Single Responsibility Principle - Bad Example

```
import java.io.FileWriter;
import java.io.IOException;

class User {
    private String name;
    private String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public void saveToFile() {
        try (FileWriter writer = new FileWriter(name + ".txt")) {
            writer.write("Name: " + name + "\nEmail: " + email);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Single Responsibility Principle - Good Example

```
class UserGood {
    private String name;
    private String email;

    public UserGood(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getName() { return name; }
    public String getEmail() { return email; }
}

class UserPersistence {
    public static void saveToFile(UserGood user) {
        try (FileWriter writer = new FileWriter(user.getName() + ".txt")) {
            writer.write("Name: " + user.getName() + "\nEmail: " + user.getEmail());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Open/Closed Principle - Bad Example

```
class Rectangle {
    public int width;
    public int height;
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
}

class Circle {
```

```
    public int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
}

class AreaCalculator {
    public double calculateArea(Object shape) {
        if (shape instanceof Rectangle r) {
            return r.width * r.height;
        } else if (shape instanceof Circle c) {
            return 3.14 * c.radius * c.radius;
        }
        return 0;
    }
}
```

Open/Closed Principle - Good Example

```
abstract class Shape {
    public abstract double area();
}

class RectangleGood extends Shape {
    private double width, height;
    public RectangleGood(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double area() {
        return width * height;
    }
}

class CircleGood extends Shape {
    private double radius;
    public CircleGood(double radius) {
        this.radius = radius;
    }
    public double area() {
        return 3.14 * radius * radius;
    }
}

class AreaCalculatorGood {
    public double calculateArea(Shape shape) {
        return shape.area();
    }
}
```

Liskov Substitution Principle - Bad Example

```
class Bird {
    public void fly() {
        System.out.println("Flying");
    }
}

class Ostrich extends Bird {
    @Override
    public void fly() {
```

```
        throw new UnsupportedOperationException("Ostriches can't fly");
    }
}

class TestLSP {
    public static void makeBirdFly(Bird bird) {
        bird.fly();
    }

    public static void main(String[] args) {
        Bird sparrow = new Bird();
        Ostrich ostrich = new Ostrich();
        makeBirdFly(sparrow);
        makeBirdFly(ostrich); // throws exception
    }
}
```

Liskov Substitution Principle - Good Example

```
abstract class BirdGood {
    // Common bird functionality
}

interface FlyingBird {
    void fly();
}

class Sparrow extends BirdGood implements FlyingBird {
    public void fly() {
        System.out.println("Flying");
    }
}

class OstrichGood extends BirdGood {
    // Doesn't implement fly()
}

public class TestLSPGood {
    public static void makeBirdFly(FlyingBird bird) {
        bird.fly();
    }

    public static void main(String[] args) {
        Sparrow sparrow = new Sparrow();
        makeBirdFly(sparrow);
    }
}
```

Interface Segregation Principle - Bad Example

```
interface Worker {
    void work();
    void eat();
}

class Robot implements Worker {
    public void work() {
        System.out.println("Robot working");
    }

    public void eat() {
```

```
        // Robots don't eat
    }
}
```

Interface Segregation Principle - Good Example

```
interface Workable {
    void work();
}

interface Eatable {
    void eat();
}

class Human implements Workable, Eatable {
    public void work() {
        System.out.println("Human working");
    }

    public void eat() {
        System.out.println("Human eating");
    }
}

class RobotGood implements Workable {
    public void work() {
        System.out.println("Robot working");
    }
}
```

Dependency Inversion Principle - Bad Example

```
class MySQLDatabase {
    public void connect() {
        System.out.println("Connecting to MySQL");
    }
}

class PasswordReminder {
    private MySQLDatabase db;

    public PasswordReminder() {
        this.db = new MySQLDatabase();
    }

    public void remind() {
        db.connect();
        System.out.println("Reminding password");
    }
}
```

Dependency Inversion Principle - Good Example

```
interface Database {
    void connect();
}

class MySQLDatabaseGood implements Database {
    public void connect() {
        System.out.println("Connecting to MySQL");
```

```
    }
}

class PasswordReminderGood {
    private Database db;

    public PasswordReminderGood(Database db) {
        this.db = db;
    }

    public void remind() {
        db.connect();
        System.out.println("Reminding password");
    }
}

public class Main {
    public static void main(String[] args) {
        Database mysql = new MySQLDatabaseGood();
        PasswordReminderGood reminder = new PasswordReminderGood(mysql);
        reminder.remind();
    }
}
```