* What is the dependency inversion principle? Explain how it contributes to the more testable code.

Ans: Dependency inversion principle is one of the five solid principle of object-oreinted programming and software design.

1.Dependency Substitution: Easily replace real dependencies with mocks or stubs during testing.

2.Supports Isolation: Enables testing a module without involving real external systems (e.g., databases, APIs).

3.Facilitates Dependency Injection: Allows injecting test-specific implementations (e.g., mocks) without modifying the class.

4.Reduces Coupling: Decouples high-level modules from low-level details, preventing cascading changes during testing.

5.Simplifies Mocking: Makes it easier to mock complex systems or external services for testing.

* Describe the scenario where applying the Open-Closed Principle leads to improved code quality.

Ans:

Applying the Open-Closed Principle (OCP) improves code quality when you need to add new features without modifying existing code. For example, in a payment processing system, adding a new payment method (e.g., PayPal) can be done by creating a new class that implements a common interface without altering existing payment classes. This reduces the risk of introducing bugs in tested code and enhances flexibility. OCP ensures the system is extensible and maintainable over time.

* Explain the scenario where the Interface Segregation Principle was beneficial.
* Examine the following code.

public class Report {

public void generateReport() {

// generate report logic

}

public void exportToPDF() {

// export report to PDF logic

}

public void exportToExcel() {

// export report to Excel logic

}

}

Which principle is violated in the code among Single Responsibility, Open Closed, Interface Segregation, and Dependency Inversion Principles? Explain in detail.

* Can you provide an example of how to design an online payment processing system while adhering to the SOLID principles? Please explain how each principle can be applied in the context of this system and illustrate with code or a conceptual overview. Let’s assume we have payment types like CreditCardPayment, PayPalPayment, Esewa, and Khalti. Each of these payments should have a method of transferring the amount.
* Examine the following code.

public class Shape {

public void drawCircle() {

// drawing circle logic

}

public void drawSquare() {

// drawing square logic

}

}

You want to add more shapes (e.g., triangles, rectangles) without modifying the existing Shape class. Which design change would adhere to the Open-Closed Principle?

* Examine the following code.

public class Duck {

public void swim() {

System.out.println("Swimming");

}

public void quack() {

System.out.println("Quacking");

}

}

public class WoodenDuck extends Duck {

@Override

public void quack() {

throw new UnsupportedOperationException("Wooden ducks don't quack");

}

}

Which principle is violated in the above code among Open Closed, Single Responsibility, Liskov, and Interface Segregation Principle? Explain in detail. Also, update the above code base to resolve the issue.

* Examine the following code.

public interface PaymentMethod {

void processPayment();

}

public class PaypalPayment implements PaymentMethod {

@Override

public void processPayment() {

System.out.println("Processing PayPal payment");

}

}

public class OrderService {

private PaymentMethod paymentMethod;

public OrderService(PaymentMethod paymentMethod) {

this.paymentMethod = paymentMethod;

}

public void makePayment() {

paymentMethod.processPayment();

}

}

Which solid principle is being used in above? Explain in detail.