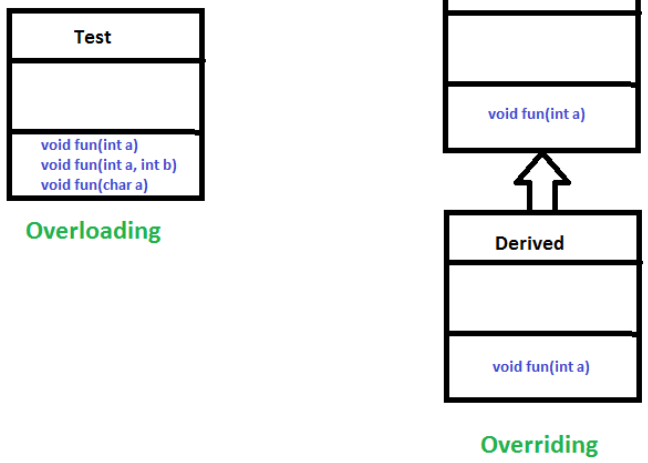


Polymorphism in Java

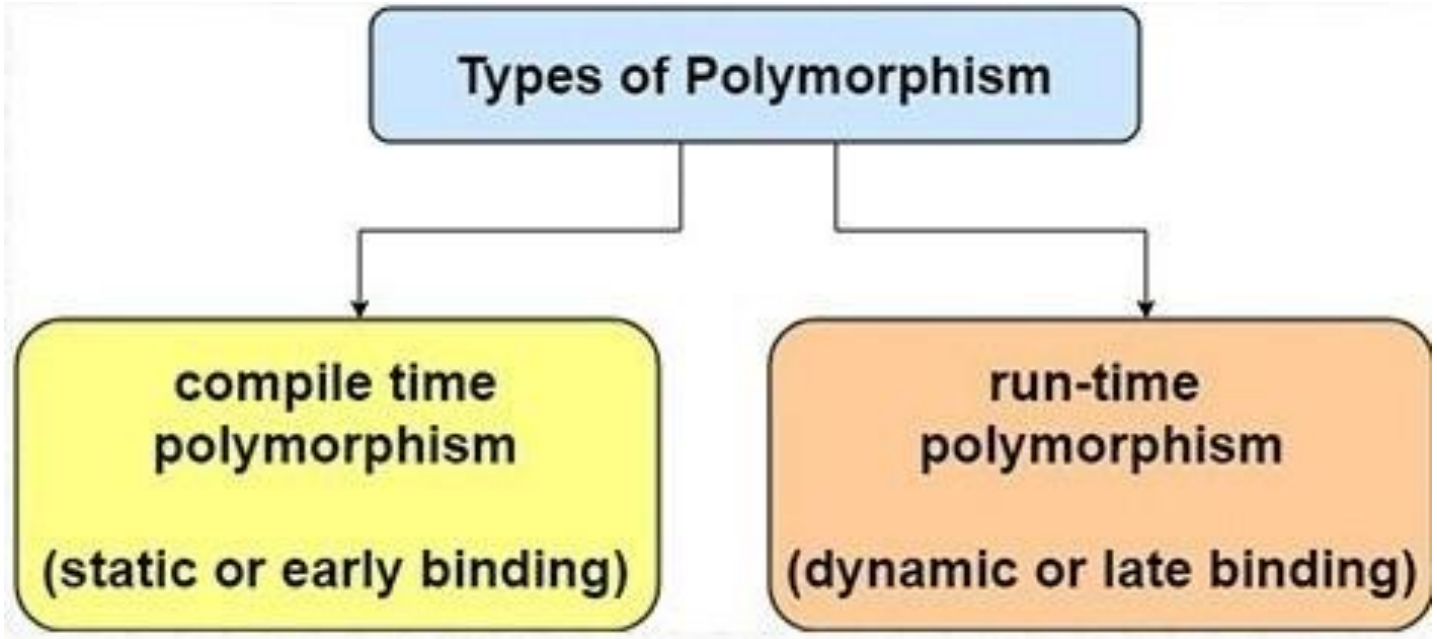
Nimesha Hewawasam
nimesha.h@nsbm.ac.lk

What Is Polymorphism?

- One entity can take many forms.
- Same method or object behaves differently based on the context, specially on the project's actual runtime class.
- Analogy: A person can be a father, husband, employee.
- **Key features of polymorphism:**
  - **Multiple Behaviors**: The same method can behave differently depending on the object that calls this method.
  - **Method Overriding**: A child class can redefine a method of its parent class.
  - **Method Overloading**: We can define multiple methods with the same name but different parameters.
  - **Runtime Decision:** At runtime, Java determines which method to call depending on the object's actual class.

Types of Polymorphism

- Compile-Time (Static Polymorphism) → Method Overloading
- Runtime (Dynamic Polymorphism) → Method Overriding / Dynamic Dispatch



Compile-Time Polymorphism (Method Overloading)

- The method overloading or static polymorphism, also known as Static Binding, also known as compile-time binding is a type where method calls are defined at the time of compilation.
- Method overloading allows us to have multiple methods with the same name having different datatypes of parameter, or a different number of parameters, or both.

No Method Overloading:

```
public class Number {

  public void sumInt(int a, int b) {
        System.out.println("Sum: " + (a + b));
  }

  public void sumDouble(double a, double b) {
        System.out.println("Sum: " + (a + b));
  }

  public static void main(String[] args) {

        Number number = new Number();

        number.sumInt(1, 2);
        number.sumDouble(1.8, 2.5);
  }
}
```

With Method Overloading:

```
public class Number {

  public void sum(int a, int b) {
        System.out.println("Sum: " + (a + b));
  }

  public void sum(double a, double b) {
        System.out.println("Sum: " + (a + b));
  }

  public static void main(String[] args) {

        Number number = new Number();

        number.sum(1, 2);
        number.sum(1.8, 2.5);
  }
}
```

Runtime Polymorphism (Method Overriding)

- In contrast to method overloading, method overriding allows you have to exactly the same signature as multiple methods, but they should be in multiple different classes.
- The question is how is this special? These classes have an IS-A relationship i.e. should have inheritance between them.
- In other words, in method overriding or dynamic polymorphism, methods are resolved dynamically at the runtime when the method is called.
- This is done based on the reference of the object it is initialized with.

Here is a small example of method overriding:

```java
public class Animal {

  public void walk() {
        System.out.println("Animal walks");
  }
}

public class Cat extends Animal {

  @Override
  public void walk() {
        System.out.println("Cat walks");
  }
}

public class Dog extends Animal {

  @Override
  public void walk() {
        System.out.println("Dog walks");
  }
}
```

```java
public class Main {

  public static void main(String[] args) {

        Animal animal = new Animal();
        animal.walk(); // Animal walks

        Cat cat = new Cat();
        cat.walk(); // Cat walks

        Dog dog = new Dog();
        dog.walk(); // Dog walks

        Animal animalCat = new Cat(); // Dynamic Polymorphism
        animalCat.walk(); // Cat walks

        Animal animalDog = new Dog(); // Dynamic Polymorphism
        animalDog.walk(); //Dog walks
  }
}
```

Why Use Polymorphism?

- Code Reusability – generic methods work across subclasses.
- Flexibility – treat different objects uniformly via superclass references.
- Dynamic Behavior – runtime method resolution ensures context-appropriate action.

Summary

- Polymorphism = many forms.
- Two types: Compile-time (overloading) and Runtime (overriding).
- Builds on inheritance for flexibility and maintainability.