Lecture – 10 (Polymorphism)

Polymorphism in Java

Polymorphism in java is a concept by which we can perform a single action by 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: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Method Overloading in Java

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

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

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
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 TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}
}
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}
}
```

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}
}
```

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.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- 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 of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle{
void run(){
```

```
System.out.println("Vehicle is running");
}
}
class Bike2 extends Vehicle{
void run(){
System.out.println("Bike is running safely");
}
public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Real example of Java Method Overriding

```
class Bank{
int getRateOfInterest(){
return 0;
}
}
class CityBank extends Bank{
int getRateOfInterest(){
return 8;
}
}
class DBBL extends Bank{
int getRateOfInterest(){
return 7;
}
}
class BrackBank extends Bank{
int getRateOfInterest(){
return 9;
}
}

class Test2{
public static void main(String args[]){
CityBank  s=new CityBank();
```

```
DBBL  i=new DBBL();
BrackBank  a=new BrackBank ();
System.out.println("CityBank  Rate of Interest: "+s.getRateOfInterest());
System.out.println("DBBL  Rate of Interest: "+i.getRateOfInterest());
System.out.println("BrackBank  Rate of Interest: "+a.getRateOfInterest());
}
}
```

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

(1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
```

```
 obj.run();
 }
}//end of class
```

Output:Compile Time Error

(2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){
   System.out.println("running safely with 100kmph");
   }
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:Compile Time Error

(3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{
 }
class Honda1 extends Bike{
```

```
void run(){
System.out.println("running safely with 100kmph");
}
public static void main(String args[]){
Honda1 honda= new Honda();
honda.run();
}
}
```

Output:Compile Time Error

Q) Is final method inherited?

Ans.) Yes, final method is inherited but you cannot override it.

Example:

```
class Bike{
  final void run(){
   System.out.println("running...");
  }
  }
class Honda2 extends Bike{
   public static void main(String args[]){
    new Honda2().run();
   }
}
```

Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:

```
class A{
}
class B extends A{
}

A a=new B();//upcasting
```

Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run 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.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
class Bike{
  void run(){
System.out.println("running");
}
}
class Splender extends Bike{
  void run(){
  System.out.println("running safely with 60km");
}
  public static void main(String args[]){
    Bike b = new Splender();//upcasting
    b.run();
  }
}
```

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

(1) static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example of static binding

```
class Dog{
 private void eat(){
System.out.println("dog is eating...");
}
 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

(2) Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example of dynamic binding

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

```
class Dog extends Animal{
 void eat(){
System.out.println("dog is eating...");
}
 public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
```

```
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){
System.out.println("eating...");
}
}
class Dog extends Animal{
void eat(){
System.out.println("eating bread...");
}
void bark(){
System.out.println("barking...");
}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}
}
```

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
Animal(){
System.out.println("animal is created");
}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}
}
```