Design Principles Results for Christine En-Tse Cheng

Correct answers are hidden.

Submitted Dec 13 at 6:09p.m.

Question 1

1 / 1 pts

In terms of coupling and cohesion, what is the goal in software design?

- high coupling and high cohesion
- low coupling and low cohesion
- low coupling and high cohesion
- high coupling and low cohesion

Unanswered

Question 2

0 / 1 pts

Which of the following best describes the SRP?

- A class should only contain one method to ensure simplicity.
- A class should have only one reason to change, typically driven by a single stakeholder or tightly related group.
- A class should only perform one task, such as either reading input or displaying output.
- A class should only be used by one other class to reduce dependencies.

Unanswered

Question 3

0 / 1 pts

The Open/Closed Principle states that software entities should be:

- Open for modification, closed for extension
- Closed for modification, open for extension
- Open for both modification and extension
- Closed for both modification and extension

Unanswered

Question 4

0 / 1 pts

Which of the following best describes the Liskov Substitution Principle (LSP)?

- Subclasses should override all methods of their superclass
- A subclass should be usable anywhere its superclass is expected, without causing errors or changing expected behavior.
- A class should only implement interfaces it actually uses.
- High-level modules should not depend on low-level modules.

Unanswered

Question 5

0 / 1 pts

Which SOLID principle states that a class should not be forced to implement extra methods that it does not use?

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Unanswered

Question 6

0 / 1 pts

Which SOLID principle addresses the issue of designing interfaces that are too large and overly general? Choose the best answer.

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Unanswered

Question 7

0 / 1 pts

The Dependency Inversion Principle suggests that:

- High-level modules should depend on low-level modules
- Abstractions should depend on details
- Low-level modules should depend on high-level modules
- Both high-level and low-level modules should depend on abstractions

Unanswered

Question 8

0 / 1 pts

Consider an application that processes user data that is collected from external providers. This data is read into the system, transformed, and then saved in a standardized format for use by a team of data analysts.

How could the code below be refactored to more closely adhere to the Single Responsibility Principle (SRP)?

```
class FileParser {
    public String read(String filename) {
        // File reading logic
    }

    public void save(String data) {
        // File saving logic
    }
}
```

```
class UserManager {
    private FileParser parser;

    public UserManager() {
        parser = new FileParser();
    }

    public void readUserData() {
        String userData = parser.read("users.txt");
        // Process user data
    }
```

```
}
```

- Combine the `FileParser` and `UserManager` classes into a single class
- Move the `readUserData` method to the `FileParser` class
- Split the `FileParser` class into separate classes for reading and saving files
- Rename the `FileParser` class to more clearly indicate its two responsibilities.

Unanswered

Question 9

0 / 1 pts

Our goal is to refactor* the following code to more closely adhere to the Dependency Inversion Principle (DIP):

> *Refactoring refers to the act of changing the ***structure*** of the code — without changing its functionality.

```
class DatabaseConnection {
    public void connect() {
        // Connect to the database
    }
}

class UserManager {
    private DatabaseConnection dbConnection;

    public UserManager() {
        this.dbConnection = new DatabaseConnection();
    }

    public User getUser(int userId) {
        return dbConnection.query("SELECT * FROM users WHERE id = " + userId);
    }
}
```

Which approach below best describes an attempt to successfully do such a refactoring?

- Use a different database technology
- Introduce an interface for database connections and update the `UserManager` class to use the interface.
- Remove the `DatabaseConnection` class from the program.
- Encapsulate the database connection details within `UserManager`