SOLID PRINCIPLES

Your Path to Cleaner and Maintainable Code

CloudnLoud Community

@jahnavivetukuri

S

Single Responsibility Principle (SRP)

O

Open Closed Principle (OCP)

L

Liskov Substitution Principle (LSP)

I

Interface Segregation Principle (ISP)

D

Dependency Inversion Principle (DIP)

SINGLE RESPONSIBILITY PRINCIPLE

A class should have only one reason to change, meaning it should have only one responsibility

@jahnavivetukuri

DO'S

```
class EmailSender {
 public void sendEmail(String to, String subject, String body) {
     // Send an email
 }
}

class Logger {
 public void log(String message) {
     // Log the message
 }
}
```

DON'TS

```
//Example of violating SRP
class User {
 public void createUser() {
     // Create a user
 }

 public void sendEmail(String to, String subject, String body) {
     // Send an email
 }
}
```

OPEN-CLOSED PRINCIPLE (OCP)

Entities (classes, modules, functions) should be open for extension but closed for modification

DO'S

```java
//Example of adhering to OCP
interface Shape {
 double area();
}

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

class Rectangle implements Shape {
 private double width;
 private double height;
 public Rectangle(double width, double height) {
     this.width = width;
     this.height = height;
 }
 public double area() {
     return width * height;
 }
}
```

❌ DON'TS

```
//Example of violating OCP
class AreaCalculator {
 public double calculateArea(Object shape) {
     if (shape instanceof Circle) {
         Circle circle = (Circle) shape;
         return Math.PI * circle.getRadius() * circle.getRadius();
     } else if (shape instanceof Rectangle) {
         Rectangle rectangle = (Rectangle) shape;
         return rectangle.getWidth() * rectangle.getHeight();
     }
     return 0;
 }
}
```

LISKOV SUBSTITUTION PRINCIPLE (LSP)

Subtypes must be substitutable for their base types without altering the correctness of the program.

@jahnavivetukuri

DO'S

```
//Example of adhering to LSP
class Bird {
 void fly() {
     // All birds can fly
 }
}

class Sparrow extends Bird {
 // Sparrows can fly like other birds
}

class Ostrich extends Bird {
 void fly() {
     throw new UnsupportedOperationException("Ostriches cannot fly");
 }
}
```

DON'TS

```
//Example of violating LSP
class Penguin extends Bird {
 // Penguins are birds but cannot fly
}
```

INTERFACE SEGREGATION PRINCIPLE (ISP)

A class should not be forced to implement interfaces it does not use, and clients should not be forced to depend on interfaces they do not use.

@jahnavivetukuri

DO'S

```
//Example of adhering to ISP
interface Worker {
 void work();
}
interface Eater {
 void eat();
}
class Engineer implements Worker {
 public void work() {
     // Engineer's work
 }
}
class Chef implements Worker, Eater {
 public void work() {
     // Chef's work
}
 public void eat() {
     // Chef's eating habits
 }
}
```

DON'TS

```
//Example of violating ISP
interface Worker {
 void work();
 void eat();
}
class Engineer implements Worker {
 public void work() {
     // Engineer's work
 }
 public void eat() {
     // Engineer's eating habits
 }
}
```

DEPENDENCY INVERSION PRINCIPLE (DIP)

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

@jahnavivetukuri


```java
//Example of adhering to DIP
interface Switchable {
 void turnOn();
 void turnOff();
}

class LightBulb implements Switchable {
 public void turnOn() {
     // Turn on the light bulb
 }
 public void turnOff() {
     // Turn off the light bulb
 }
}
class Fan implements Switchable {
 public void turnOn() {
     // Turn on the fan
 }
 public void turnOff() {
     // Turn off the fan
 }
}
class Switch {
 private Switchable device;
 public void connect(Switchable device) {
     this.device = device;
 }
 public void turnOn() {
     device.turnOn();
 }
 public void turnOff() {
     device.turnOff();
 }
}
```

DON'TS

```
//Example of violating DIP
class Switch {
 private LightBulb bulb;

 public void connect(LightBulb bulb) {
     this.bulb = bulb;
 }

 public void turnOn() {
     bulb.turnOn();
 }

 public void turnOff() {
     bulb.turnOff();
 }
}
```

These SOLID principles help create maintainable, flexible, and understandable code by promoting good design practices.

@jahnavivetukuri

REPOST IF USEFUL

FOLLOW FOR MORE

CloudnLoud

Community

@jahnavivetukuri