

Java - Introduction to Programming

Lecture 10

Polymorphism

Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. Precisely, Poly means ‘many’ and morphism means ‘forms’.

Types of Polymorphism **IMP**

1. Compile Time Polymorphism (Static)
2. Runtime Polymorphism (Dynamic)

Let's understand them one by one:

Compile Time Polymorphism : The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Example - Method Overloading

Method Overloading : Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. The type of the parameters passed to the function.
2. The number of parameters passed to the function.
3. The Sequence of parameters passed to the function.

```
class Student{  
    String name;  
    int age;  
  
    public void printInfo(String name){  
        System.out.println(name);  
    }  
  
    public void printInfo(int age){  
        System.out.println(age);  
    }  
  
    public void printInfo(String name, int age){  
        System.out.println(name + " " + age);  
    }  
    public void printInfo(int age, String name){  
        System.out.println(age + " " + name);  
    }  
}
```

```

        }

    }

public class OOPS {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.printInfo(24);
        s1.printInfo("aman");
        s1.printInfo("piyush",34);
        s1.printInfo(35, "akash");
    }
}

```

Runtime Polymorphism : Runtime polymorphism is also known as **dynamic polymorphism**. Function overriding is an example of runtime polymorphism. Function overriding means when the child class contains the method which is already present in the parent class. Hence, **the child class overrides the method of the parent class**. In case of function overriding, parent and child classes both contain the same function with a different definition. The call to the function is determined at runtime is known as runtime polymorphism.

```

// Example of Overriding in Java
class Animal {
    // Base class
    void move()
    {
        System.out.println("Animal is moving.");
    }
    void eat()
    {
        System.out.println("Animal is eating.");
    }
}

class Dog extends Animal {
    @Override void move()
    { // move method from Base class is overriden in this
        // method
        System.out.println("Dog is running.");
    }
    void bark() { System.out.println("Dog is barking."); }
}

public class MethOverRiding2 {
    public static void main(String[] args)

```

```

{
    Dog d = new Dog();
    d.move(); // Output: Dog is running.

}

```

Inheritance

Inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such a way, you can **reuse**, **extend or modify** the attributes and behaviours, which are defined in other classes.

In Java, the class, which inherits the members of another class, is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

```

class Shape{
    String color;
    //base class
}

class Triangle extends Shape{
    //derived class
}

class Rectangle extends Shape{
    //derived Class
}

public class Inheritance {
    public static void main(String[] args) {
        Triangle t1 = new Triangle();
        t1.color = "red";

        Rectangle r1 = new Rectangle();
        r1.color = "black";
    }
}

```

Types of Inheritance:

1. Single inheritance: When one class inherits another class, it is known as single level inheritance

```

class Shape{
    String color;
    //base class
}

class Triangle extends Shape{
    //derived class
}

public class Inheritance {
    public static void main(String[] args) {
        Triangle t1 = new Triangle();
        t1.color = "red";
        System.out.println(t1.color);
    }
}

```

2. Hierarchical inheritance : Hierarchical inheritance is defined as the process of deriving more than one class from a base class

```

//base class
class Shape{
    public void area(){
        System.out.println("display area");
    }
}

class Triangle extends Shape{
    public void area(float l, float h){
        System.out.println((1/2)*l*h);
    }
}

class Circle extends Shape {
    public void area(double r){
        System.out.println((3.14)*r*r);
    }
}

public class Inheritance {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        c1.area();
        c1.area(5);
    }
}

```

3. Multilevel inheritance: Multilevel inheritance is a process of deriving a class from another derived class.

```
//base class
class Shape{
    public void area(){
        System.out.println("display area");
    }
}
//derived class of base class
class Triangle extends Shape{
    public void area(double l, double h){
        System.out.println((l*h)/2);
    }
}
//derived class of derived class
class EquilateralTriangle extends Triangle {

}

public class Inheritance {
    public static void main(String[] args) {
        EquilateralTriangle e1 = new EquilateralTriangle();
        e1.area();
        e1.area(10, 10);
    }
}
```

4. Hybrid inheritance: Hybrid inheritance is a combination of Single, multiple inheritance and hierarchical inheritance.

```
//base class
class Shape{
    public void area(){
        System.out.println("display area");
    }
}
//derived class of base class
class Triangle extends Shape{
    public void area(double l, double h){
        System.out.println((l*h)/2);
    }
}
//derived class of derived class
```

```
class EquilateralTriangle extends Triangle {  
}  
  
//derived class of base class  
class Circle extends Shape{  
    public void area(double r){  
        System.out.println((3.14)*r*r);  
    }  
}  
  
public class Inheritance {  
    public static void main(String[] args) {  
        EquilateralTriangle e1 = new EquilateralTriangle();  
        e1.area();  
        e1.area(10, 10);  
  
        Circle c1 = new Circle();  
        c1.area(5);  
        c1.area();  
    }  
}
```