Abstraction in Java — Simple Meaning

Abstraction means hiding unnecessary implementation details and showing only the important functionality.

Real-life example

Think about a **car** .

You use:

Start
Accelerate
Brake
Steering

You don't need to know exactly how the engine, fuel injection, or transmission works internally.

That's **abstraction**:

Show what something does, hide how it does it.

---

1. Abstraction Using Abstract Class

Java provides the `abstract` keyword.

Example

```
abstract class Animal {

   abstract void sound();

   void eat() {
      System.out.println("Animal eats");
   }
}

class Dog extends Animal {

   void sound() {
      System.out.println("Dog barks");
   }
}

public class Main {

   public static void main(String[] args) {

      Dog d = new Dog();

      d.sound();
      d.eat();
   }
}
```

Output

```
Dog barks
Animal eats
```

How does it work?

We have:

`abstract class Animal`

This means `Animal` is an **abstract class**.

Inside it:

```
abstract void sound();
```

This is an **abstract method**.

It says:

Every child class must provide its own implementation of `sound()`.

The `Dog` class provides that implementation:

```
void sound() {
   System.out.println("Dog barks");
}
```

So:

```
Animal
  ↓
 abstract sound()
  ↓
 Dog
  ↓
 sound() → Dog barks
```

---

2. Abstract Class Can Have Normal Methods

An abstract class can contain both:

- Abstract methods
- Normal methods

Example:

```
abstract class Animal {

   abstract void sound();

   void eat() {
      System.out.println("Animal eats");
   }
}
```

Here:

```
abstract void sound();
```

is an abstract method.

And:

```
void eat() {
   System.out.println("Animal eats");
}
```

is a normal method.

---

3. You Cannot Create an Object of an Abstract Class

This is **not allowed**:

```
Animal a = new Animal();  // □ Error
```

But you can create an object of the child class:

```
Dog d = new Dog();  // □
```

---

4. Abstraction Using Interface

An **interface** is another major way to achieve abstraction in Java.

Example

```
interface Animal {

   void sound();
}

class Dog implements Animal {

   public void sound() {
      System.out.println("Dog barks");
   }
}

public class Main {

   public static void main(String[] args) {

      Dog d = new Dog();

      d.sound();
   }
}
```

Output

```
Dog barks
```

Here:

`interface Animal`

defines what the class should do.

And:

`class Dog implements Animal`

provides the implementation.

Abstract Class vs Interface

Abstract Class	Interface
Uses `abstract class`	Uses `interface`
Child uses `extends`	Class uses `implements`
Can have abstract methods	Can declare abstract methods
Can have normal methods	Can also have `default`/`static` methods
Can have constructors	Cannot be instantiated
A class can extend only one class	A class can implement multiple interfaces

---

Encapsulation vs Abstraction

These two are often confusing.

Encapsulation

Hides/protects data.

private int balance;

Think:

"Protect my data."

Abstraction

Hides implementation details.

abstract void sound();

Think:

"Don't worry about how it works; just know what it does."

Easy Memory Trick □

Encapsulation → HOW to protect data
Abstraction → WHAT to show/hide

Encapsulation → Data hiding
Abstraction → Implementation hiding