5. D - Dependency Inversion Principle (DIP)

Resources

- Video → 1. SOLID Principles with Easy Examples (Hindi) | OOPs SOLID Principles - Low Level Design

What is the Dependency Inversion Principle (DIP)?

> *The Dependency Inversion Principle (DIP) states that* ***"high-level components should not depend on low-level components directly; instead, they should depend on abstractions".***

In other words, classes should depend on interfaces rather than concrete classes. Additionally, abstractions should not depend on details; details should depend on abstractions.

Code Example: Violating DIP

Utility Classes

```java
public interface Keyboard {
    void getSpecifications();
}
public interface Mouse {
    void getSpecifications();
}

// Low-level module - concrete implementation
public class WiredKeyboard implements Keyboard {
    private final String connectionType;
    private final String company;
    private final String modelVersion;
    private final String color;

    public WiredKeyboard(String connectionType, String company,
String modelVersion, String color) {
        this.connectionType = connectionType;
        this.company = company;
        this.modelVersion = modelVersion;
        this.color = color;
    }

    public void getSpecifications() {
        System.out.println("===> Wired Keyboard");
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Company: " + company);
        System.out.println("Model Version: " + modelVersion);
        System.out.println("Color: " + color);
    }
}

// Low-level module - concrete implementation
public class BluetoothKeyboard implements Keyboard {
```

```
    private final String connectionType;
    private final String company;
    private final String modelVersion;
    private final String color;

    public BluetoothKeyboard(String connectionType, String company, String modelVersion, String color) {
        this.connectionType = connectionType;
        this.company = company;
        this.modelVersion = modelVersion;
        this.color = color;
    }

    public void getSpecifications() {
        System.out.println("===> Bluetooth Keyboard");
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Company: " + company);
        System.out.println("Model Version: " + modelVersion);
        System.out.println("Color: " + color);
    }
}

// Low-level module - concrete implementation
public class WiredMouse implements Mouse {
    private final String connectionType;
    private final String company;
    private final String modelVersion;
    private final String color;

    public WiredMouse(String connectionType, String company, String modelVersion, String color) {
        this.connectionType = connectionType;
        this.company = company;
        this.modelVersion = modelVersion;
        this.color = color;
    }

    public void getSpecifications() {
        System.out.println("===> Wired Mouse");
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Company: " + company);
        System.out.println("Model Version: " + modelVersion);
        System.out.println("Color: " + color);
    }
}

// Low-level module - concrete implementation
public class BluetoothMouse implements Mouse {
    private final String connectionType;
    private final String company;
    private final String modelVersion;
    private final String color;

    public BluetoothMouse(String connectionType, String company, String modelVersion, String color) {
        this.connectionType = connectionType;
        this.company = company;
        this.modelVersion = modelVersion;
        this.color = color;
    }

    public void getSpecifications() {
        System.out.println("===> Bluetooth Mouse");
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Company: " + company);
        System.out.println("Model Version: " + modelVersion);
        System.out.println("Color: " + color);
    }
}

// Low-level module - concrete implementation
```

```java
public class BluetoothMouse implements Mouse {
    private final String connectionType;
    private final String company;
    private final String modelVersion;
    private final String color;

    public BluetoothMouse(String connectionType, String company, String modelVersion, String color) {
        this.connectionType = connectionType;
        this.company = company;
        this.modelVersion = modelVersion;
        this.color = color;
    }

    public void getSpecifications() {
        System.out.println("===> Bluetooth Mouse");
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Company: " + company);
        System.out.println("Model Version: " + modelVersion);
        System.out.println("Color: " + color);
    }
}
```

```java
// VIOLATION OF DIP
// High-level module directly depending on low-level module
public class MacBook {
    private final WiredKeyboard keyboard;
    private final WiredMouse mouse;

    // Direct dependency on concrete class
    public MacBook(WiredKeyboard wiredKeyboard, WiredMouse wiredMouse) {
        keyboard = wiredKeyboard; // Tight coupling
        mouse = wiredMouse; // Tight coupling
    }

    public Mouse getMouse() {
        return mouse;
    }

    public Keyboard getKeyboard() {
        return keyboard;
    }
}


public class DemoViolation {
    public static void main(String[] args) {

        // create keyboard and mouse objects
        WiredKeyboard wiredKeyboard = new WiredKeyboard("USB", "Dell", "F602", "Grey");
        WiredMouse wiredMouse = new WiredMouse("USB", "Dell", "F602", "Grey");
        BluetoothKeyboard bluetoothKeyboard = new BluetoothKeyboard("Bluetooth", "Logitech", "G102", "Black");
        BluetoothMouse bluetoothMouse = new BluetoothMouse("Bluetooth", "Logitech", "G102", "Black");

        // create macbook
        MacBook macBookWithWiredParts = new MacBook(wiredKeyboard, wiredMouse);
        macBookWithWiredParts.getKeyboard().getSpecifications();
        macBookWithWiredParts.getMouse().getSpecifications();

        // create macbook with bluetooth keyboard and mouse
        // MacBook macBookWithBluetoothParts = new MacBook(bluetoothKeyboard, bluetoothKeyboard);
        // cannot create macbook with bluetooth keyboard and mouse because
```

```
        // macbook depends on wired keyboard and mouse - tight coupling - violation of DIP
    }
}
```

