Assignment:

Secure Coding Practice:

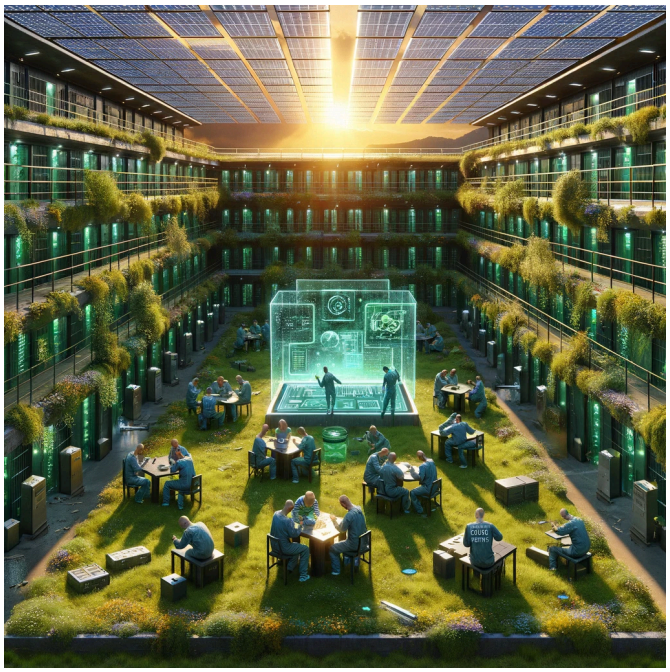

Image Generated by Dalle-E 2 from chat GPT

Prepared for

Kokub Sultan

Prepared by Group 6

Valentine Jingwa, Jean-Pierre Nde-Forgwang

Security System, Cohort E

School of Advance Digital Technology

SAIT

31 January 2024

Composition vs. Inheritance

Define Composition:

Composition is a design technique in which a class is built with references to other classes in order to accomplish functionality. This method enables a class to delegate part of its tasks to the mentioned classes [1].

Define Inheritance:

Inheritance is a mechanism for deriving a new class from an existing one. The derived class inherits all of the original class's features and can add or modify them [2]

Advantages and Disadvantages of Composition:

Advantages:

- Enhances flexibility and reusability.
- Easier to change behavior on runtime.
- Reduces tight coupling and simplifies code dependencies.

Disadvantages:

- Can lead to complex designs.
- Might require more boilerplate to set up relationships between classes.

Advantages and Disadvantages of Inheritance:

Advantages:

- Easy to implement.
- Promotes code reusability.
- Simplifies code hierarchy.

Disadvantages:

- Can lead to tightly coupled code.
- Risk of overloading the base class with too much functionality.
- Difficulties in modifying inherited behavior.

Code Example:

Composition:

Composition: [1]

```
// Java program to illustrate
// The concept of Composition

import java.io.*;
import java.util.*;

// class book
class Book {

        public String title;
        public String author;

        Book(String title, String author)
        {

                this.title = title;
                this.author = author;
        }
}

// Library class contains
// list of books.
class Library {

        // reference to refer to the list of books.
        private final List<Book> books;

        Library(List<Book> books)
        {
                this.books = books;
        }

        public List<Book> getTotalBooksInLibrary()
        {

                return books;
        }
}
```

```
// main method
class GFG {
        public static void main(String[] args)
        {

                // Creating the Objects of Book class.
                Book b1 = new Book(
                        "EffectiveJ Java",
                        "Joshua Bloch");
                Book b2 = new Book(
                        "Thinking in Java",
                        "Bruce Eckel");
                Book b3 = new Book(
                        "Java: The Complete Reference",
                        "Herbert Schildt");

                // Creating the list which contains the
                // no. of books.
                List<Book> books = new ArrayList<Book>();
                books.add(b1);
                books.add(b2);
                books.add(b3);

                Library library = new Library(books);

                List<Book> bks = library.getTotalBooksInLibrary();
                for (Book bk : bks) {

                        System.out.println("Title : "
                                                        + bk.title + " and "
                                                        + " Author : "
                                                        + bk.author);

                }
        }
}
```

Inheritance:

Inheritance: [2]

```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Analyzing Impact on Software Design:

- Inheritance creates a strong hierarchical relationship but can lead to rigid structures. Composition, on the other hand, promotes flexibility and adaptability, allowing for more dynamic and modular design.

Immutable vs. Mutable Objects

Define Immutable:

- An Immutable object is an object whose state cannot be modified after it is created [3].

Define Mutable Object:

- A Mutable object can be changed after it is created [3].

Advantages and Disadvantages of Immutable:

Advantages

- Simplifies understanding and working with code.
- Safer in concurrent environments.
- Can improve performance due to caching.

Disadvantages

- It may lead to increased memory use.
- Can be less efficient for large objects or complex operations.

Advantages and Disadvantages of Mutable Objects

Advantages:

- More memory efficient in certain scenarios.
- Often more intuitive for managing changing states.

Disadvantages:

- Can introduce bugs in concurrent environments.
- Harder to reason about and debug.

Code Example

Immutable: [4]

```
var aString = "This is a string";
var bString = aString;
//now we alter aString
aString = "The string has changed";

console.log(aString); //The string has changed
console.log(bString); //This is a string
console.log(aString === bString); //false
```

Mutable: [4]

```
var car = {
    color: 'red',
    tyres: '4'
}

var anotherCar = car; //here we assign the value of car to anotherCar.

car.color = 'blue'; //we alter a value in the original object

console.log(anotherCar.color); //this shows blue because it is referencing
the memory location of the car object.
console.log(car === anotherCar) //true (because comparison is by reference)
```

Preferred Scenarios

Immutable

- Preferred in multi-threaded environments for safety and predictability.

Mutable

- Used when there's a need for efficient memory and performance, like in real-time processing.

Conclusion

In software design, the choice between Composition and Inheritance, or Immutable and Mutable objects, largely depends on the specific requirements and constraints of the project. Composition offers more flexibility and modularity but can lead to complex structures, while Inheritance provides a straightforward, hierarchical model that can become rigid. Similarly, immutable objects are safer and easier to work with in concurrent scenarios, but mutable objects are more memory and performance-efficient

in certain contexts. Balancing these concepts effectively is key to crafting robust, efficient, and maintainable software.

Reference:

[1] GeeksforGeeks, "Difference Between Inheritance and Composition in Java." Available: https://www.geeksforgeeks.org/difference-between-inheritance-and-composition-in-java/. [Accessed: Jan. 31, 2024].
[2] W3Schools, "Java Inheritance." Available: https://www.w3schools.com/java/java_inheritance.asp. [Accessed: Jan. 31, 2024].
[3] JavaTpoint, "Mutable and Immutable in Java." Available: https://www.javatpoint.com/mutable-and-immutable-in-java. [Accessed: Jan. 31, 2024].
[4] How To Create Apps, "Mutable and Immutable Types in JavaScript with Examples." Available: https://howtocreateapps.com/mutable-and-immutable-types-in-javascript-with-examples/. [Accessed: Jan. 31, 2024].