B b = new B();  
        b.callme();           // calls B's version of callme  
    }  
}
```

Output:

```
C:\Windows\system32\cmd.exe  
E:\J>javac Overriding.java  
E:\J>java Overriding  
Inside B's callme method  
E:\J>
```

Explanation: When callme () is invoked on an object of type B, the version of callme () defined within B is used. That is, the version of callme () inside B overrides the version declared in A. If we wish to access the super class version of an overridden function, we can do so by using super.

Note: A static method cannot be overridden. It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area. The main is a static method, hence it cannot be overridden.

Dynamic Method Dispatch

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism.

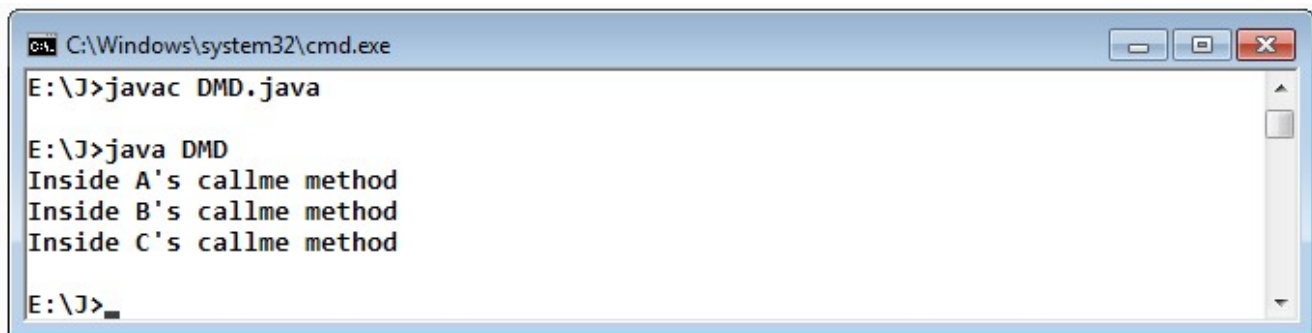
Program 4 Write a program to demonstrate the use of dynamic method dispatch.

Solution:

```
class A
{
    void callme()
    {
        System.out.println("Inside A's callme method");
    }
}
class B extends A
{
    void callme()
    {
        System.out.println("Inside B's callme method");
    }
}
class C extends B
{
    void callme()
    {
        System.out.println("Inside C's callme method");
    }
}
class DMD
{
    public static void main(String args[])
    {
        A a = new A();
        B b = new B();
        C c = new C();
    }
}
```

```
A r;                // obtain a reference of type A
r = a;              // r refers to an A object
r.callme();         // calls A's version of callme
r = b;              // r refers to a B object
r.callme();         // calls B's version of callme
r = c;              // r refers to a C object
r.callme();         // calls C's version of callme
}
}
```

Output



```
C:\Windows\system32\cmd.exe
E:\J>javac DMD.java

E:\J>java DMD
Inside A's callme method
Inside B's callme method
Inside C's callme method

E:\J>
```

Abstract class

Abstract classes are the class which contains method without providing a complete implementation (or without having background details). It is not possible to create object of an abstract class. Abstract classes can include methods having all details.

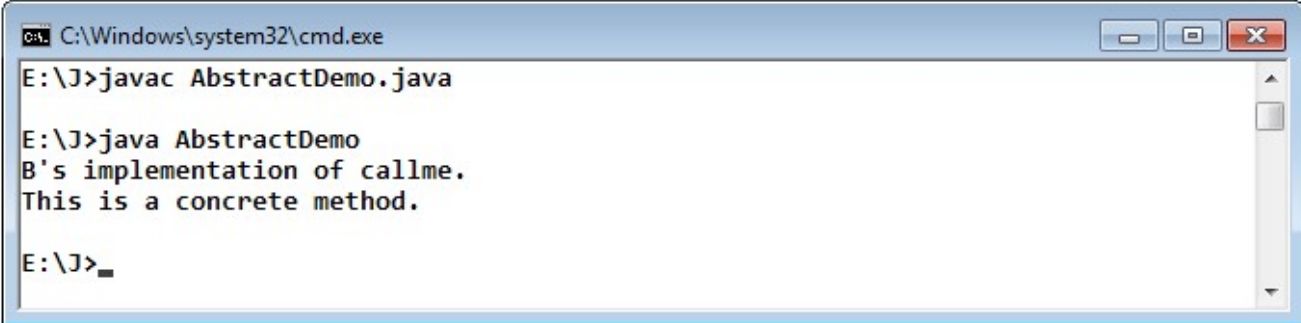
Program 5 Write a program to demonstrate the use of abstract class in java.

Solution:

