Single responsibility Principle

public class User {

private String username;

private String email;

public User(String username, String email) {

this.username = username;

this.email = email;

}

// Getters

public String getUsername() {

return username;

}

public String getEmail() {

return email;

}

}

public class UserPersistence {

public void saveUser(User user) {

// For demonstration, we'll just print to console.

System.out.println("Saving user to database...");

System.out.println("Username: " + user.getUsername());

System.out.println("Email: " + user.getEmail());

}

}

**Open/Closed Principle**

**public interface Notification {**

**void send(String message);**

**}**

**public class EmailNotification implements Notification {**

**@Override**

**public void send(String message) {**

**System.out.println("Sending EMAIL: " + message);**

**}**

**}**

**public class SMSNotification implements Notification {**

**@Override**

**public void send(String message) {**

**System.out.println("Sending SMS: " + message);**

**}**

**}**

**public class Notifier {**

**private final Notification notification;**

**public Notifier(Notification notification) {**

**this.notification = notification;**

**}**

**public void alert(String message) {**

**notification.send(message);**

**}**

**}**

**L — iskov Substitution Principle**

public class Bird {

public void layEgg() {

System.out.println("Bird lays an egg.");

}

}

public class FlyingBird extends Bird {

public void fly() {

System.out.println("FlyingBird is flying.");

}

}

public class Ostrich extends Bird {

public void run() {

System.out.println("Ostrich is running.");

}

}

public class Sparrow extends FlyingBird {

@Override

public void fly() {

System.out.println("Sparrow is flying fast.");

}

}

**I — nterface Segregation Principle**

public interface Workable {

void work();

}

public interface Eatable {

void eat();

}

public interface Sleepable {

void sleep();

}

public class HumanWorker implements Workable, Eatable, Sleepable {

public void work() {

System.out.println("Human working...");

}

public void eat() {

System.out.println("Human eating...");

}

public void sleep() {

System.out.println("Human sleeping...");

}

}

public class RobotWorker implements Workable {

public void work() {

System.out.println("Robot working...");

}

}

**D — ependency Inversion Principle**

public class App {

private final NotificationService service;

// App depends on abstraction, not a specific class

public App(NotificationService service) {

this.service = service;

}

public void notifyUser(String message) {

service.send(message);

}

}

public interface NotificationService {

void send(String message);

}

public class EmailService implements NotificationService {

public void send(String message) {

System.out.println("Email sent: " + message);

}

}

public class SMSService implements NotificationService {

public void send(String message) {

System.out.println("SMS sent: " + message);

}

}

public class LoggerService implements NotificationService {

public void send(String message) {

System.out.println("Log entry: " + message);

}

}