

Advanced Programming

Inheritance & Polymorphism

Instructor: Ali Najimi

Author: Hossein Masihi

Department of Computer Engineering

Sharif University of Technology

Fall 2025





Extended Topics

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Compile-time Polymorphism
5. Runtime Polymorphism
6. Interface-based Polymorphism
7. Upcasting & Downcasting
8. Real-World Example



Single Inheritance

```
class Vehicle {  
    void start() { System.out.println("Vehicle started"); }  
}  
class Car extends Vehicle {  
    void playMusic() { System.out.println("Playing music..."); }  
}  
  
Car c = new Car();  
c.start();      // inherited  
c.playMusic(); // subclass method
```

- ◆ Simple and direct “is-a” relationship between two classes.



Multilevel Inheritance

```
class Animal {  
    void eat() { System.out.println("Eating..."); }  
}  
class Mammal extends Animal {  
    void walk() { System.out.println("Walking..."); }  
}  
class Human extends Mammal {  
    void speak() { System.out.println("Speaking..."); }  
}  
  
Human h = new Human();  
h.eat(); // from Animal  
h.walk(); // from Mammal  
h.speak(); // from Human
```

- ◆ Builds hierarchy across generations of classes.



Hierarchical Inheritance

```
class Shape {  
    void draw() { System.out.println("Drawing shape"); }  
}  
class Circle extends Shape {  
    void area() { System.out.println("Area = πr²"); }  
}  
class Rectangle extends Shape {  
    void area() { System.out.println("Area = l × w"); }  
}  
  
Shape s1 = new Circle();  
Shape s2 = new Rectangle();  
s1.draw();  
s2.draw();
```

- ◆ Multiple subclasses extend a common base class.



Compile-Time Polymorphism (Overloading)

```
class Calculator {  
    int add(int a, int b) { return a + b; }  
    double add(double a, double b) { return a + b; }  
    int add(int a, int b, int c) { return a + b + c; }  
}  
  
Calculator c = new Calculator();  
System.out.println(c.add(2,3));          // int version  
System.out.println(c.add(2.5,3.5));      // double version  
System.out.println(c.add(1,2,3));        // 3-arg version
```

- ◆ Method selected during **compile time** based on signature.



Runtime Polymorphism (Overriding)

```
class Employee {  
    void work() { System.out.println("Generic work"); }  
}  
class Developer extends Employee {  
    void work() { System.out.println("Writing code"); }  
}  
class Manager extends Employee {  
    void work() { System.out.println("Managing team"); }  
}  
  
Employee e1 = new Developer();  
Employee e2 = new Manager();  
  
e1.work(); // Writing code  
e2.work(); // Managing team
```

- ◆ Method resolved at **runtime** depending on actual object type.



Interface-based Polymorphism

```
interface Payment {
    void pay(double amount);
}
class PayPal implements Payment {
    public void pay(double amount) { System.out.println("Paid " + amount + " via PayPal"); }
}
class CreditCard implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " via Credit Card");
    }
}
Payment p1 = new PayPal();
Payment p2 = new CreditCard();
p1.pay(1000);
p2.pay(2000);
```

- ◆ Multiple implementations share the same interface for flexibility.



Upcasting & Downcasting

```
class Animal {  
    void makeSound() { System.out.println("Animal sound"); }  
}  
class Dog extends Animal {  
    void makeSound() { System.out.println("Woof!"); }  
    void bark() { System.out.println("Dog barking!"); }  
}  
  
Animal a = new Dog(); // Upcasting  
a.makeSound(); // Woof!  
  
Dog d = (Dog) a; // Downcasting  
d.bark(); // Access Dog-specific method
```

- ◆ Upcasting = safe and common
- ◆ Downcasting = risky, use only if type is certain



Real-world Example — Payment Gateway

```
abstract class Payment {  
    abstract void process();  
}  
class CardPayment extends Payment {  
    void process() { System.out.println("Processing card payment..."); }  
}  
class UpIPayment extends Payment {  
    void process() { System.out.println("Processing UPI payment..."); }  
}  
class PaymentSystem {  
    void execute(Payment p) { p.process(); // polymorphic call }  
}  
PaymentSystem ps = new PaymentSystem();  
ps.execute(new CardPayment());  
ps.execute(new UpIPayment());
```

- ◆ Same interface, different behaviors — real polymorphism in action.



Summary of Extended Concepts

Type	Description
Single Inheritance	One parent, one child
Multilevel	Multi-step chain of inheritance
Hierarchical	One parent, many children
Overloading	Compile-time polymorphism
Overriding	Runtime polymorphism
Interfaces	Multiple behavior contracts
Up/Down Casting	Type flexibility with inheritance

Thank You – Extended OOP in Java

Inheritance & Polymorphism – Advanced Examples



Advanced Programming – Fall 2025 – Sharif University of Technology