SOLID Principles in Java with Real-Time Examples

1. Single Responsibility Principle (SRP)

Definition: A class should have only one reason to change.

Bad Example:

```
public class Employee {
    public String getEmployeeDetails() {
        return "Employee Data";
    }
    public void generatePDFReport() {
        // Generates report
    }
}
```

Good Example:

```
public class Employee {
    public String getEmployeeDetails() {
        return "Employee Data";
    }
}
public class ReportGenerator {
    public void generatePDFReport(Employee emp) {
        // Generates report
    }
}
```

SOLID Principles in Java with Real-Time Examples

2. Open/Closed Principle (OCP)

Definition: Software entities should be open for extension, but closed for modification.

Bad Example:

```
public class DiscountCalculator {
    public double calculate(String customerType) {
        if (customerType.equals("Premium")) return 0.2;
        else if (customerType.equals("Regular")) return 0.1;
        return 0;
    }
}
```

Good Example:

```
public interface DiscountStrategy {
    double getDiscount();
}
public class PremiumCustomer implements DiscountStrategy {
    public double getDiscount() { return 0.2; }
}
public class RegularCustomer implements DiscountStrategy {
    public double getDiscount() { return 0.1; }
}
public class DiscountCalculator {
    public double calculate(DiscountStrategy strategy) {
        return strategy.getDiscount();
```

```
    }
}
```

3. Liskov Substitution Principle (LSP)

Definition: Subtypes must be substitutable for their base types.

Bad Example:

```
public class Bird {
    public void fly() {
        System.out.println("Flying");
    }
}
public class Ostrich extends Bird {
    public void fly() {
        throw new UnsupportedOperationException("Ostrich can't fly");
    }
}
```

Good Example:

```
public interface Bird {}
public interface FlyingBird extends Bird {
    void fly();
}
public class Sparrow implements FlyingBird {
```

```
    public void fly() {
        System.out.println("Flying");
    }
}
public class Ostrich implements Bird {
    // No fly method
}
```

4. Interface Segregation Principle (ISP)

Definition: No client should be forced to depend on interfaces it does not use.

Bad Example:

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

Good Example:

```
public interface Workable {
    void work();
}
public interface Eatable {
    void eat();
}
```

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

5. Dependency Inversion Principle (DIP)

Definition: High-level modules should not depend on low-level modules; both should depend on abstractions.

Bad Example:

```
public class MySQLDatabase {
    public void connect() {
        System.out.println("Connected to MySQL");
    }
}
public class App {
    private MySQLDatabase db = new MySQLDatabase();
    public void start() {
        db.connect();
    }
}
```

Good Example:

```
public interface Database {
```

```
    void connect();
}
public class MySQLDatabase implements Database {
    public void connect() {
        System.out.println("Connected to MySQL");
    }
}
public class App {
    private Database db;
    public App(Database db) {
        this.db = db;
    }
    public void start() {
        db.connect();
    }
}
```