Abstraction in Java

Abstraction means **showing only necessary information** and **hiding the unnecessary or complex implementation details**.

Real-life example:

ATM Machine
We perform actions like *withdraw*, *deposit*, *check balance*, etc.
But we don't know how the ATM internally communicates with the bank server.
This hiding of internal details is an abstraction.

In Java, we achieve abstraction in **two ways:**

1. **Abstract Class**
2. **Interface**

1. Abstract Class

A class declared using the abstract keyword is called an abstract class.

Key Points:

- It can have **both concrete (implemented)** methods and **abstract (non-implemented)** methods.
- Abstract methods **cannot** be `static`, `final`, or `private`.
- **Abstract method** → only method declaration, no body.
- We **cannot create objects** of an abstract class.
- It **can have constructors** (these will run when a child class object is created).
- Variables: Can be declared **with or without initialization** (your choice).
- Used when you know **partial implementation**.
- It can have:
    - **Instance blocks**
    - **Static blocks**
- **Cannot achieve multiple inheritance.**

2. Interface

An interface is a **blueprint of a class**.
Use an interface when you have **only requirements (specification)** and **no implementation**.

Key Points:

- All methods inside an interface are **public** and **abstract** by default.
- We can create concrete (implemented) methods in an interface using the `default` keyword.

- It provides **100% abstraction**.

- Methods **cannot be**: `private`, `protected`, `final`, `static`, `synchronized`.
- Variables inside an interface are always: public, static,final
- In an interface, you **cannot declare variables** as**:**
  `private`,`protected`,`transient`,`volatile`

- So you **must assign value** at the time of declaration, otherwise compile-time error.

- Cannot have:
  - **Constructors**
  - **Instance blocks**
  - **Static blocks**
- Interfaces **support multiple inheritance**.

```
interface Abstraction {

    // Interface variable: always public static final
    int fee = 0;

    // Default method (concrete method allowed from Java 8)
    default void greet() {
        System.out.println("Welcome to Abstraction Tutorial");
    }

    // Abstract method (public + abstract by default)
    void defination();
}

class Oops implements Abstraction {

    @Override
    public void defination() {
        System.out.println("Abstraction hides internal details and shows only necessary information.");
    }
}

class Main {
    public static void main(String[] args) {
        Oops obj = new Oops();
        obj.greet();
        obj.defination();
        System.out.println(obj.fee);
    }
}

Output
------
Welcome to Abstraction Tutorial
Abstraction hides internal details and shows only necessary information.
0
```

```java
abstract class Abstraction{
    //constructor
    Abstraction(){
        System.out.println("Abstraction-Constructor");
    }

    int fee;

    //concrete method
    public void greet(){
        System.out.println("Welcome, To Abstraction tutorial");
    }

    //abstract method
    public abstract void defination();
}

class Oops extends Abstraction{

    Oops(){
        System.out.println("Object Oriented Programming");
    }

    //implement abstract method
    @Override
    public void defination(){
        System.out.println("Abstraction hides internal details and shows only necessary information.");
    }
}

class Main {
    public static void main(String[] args) {
        Oops obj = new Oops();    // constructor chain
        obj.greet();              // concrete method
        obj.defination();         // abstract method implemented in child
    }
}
```

Output

```
Abstraction-Constructor
Object Oriented Programming
Welcome, To Abstraction tutorial
Abstraction hides internal details and shows only necessary information.
```