```
abstract class A
{
    abstract void callme();
    // concrete methods are still allowed in abstract classes
    void callmetoo()
    {
        System.out.println("This is a concrete method.");
    }
}
```

```
}  
class B extends A  
{  
    void callme()  
    {  
        System.out.println("B's implementation of callme.");  
    }  
}  
class AbstractDemo  
{  
    public static void main(String args[])  
    {  
        B b = new B();  
        b.callme();  
        b.callmetoo();  
    }  
}
```

Output



```
C:\Windows\system32\cmd.exe  
E:\J>javac AbstractDemo.java  
  
E:\J>java AbstractDemo  
B's implementation of callme.  
This is a concrete method.  
  
E:\J>
```

Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java's approach to run-time polymorphism is implemented through the use of super class references. Thus, it must be possible to create a reference to an abstract class so that it can be used to point to a subclass object.

Uses of final Keyword

The keyword final has three uses:

- First, it can be used to create constant variable whose value cannot be changed.

- To disallow method Overriding
- To prevent inheritance

Final variable

The value of a variable declared as final cannot be changed. Below code depicts the use of final keyword against a variable.

```
final int x=7;  
x++; // Compilation error, x is final  
System.out.println(x); // Compilation error, x is final
```

Here x is declared as final, hence its value cannot be modified.

final method

While method overriding is one of Java's most powerful features, there will be times where we will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden.

```
class A  
{  
    final void meth()  
    {  
        System.out.println("This is a final method.");  
    }  
}  
class B extends A  
{  
    void meth()  
    {  
        System.out.println("Illegal!");  
    }  
}
```

Because meth () is declared as final, it cannot be overridden in B. If you attempt to do so, a compile-time error will result.

final class

Sometimes we want to prevent a class from being inherited. To do this, precede the class declaration with the keyword final. Declaring a class as final implicitly declares all of its methods as final, too.

```
final class A
```

```
{
```

```
    // ...
```

```
}
```

```
// The following class is illegal.
```

```
class B extends A
```

```
{
```

```
    // ERROR! Can't subclass A
```

```
    // ...
```

```
}
```

As the comments imply, it is illegal for B to inherit A since A is declared as final.

Review Questions of Chapter 6 (Polymorphism)

MCQ Type**1. Choose the most appropriate answer:****a) Static polymorphism is also known as**

- a) Compile time polymorphism
- b) Early binding
- c) both a) and b)
- d) None of these

b) Dynamic polymorphism is also known as

- a) Run time polymorphism
- b) late binding
- c) both a) or b)
- d) None of these

c) Method overloading is an example of

- a) Static polymorphism
- b) Structure
- c) Inheritance
- d) Polymorphism

d) Which one of the following is/ are examples of low level language?

- a) Assembly language
- b) Machine Language
- c) JAVA
- d) Both a) and b)

e) Which one of the following is/ are OOPs characteristics?

- a) Inheritance
- b) Data hiding
- c) Encapsulation
- d) All of these

f) The property of OOPs by which an entity can acquire multiple forms is known as

- a) Inheritance
- b) Polymorphism
- c) Encapsulation
- d) Data Aggregation

g) The act of representing essential feature of a class without specifying its complete details is known as

- a) Inheritance
- b) Polymorphism
- c) Data Encapsulation
- d) Data Abstraction

h) The act of wrapping data and methods into a single unit is known as

- a) Inheritance
- b) Polymorphism
- c) Data Encapsulation
- d) Data Abstraction

i) The technique by which object of one class acquires the properties of object of some other class is known as

- a) Inheritance
- b) Polymorphism
- c) Data Encapsulation
- d) Data Abstraction

j) What is use of interpreter?

- a) They convert byte code to machine language code
- b) They read high level code and execute them
- c) They are intermediated between JIT and JVM
- d) It is a synonym for JIT

k) The technique by which object of one class acquires the properties of object of some other class is known as

- a) Inheritance
- b) Polymorphism
- c) Data Encapsulation
- d) Data Abstraction

Short Type**2. Answer briefly**

- a) What is polymorphism?
- b) Define early binding. How it can be achieved?
- c) Define late binding. How it can be achieved?
- d) Can we inherit a final class?
- e) What is abstract class?

Long Type**3. Answer in details**

- a) What is polymorphism? Explain with a programming example how dynamic method dispatch (DMD) is used in java to achieve dynamic polymorphism.
- b) What is method overriding? Explain with the help of a suitable example.
- c) What is method overloading? Explain with the help of a programming example.
- d) With the help of suitable examples, explain the uses of final keyword in java.
