Java Abstraction

Table of contents

Introduced in 2026-03-05-object-oriented-programming-i[1] (Lecture, Week 2).

What is abstraction?

The ability to hide complex implementation details and expose only the essential behaviours. In Java, abstraction is often implemented using **interfaces** and **abstract classes** — both define methods that subclasses must implement, specifying a contract for *what* operations can be performed on an object without specifying *how* those operations are implemented.

Interfaces

An interface is a group of related methods with empty bodies (a contract that implementing classes must fulfil). To use an interface's methods, a class must **implement** it — a class can implement multiple interfaces.

```
public interface PaymentProcessor {
    void processPayment(double amount);
}

public class PayPalProcessor implements PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment: " + amount);
```

[1] courses/csse2002/2026-03-05-object-oriented-programming-i.pdf

```
    }
}

public class Main {
    public static void main(String[] args) {
        PaymentProcessor processor = new PayPalProcessor();
        processor.processPayment(100);
    }
}
```

The program only interacts with the `PaymentProcessor` interface — the implementation details are hidden inside the class that implements it.

Abstract classes

An abstract class **cannot be instantiated** directly. It can mix abstract methods (no body — declared with the `abstract` keyword) with fully-implemented methods, and subclasses must implement all its abstract methods.

```
public abstract class Payment {
    public void validateTransaction() {
        System.out.println("Validating transaction...");
    }
    abstract void processPayment(double amount);
}

public class PayPalPayment extends Payment {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment: $" + amount);
    }
}

public class Main {
    public static void main(String[] args) {
        Payment payment = new PayPalPayment();
        payment.validateTransaction();
        payment.processPayment(150.0);
    }
}
```

The abstract class defines the common behaviour of all payments (`validateTransaction`), while leaving the payment-specific logic (`processPayment`) to subclasses.

Interfaces vs abstract classes

Covered in 2026-03-12-object-oriented-programming-ii[2] (Lecture, Week 3) — Week 2's lecture left this comparison as a homework/self-study item.

- **Use an interface** to define a capability or contract, e.g. `Flyable`, `Runnable`, `Payable`. A class can implement multiple interfaces.
- **Use an abstract class** to define a shared base implementation (common behaviour), e.g. an `Animal` abstract class shared by `Dog` and `Cat`. A class can only extend one (super)class.

```
abstract class Animal {
    public void eat() {
        System.out.println("Animal is eating");
    }
    public abstract void makeSound();
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat Meows");
    }
}
```

A class can combine both: extend at most one abstract/concrete class, while implementing as many interfaces as needed:

```
class Dog extends Animal implements Runnable, Swimmable
class Duck extends Animal implements Flyable, Swimmable
```

[2] courses/csse2002/2026-03-12-object-oriented-programming-ii.pdf