Problems with the Above Code

- The `MacBook` is tightly coupled to the `WiredKeyboard` and the `WiredMouse`.
- Cannot create `MacBook` objects with different parts without modifying the `MacBook` class.
- Difficult to test `MacBook` in isolation
- High-level module `MacBook` depends on low-level module `WiredKeyboard` and the `WiredMouse`.

Code Example: Follows DIP

Utility Code remains the same

```
// Following DIP
// High-level module uses abstraction
public class MacBook {
    private final Keyboard keyboard;
    private final Mouse mouse;

    // Abstraction - defines contract
    // Dependency injection through constructor
    public MacBook(Mouse mouse, Keyboard keyboard) {
        this.keyboard = keyboard; // Works with any kind of keyboard and mouse
        this.mouse = mouse;
    }

    public Mouse getMouse() {
        return mouse;
    }

    public Keyboard getKeyboard() {
        return keyboard;
    }
}

// Usage example - following DIP
public class DemoSolution {
    public static void main(String[] args) {
        // create keyboard and mouse objects
        WiredKeyboard wiredKeyboard = new WiredKeyboard("USB", "Dell", "F602", "Grey");
        WiredMouse wiredMouse = new WiredMouse("USB", "Dell", "F602", "Grey");
        BluetoothKeyboard bluetoothKeyboard = new BluetoothKeyboard("Bluetooth", "Logitech", "G102", "Black");
        BluetoothMouse bluetoothMouse = new BluetoothMouse("Bluetooth", "Logitech", "G102", "Black");

        // Can easily create macbooks with different combinations of keyboard and mouse

        // create macbook with wired keyboard and mouse
        MacBook macBookWithWiredParts = new MacBook(wiredMouse, wiredKeyboard);
        macBookWithWiredParts.getKeyboard().getSpecifications();
        macBookWithWiredParts.getMouse().getSpecifications();

        // create macbook with bluetooth keyboard and mouse
        MacBook macBookWithBluetoothParts = new MacBook(bluetoothMouse, bluetoothKeyboard);
        macBookWithBluetoothParts.getKeyboard().getSpecifications();
        macBookWithBluetoothParts.getMouse().getSpecifications();
```

```
    }
}
```

Key Benefits of the Refactored Code

- `BluetoothKeyboard` and `BluetoothMouse` depend only on the respective `Keyboard` and `Mouse` abstractions.
- Any different `MacBook` parts can be injected without changing the `MacBook`.
- Easy to mock for testing.
- New `MacBook` parts can be added without touching existing code.

Summary

The solution demonstrates true dependency inversion because the high-level module (`MacBook`) and low-level modules (`BluetoothKeyboard` , `BluetoothMouse` , `WiredKeyboard, WiredMouse`) both depend on the same abstraction (`Keyboard` and `Mouse`), and the abstraction doesn't depend on any concrete implementation details.