SOLID Principles

Solid principles are a set of five object-oriented programming principles that make software more maintainable, scalable, and flexible.

S – Single Responsibility Principle

A class should have/do only one job.

Example:

```
class Invoice {
    public void calculateTotal() {
        // logic to calculate invoice total
    }
}

class InvoiceRepository {
    public void saveToDatabase(Invoice invoice) {
        // save logic
    }
}

class InvoicePrinter {
    public void printInvoice(Invoice invoice) {
        // print logic
    }
}
```

O – Open/Closed Principle

Software should be open for extensions but closed for modifications.

Example:

```
interface Shape {
    double area();
}

class Circle implements Shape {
    double radius;
    public Circle(double r) { radius = r; }
    public double area() { return Math.PI * radius * radius; }
}

class Rectangle implements Shape {
    double length, width;
    public Rectangle(double l, double w) { length = l; width = w; }
    public double area() { return length * width; }
}

class AreaCalculator {
    public double calculate(Shape shape) {
        return shape.area();
    }
```

L – Liskov Substitution Principle

Objects of a superclass should be replaceable with objects of subclasses without breaking the application.

Example:

```
interface Bird {}

interface FlyableBird extends Bird {
    void fly();
}

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

class Ostrich implements Bird {
    // no fly() here, because ostriches don't fly
}
```

I – Interface Segregation Principle

A client should not be forced to depend on methods it does not use.

Example:
(If ostrich extends FlyableBird in above example its violation)

D – Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Example (Violation):

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

class UserService {
    private MySQLDatabase db = new MySQLDatabase();
    public void saveUser() {
        db.connect();
        System.out.println("User saved");
    }
}
```

Example (Correct):

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

class MySQLDatabase implements Database {
    public void connect() {
```

```
        System.out.println("Connected to MySQL");
    }
}

class MongoDBDatabase implements Database {
    public void connect() {
        System.out.println("Connected to MongoDB");
    }
}

class UserService {
    private Database db;
    public UserService(Database db) {
        this.db = db;
    }
    public void saveUser() {
        db.connect();
        System.out.println("User saved");
    }
}
```