

Object Oriented Design

Table of contents

| | |
|------------------------------------|----------|
| 1 Multiple Choice Questions | 1 |
| 2 Short Answer Questions | 2 |
| 3 Code-Based Questions | 2 |
| 4 Conceptual Questions | 4 |

1 Multiple Choice Questions

1. What does Modularity refer to in Object-Oriented Analysis and Design?
 - A) The ability to change the functionality of an application without impacting the user interface.
 - B) The degree to which a system is composed of discrete components.
 - C) The process of creating objects from classes.
 - D) The ability of different classes to be treated as instances of the same class.
2. In the context of software design, what does Low Coupling imply?
 - A) High dependency between different modules of a system.
 - B) Minimal impact on other components when a component is changed.
 - C) The ability of a system to handle unexpected inputs gracefully.
 - D) The degree to which a system can be used in more than one application.
3. Which principle is best demonstrated by creating a class that does not represent a real-world entity but aids in design?
 - A) Information Expert

- B) Polymorphism
 - C) Pure Fabrication
 - D) High Cohesion
4. In Object-Oriented Design, what is a 'Use Case' primarily used for?
- A) Defining the structure of different classes in a system.
 - B) Describing how users interact with the system to achieve specific goals.
 - C) Illustrating the relationships and dependencies among different classes.
 - D) Outlining the sequence of messages between objects in a particular interaction.

2 Short Answer Questions

1. Explain the concept of 'Aggregation' in object-oriented design with an example.
2. Describe the difference between 'Functional Requirements' and 'Non-Functional Requirements' in software development.
3. Illustrate the concept of 'Polymorphism' with a code example in Java.
4. Discuss the role and importance of 'Use Cases' in understanding user interactions with a system.
5. Give an example of applying the 'Law of Demeter' in a Java class.

3 Code-Based Questions

1. Refactor the given code snippet to improve its adherence to the KISS principle.

```
public class StudentGradeCalculator {  
  
    public double calculateFinalGrade(int[] assignmentScores,  
                                     int midtermScore,  
                                     int finalExamScore) {  
        double assignmentAverage = 0;  
  
        for (int i = 0; i < assignmentScores.length; i++) {  
            assignmentAverage += assignmentScores[i];  
        }  
        assignmentAverage /= assignmentScores.length;  
    }  
}
```

```

    double decimalMidterm = midtermScore / 100.0;
    double decimalFinalExam = finalExamScore / 100.0;
    double decimalAssignments = assignmentAverage / 100.0;

    double weightedMidterm = decimalMidterm * 30;
    double weightedFinalExam = decimalFinalExam * 30;
    double weightedAssignments = decimalAssignments * 40;

    double finalGrade = weightedMidterm + weightedFinalExam + weightedAssignments;

    finalGrade *= 100;

    return finalGrade;
}
}

```

2. Given a high coupling scenario in the following Java application, suggest improvements to reduce coupling and rewrite the code.

```

// EmailNotification.java
public class EmailNotification {
    public void sendEmail(String message, String recipient) {
        // Logic to send an email
        System.out.println("Sending email to " + recipient + ": " + message);
    }
}

// OrderManager.java
public class OrderManager {
    private EmailNotification emailNotification;

    public OrderManager() {
        this.emailNotification = new EmailNotification();
    }

    public void processOrder(String orderDetails, String customerEmail) {
        // Logic to process the order
        System.out.println("Order processed: " + orderDetails);

        emailNotification.sendEmail("Your order has been processed.", customerEmail);
    }
}

```

```
}  
  
// Main.java  
public class Main {  
    public static void main(String[] args) {  
        OrderManager orderManager = new OrderManager();  
        orderManager.processOrder("Book: Java Programming", "customer@example.com");  
    }  
}
```

4 Conceptual Questions

1. Given a Shape interface with a method draw(), and classes Circle and Square implementing Shape, explain how adding a new Triangle class without modifying existing code aligns with the Open/Closed Principle (OCP).