Zoho Schools for Graduate Studies

Notes

DAY - 7(25-07-2025)

Session -1

Polymorphism

1. Polymorphism in Java:

Definition:

Polymorphism allows objects to take many forms. In Java, this means a single interface can be used for different underlying forms (data types).

Type	Description	How to Achieve
Compiletime	Decided at compile-time (method overloading)	Method Overloading
Runtime	Decided at run-time (method overriding, dynamic method dispatch)	Method Overriding, Interfaces

2.Compile-time Polymorphism (Method Overloading):

How:

Multiple methods with the same name but different parameters in class.

Example Code:

```
class Calculator {
        int add(int a, int b)
        {
                return a + b;
         }
        double add(double a, double b)
        {
                return a + b;
        }
        String add(String a, String b)
        {
        return a + b;
         }
        }
         class Main {
                public static void main(String[] args) {

                         Calculator calc = new Calculator();

                         System.out.println(calc.add(2, 3)); // 5

                         System.out.println(calc.add(2.0, 3.0)); // 5.0

                         System.out.println(calc.add("Hi", "All")); // HiAll

                        }
                }
```

Runtime Polymorphism (Method Overriding):

How:

A subclass provides a specific implementation of a method already declared in its superclass..

Example:

```
`       class Animal {
                void sound() {
                        System.out.println("Some sound");
        }
}
 class Dog extends Animal {
        void sound() {
                System.out.println("Bark");
        }
}
class Main {
         public static void main(String[] args) {
                Animal obj = new Dog(); obj.sound(); // Output: Bark
         }
}
```

Real-lifeExample:

- A "makePayment()" method — a superclass Payment has the method, subclasses like CreditCard, UPI, NetBanking override it to process payment differently. At runtime, Java decides which method to call depending on the payment method object created.

2. Abstraction

Definition:

Abstraction is the process of hiding internal details and showing only necessary features of an object.

How to Achieve:

In Java:

- Abstract Classes (Partial abstraction)
- Interfaces (100% abstraction)

Example:

```
      abstract class Vehicle {
              abstract void start();
             }
class Car extends Vehicle
             {
              void start() {
                    System.out.println("Car started");
             }
      }
```

```
class Main {
        public static void main(String[] args) {
         Vehicle v = new Car(); v.start(); // Output: Car started

        }

        }
```

Session -2

3. Interfaces Relationship

Definition:

An interface is a contract. Classes that implement it must provide concrete implementations for all its methods.

Relationship Type:

"Implements" relationship — a class IS-A implementer of the interface
Interface defines the "what", implementing class defines the "how".

Example Code:

```
interface Flyable {
                void fly();
        }
 class Bird implements Flyable {
        public void fly() {
                System.out.println("Bird flies");
        }
 }
 class Aeroplane implements Flyable {
        public void fly() {
                System.out.println("Aeroplane flies");
               }
        }
class Main {
        public static void main(String[] args) {

               Flyable obj1  = new Bird();
               Flyable obj2  = new Aeroplane();
               obj1.fly(); // Bird flies
               obj2.fly(); // Aeroplane flies

               }
 }
```

Real-life Example:

Different vehicles (Drone, Helicopter, Bird) — all can “fly” but the way they fly is different.

4. “IS-A” Relationship

Definition:

Using inheritance (extends), a subclass “is a” superclass (e.g., Dog is an Animal). When a class implements an interface, it “is a” implementer of that interface.

5. Table: Differences and Use Cases:

Topic	Compile-time Polymorphism	Runtime Polymorphism	Abstraction	Interface
How	Method Overloading	Method Overloading	Abstract class/interface	Abstract class/interface
When decided	Compile-time	Runtime	Compile/runtime	Compile-time
Real-time Usage	print(), add() with overload	Payment, sound() (animal)	TV remote	All “can do” actions
Relationship	NA	Inheritance	Generalization	Contract/blueprint

6. More Real-Time Examples

Compile-time polymorphism:

Calculator app: add(int, int), add(double, double)

Runtime polymorphism:

Notification app: send(message), override for SMS, email, push notification

Abstraction:

ATM machine: User knows to insert card and enter PIN, not how cash is dispensed.

Interface:

Payment Gateway: PayPal, Razorpay, Stripe classes implement Payment interface, each processes payment differently but can be used interchangeably.

7. Conclusion

- **Compile-time polymorphism** improves readability and function usage flexibility.
- **Runtime polymorphism** and abstraction enable Java to operate in a flexible, extensible, and modular manner.

- **Interfaces** are pure abstraction techniques representing the “contract” others must follow.

Referance :

1. https://www.geeksforgeeks.org/java/difference-between-compile-time-and-run-time-polymorphism-in -java/

2. https://www.geekster.in/articles/polymorphism-in-java/

3. https://www.tutorialspoint.com/compile-time-polymorphism-in-java

4. https://www.geeksforgeeks.org/java/polymorphism-in-java/

5. https://www.upgrad.com/blog/runtime-polymorphism-java-examples/

6. https://prepbytes.com/blog/runtime-polymorphism-in-java/

7. https://utho.com/blog/abstraction-in-java-and-oops/ .

8. https://www.geeksforgeeks.org/java/abstraction-in-java-2/

9. https://www.w3schools.com/java/java_abstract.asp

10. https://www.geeksforgeeks.org/java/interfaces-in-java/

11. https://www.programiz.com/java-programming/interfaces

12. https://stackoverflow.com/questions/35962451/what-kind-of-relationship-does-an-interface-have-with -it-implementing-class

13. https://www.geeksforgeeks.org/java/interfaces-and-polymorphism-in-java/