SOLID Principles in Java - Explained with Examples

1. S - Single Responsibility Principle (SRP)

Definition: A class should have only one reason to change - meaning it should do one job only.

Bad Example:

```
class Invoice {
    public void calculateTotal() { }
    public void printInvoice() { }
    public void saveToDatabase() { }
}
```

Good Example:

```
class Invoice { public void calculateTotal() { } }
class InvoicePrinter { public void printInvoice(Invoice invoice) { } }
class InvoiceRepository { public void save(Invoice invoice) { } }
```

2. O - Open/Closed Principle (OCP)

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

Bad Example:

```
class Shape { String type; }
class AreaCalculator {
    public double calculateArea(Shape shape) {
        if (shape.type.equals("circle")) { }
        else if (shape.type.equals("rectangle")) { }
    }
}
```

Good Example:

```
interface Shape { double calculateArea(); }
class Circle implements Shape { public double calculateArea() { return 0; } }
class Rectangle implements Shape { public double calculateArea() { return 0; } }
```

3. L - Liskov Substitution Principle (LSP)

Definition: Subclasses should be substitutable for their base classes without breaking the program.

Bad Example:

```
class Bird { public void fly() { } }
class Ostrich extends Bird { public void fly() { throw new UnsupportedOperationException(); } }
```

Good Example:

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

4. I - Interface Segregation Principle (ISP)

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

Bad Example:

```
interface Machine { void print(); void scan(); void fax(); }
class Printer implements Machine { public void print() { } public void scan() { } public void fax() { } }
```

Good Example:

```
interface Printer { void print(); }
interface Scanner { void scan(); }
interface Fax { void fax(); }
class SimplePrinter implements Printer { public void print() { } }
```

5. D - Dependency Inversion Principle (DIP)

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

Bad Example:

```
class MySQLDatabase { public void connect() { } }
class Application { private MySQLDatabase db = new MySQLDatabase(); public void start() { db.connect(); } }
```

Good Example:

```
interface Database { void connect(); }
class MySQLDatabase implements Database { public void connect() { } }
class PostgreSQLDatabase implements Database { public void connect() { } }
class Application { private Database db; public Application(Database db) { this.db = db; } public void start() { db.connect(); } }
```

Summary Table:

SRP	One class -> one responsibility
OCP	Extend without modifying
LSP	Subclasses must be replaceable for base classes
ISP	Many small interfaces > one big interface
DIP	Depend on abstractions, not concrete classes