🔌 1. What is Adapter Design Pattern?

The **Adapter Pattern** allows **incompatible interfaces** to work together by **converting one interface into another expected by the client**.

🧠 Real-World Analogy

You have a **2-pin plug**, but the socket supports **3-pin**. You use an **adapter** to connect incompatible interfaces without modifying the plug or the socket.

🆚 2. Class Adapter vs Object Adapter

Feature	Class Adapter	Object Adapter
Uses	**Inheritance**	**Composition (Has-a)**
Inherits	From both target and adaptee	From target only; holds adaptee reference
Language Support	Requires **multiple inheritance** (not in Java)	Fully supported in Java
Flexibility	Rigid – tied to one adaptee class	Flexible – can wrap multiple adaptees
Reuse	Can't reuse adaptee as-is	Can reuse adaptee without modification

---

🧪 3. Java Implementation

🔷 A. Class Adapter in Java (via `implements + extends`)

Java doesn't support **true multiple inheritance**, so this is limited.

Scenario: A `VGA` monitor expects VGA input, but the client provides HDMI.

```
// Adaptee (incompatible interface)
class HDMI {
    public void connectHDMI() {
        System.out.println("Connected via HDMI");
    }
}

// Target interface
interface VGA {
    void connectVGA();
}

// Adapter using inheritance (Class Adapter)
class HDMIToVGAAdapter extends HDMI implements VGA {
    @Override
    public void connectVGA() {
```

```
        System.out.println("Adapting HDMI to VGA");
        connectHDMI();
    }
}
```

✅ Usage:

```
public class ClassAdapterDemo {
    public static void main(String[] args) {
        VGA monitor = new HDMIToVGAAdapter();
        monitor.connectVGA();
    }
}
```

🔶 B. Object Adapter (Recommended in Java)

```
// Adaptee
class HDMI {
    public void connectHDMI() {
        System.out.println("Connected via HDMI");
    }
}

// Target
interface VGA {
    void connectVGA();
}

// Object Adapter using composition
class HDMIToVGAObjectAdapter implements VGA {
    private HDMI hdmi;

    public HDMIToVGAObjectAdapter(HDMI hdmi) {
        this.hdmi = hdmi;
    }

    @Override
    public void connectVGA() {
        System.out.println("Adapting HDMI to VGA");
        hdmi.connectHDMI();
    }
}
```

✅ Usage:

```
public class ObjectAdapterDemo {
    public static void main(String[] args) {
        HDMI hdmi = new HDMI();
        VGA adapter = new HDMIToVGAObjectAdapter(hdmi);
        adapter.connectVGA();
    }
}
```

✅ **Object Adapter is more flexible and favored in Java** because it avoids subclassing limitations.

📦 4. Examples from JDK Internal Code (Real-Life)

✅ A. `java.util.Arrays#asList()` — Object Adapter

```
String[] array = {"a", "b", "c"};
List<String> list = Arrays.asList(array);
```

- Adapts an **array** to a **List** interface.
- The returned list is a wrapper: not a full `ArrayList`, but **delegates to the array internally**.

```
public static <T> List<T> asList(T... a) {
    return new ArrayList<>(a); // internal adapter
}
```

✅ B. `java.io.InputStreamReader` – Object Adapter

```
InputStream in = new FileInputStream("test.txt");
Reader reader = new InputStreamReader(in);
```

- Adapts an `InputStream` (byte-based) to a `Reader` (character-based).
- Internally wraps the `InputStream`.

```
public class InputStreamReader extends Reader {
    private final InputStream in;
    // adapts InputStream to Reader
}
```

✅ C. `java.util.Enumeration` to `Iterator` adapter

```
Enumeration<String> e = legacyMethod();
Iterator<String> iterator = Collections.list(e).iterator();
```

- Adapts legacy `Enumeration` to `Iterator`.

✅ Interview Summary

> "The Adapter pattern bridges incompatible interfaces. In Java, **Class Adapter** uses inheritance, but it's limited due to lack of multiple inheritance. The **Object Adapter** is more common, using composition. Real-world examples in JDK include `InputStreamReader`, `Arrays.asList()`, and `Collections.list()`, which all adapt one interface to another without modifying existing code."