**1. Explain what is an Interface in Java?**

**Definition:**  
An **interface** in Java is a **blueprint of a class** that contains only **abstract methods** (until Java 7) and **default/static methods** (Java 8+). It defines **what a class should do** but not **how** it will do it.

* All methods in an interface are **public and abstract** by default (except default and static methods).
* All variables are **public, static, and final** by default.

**Example:**

java

CopyEdit

interface Animal {

void eat();

void sleep();

}

class Dog implements Animal {

public void eat() { System.out.println("Dog eats bone."); }

public void sleep() { System.out.println("Dog sleeps."); }

}

**2. Difference between Abstract Class and Interface (and when to use each)**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| **Methods** | Can have abstract **and** concrete methods | Only abstract methods (Java 7), abstract + default + static methods (Java 8+) |
| **Variables** | Can have instance & static variables | Only public static final constants |
| **Access Modifiers** | Can have any access modifier | Methods are public by default |
| **Constructors** | Can have constructors | No constructors |
| **Multiple Inheritance** | A class can extend only **one** abstract class | A class can implement **multiple** interfaces |
| **When to Use** | When you have **common code** and want to share implementation details | When you need **full abstraction** and multiple inheritance |

**Rule of Thumb:**

* Use **abstract class** when:
  + You have **partial abstraction** (some shared implementation).
  + You expect subclasses to share **state/fields**.
* Use **interface** when:
  + You want **full abstraction**.
  + You want to implement multiple unrelated capabilities.

**3. What does Abstraction mean?**

**Definition:**  
Abstraction is the process of **hiding implementation details** and **showing only the essential features** to the user.

**In Java:**

* Achieved using **abstract classes** and **interfaces**.
* Abstract class: hides partial details.
* Interface: hides all details.

**Analogy:**  
Think of a **TV remote**:

* You press buttons (interface) without knowing the circuit details (implementation).

**4. Why do we use Abstraction in Programming? (In Depth)**

**Main Reasons:**

1. **Hide complexity** – Users don’t need to know internal logic, just the API/method to call.
2. **Increase security** – Prevents direct access to the implementation details.
3. **Promote reusability** – Interfaces/abstract classes can be reused across multiple classes.
4. **Ease maintenance** – If implementation changes, user code remains unaffected.
5. **Enable polymorphism** – We can use parent type references to call different subclass implementations.

**Example in Java:**

java

CopyEdit

abstract class Payment {

abstract void pay(double amount);

}

class CreditCardPayment extends Payment {

void pay(double amount) {

System.out.println("Paid " + amount + " via Credit Card.");

}

}

class UpiPayment extends Payment {

void pay(double amount) {

System.out.println("Paid " + amount + " via UPI.");

}

}

Here, the **user only calls**:

java

CopyEdit

Payment p = new UpiPayment();

p.pay(500);

They don’t care **how** UPI payment works internally.