14. Polymorphism

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.

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.

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

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


```
1. class A{}
2. class B extends A{}
1. 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.

```
1. class Bike{
2.   void run(){System.out.println("running");}
3. }
4. class Splender extends Bike{
5.   void run(){System.out.println("running safely with 60km");}
6.
7.   public static void main(String args[]){
8.     Bike b = new Splender();//upcasting
9.     b.run();
10.  }
11. }
```

Output: running safely with 60km.

➢ Java Runtime Polymorphism Example: Shape

```
1. class Shape{
2. void draw(){
3. System.out.println("drawing...");
4. }
5. }
6. class Rectangle extends Shape{
7. void draw(){System.out.println("drawing rectangle...");}
8. }
9. class Circle extends Shape{
10. void draw(){System.out.println("drawing circle...");}
11. }
12. class Triangle extends Shape{
13. void draw(){System.out.println("drawing triangle...");}
14. }
15. class TestPolymorphism{
16. public static void main(String args[])
17. {
18. Shape s;
19. s=new Rectangle();
20. s.draw();
21. s=new Circle();
22. s.draw();
```

```
23. s=new Triangle();
24. s.draw();
25. }
26. }
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

➢ Java Runtime Polymorphism with Multilevel Inheritance

Let's see the simple example of Runtime Polymorphism with multilevel inheritance.

```
1. class Animal{
2. void eat(){System.out.println("eating");}
3. }
4. class Dog extends Animal{
5. void eat(){System.out.println("eating fruits");}
6. }
7. class BabyDog extends Dog{
8. void eat(){System.out.println("drinking milk");}
9. public static void main(String args[]){
10. Animal a1,a2,a3;
11. a1=new Animal();
12. a2=new Dog();
13. a3=new BabyDog();
14. a1.eat();
15. a2.eat();
16. a3.eat();
17. }
18. }
```

Output:

```
eating
eating fruits
drinking Milk
```

Try for Output

```
1. class Animal{
2. void eat(){System.out.println("animal is eating...");}
3. }
4. class Dog extends Animal{
5. void eat(){System.out.println("dog is eating...");}
6. }
7. class BabyDog extends Dog{
8. public static void main(String args[]){
9. Animal a=new BabyDog();
10. a.eat();
11. }}
```

Output:

Dog is eating (or) Animal is eating (or) Both Wrong

Since, BabyDog is not overriding the eat() method, so eat() method of Dog class is invoked.