**Howard University**

**College of Engineering and Architecture**

**Department of Electrical Engineering & Computer Science**

**Large Scale Programming**

**Fall 2024**

**Midterm Exam – Part I**

October 17, 2024

**Part I: 50 pts. (essay questions)**

**Instructions:**

* **Submit completed exam to your github repository under package org.howard.edu.lsp.midterm**

**Verify that the commit(s) completed successfully**

* **OPEN BOOK, OPEN NOTES. THERE IS NO COLLABORATION ON THIS EXAM**
* **Site any and ALL references for anything obtained off the internet.**

**Question 1. (10 pts.)**

You are reviewing the following class in a Fitness Tracking System. The class is intended to manage various aspects of a user’s fitness activities.



**Tasks:**

1. Analyze the UserFitness class. Explain why this class has **low cohesion.** (5 pts.)

The UserFitness class has low cohesion because it has unrelated functionalities all within the same class. Since the tracking workouts and nutrition management attributes are not working towards the same purpose, this results in low cohesion.

1. Describe, in general, how you would split the UserFitness class into **more cohesive classes**. Your answer should not be a complete design. (5 pts.)

I would split the classes by the purpose and role of each attribute. Creating a class specifically for Workout Management and Nutrition Management along with a class that stores user specific data could also boost cohesion for the program.

**Question 2. (5 pts.)**

**Given the following, answer the below questions.**

A university manages its students using a program that has a class Student with subclasses FirstYear, SecondYear, and ThirdYear for year-specific state and behaviors. The program has a List that contains all Students.

Should Student be a class, an abstract class or an interface? Explain your answer

An abstract class because it allows for you to provide shared functionality along with specialized behaviors such as their year and other responsibilities through the creation of subclasses.

1. At the end of each year, the third-year students graduate and must be removed. This is done by passing the list to the following method:

void removeThirdYears(List<Students> students) {

for (Student student : students) {

try {

ThirdYear = (ThirdYear) student;

students.remove(thirdyear);

}

}

}

Why is the above method flawed? Hint: Another Arthur Riel heuristic (5 pts.)

The method above is flawed because you are making direct changes to a collection being iterated over. This violates Riel’s heuristic 5.1 as the program is modifying the list within the same loop in which it is being iterated over.

<https://oowisdom.csse.canterbury.ac.nz/index.php/Riel%27s_heuristics>

**Question 3. (5 pts.)**

**Given the following, answer the below questions.**

public class BankAccount {

public String accountHolderName;

public String accountNumber;

public double balance;

public double interestRate;

public BankAccount(

String accountHolderName,

String accountNumber,

double initialDeposit,

double interestRate) {

this.accountHolderName = accountHolderName;

this.accountNumber = accountNumber;

this.balance = initialDeposit;

this.interestRate = interestRate;

}

// Exposed implementation for adding interest

public void addInterest() {

double interest = balance \* (interestRate / 100);

balance += interest;

}

// Exposed implementation for deposits

public void deposit(double amount) {

balance += amount;

}

// Exposed implementation for withdrawals

public void withdraw(double amount) {

balance -= amount;

}

}

1. Is BankAccount well-encapsulated? Why or why not? If so, describe what changes you would make to the class. Otherwise, state nothing. (5 pts.)

BankAccount is not well encapsulated as all of the attributes within the class are public which allows direct access to these from outside of the class. This class also does not require validation in any sense within the deposit and withdraw functions. Adding getters and setters with input validation in terms of the balance within the accounts being accessed could create a better use of encapsulation. Also, making the attributes to private would cause a restriction of access, increasing the encapsulation.

<https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/>

**Question 4. (10 pts.)**

**Given the following, answer the below questions.**

**Given:**

A car manufacturer uses Java software to track current vehicles being built. The UML diagram below shows an excerpt of the current software structure. You should assume the presence of other appropriate fields and methods

**Car**

getTrimLevel()

**Engine**

accelerate()

getFuelLevel()

**Base**

**Sports**

**Luxury**

**Electric**

**Petrol**

Each car can be built to one of three trim levels: Base, Luxury or Sport. They can also be configured with an electric or petrol engine. At various points in the manufacturing process the customer can choose to change the trim level.

**Task:**

1. Explain in detail why the current structure does not support this. (5 pts.)

This structure does not support this because the diagram shows trim level tightly coupled to the Car Class. This causes the maintenance of the code to increase in difficulty as each time a new trim level must be assigned, the entire hierarchy would have to modified in order to do so.

1. Describe how to refactor the structure to allow trim-level change for a car to dynamically change. Hint: How would you modify Car to use composition to solve the problem? (5 pts.)

Creating a TrimLevel interface and referencing an object of the interface within the Car class will allow for the trim level to be changed without having to create a new object to do so.

**Question 5. (5 pts.)**

Suppose you are about to code a particular class and find that you have written another class with nearly the functionality that you now are developing. This means that the two classes in large parts contain the same code, but in smaller parts have different code. Explain in detail why this may be a problem and how to solve it. Give your answer in terms of Arthur Riel’s design heuristics.

This may be a problem in reference to Riel’s DRY principle in which the code duplication across classes will increase the maintenance of the code, raising the possibility of errors. Riel promotes high cohesion and low coupling in which doing this would result in the latter, making the system more difficult to extend later on. A solution would be to use either inheritance or composition to create a common superclass with shared behaviors rather than duplication the code.

<https://oowisdom.csse.canterbury.ac.nz/index.php/Riel%27s_heuristics>

**Question 6. (5 pts.)**

Suppose we need to store instances of the classes Car and BankAccount in a database. To obtain low coupling it may seem as if the functionality to save an instance should be in precisely those classes, since they have the information that shall be saved. Why is it unwise to place the database calls there?

Placing database class inside the Car and BankAccount classes would violate object-oriented design principles while also increasing coupling, making the system difficult to maintain and test. The specific principle being violated by doing so is the Single Responsibility Principle by giving the the classes two responsibilities, assigning it to both the management of more than one task. Along with the low coupling principles, tightly coupling these instances would be inefficient and could be prone to errors.

<https://stackify.com/solid-design-principles/>

**Question 7. (5 pts.)**

Given a set of requirements, what conditions must be met to make it appropriate to create a new class hierarchy using inheritance? Neither superclass nor subclass exist before, the question is about designing a new class hierarchy.

Indicators that inheritance is appropriate would be a clear hierarchical structure such that each distinct object is a more specific subclass of another. A clear and distinct “is-a” relationship must be present before deciding utilize inheritance. If multiple classes also show signs of important shared attributes and functionalities, defining them in a superclass can reduce the redundancy in your code.

<https://stackoverflow.com/questions/3351666/why-use-inheritance-at-all>

**Question 8. (2.5 pts.)**

Describe the difference between inheritance and composition why would you choose one over the other.

Inheritance is an “is-a” relationship and tightly coupled while composition is referred to as a “has-a” relationship and is loosely coupled. Composition does not extend the class to reuse the fields and method unlike inheritance. It also allows you to test the implementation of the classes in use independent of the parent or child class. Composition is preferred over inheritance as it provides greater design flexibility with less issues pertaining how tightly coupled inheritance hierarchies are.

<https://www.geeksforgeeks.org/difference-between-inheritance-and-composition-in-java>/

**Question 9. (2.5 pts.)**

Explain the meaning of cohesion and describe whether low or high cohesion is better.

Cohesion refers to the degree to which elements within a module work together to fulfill a single purpose. High cohesion is considered to be better because it shows that the elements within the module are perform a focused task, which increases the readability of the code along with making it easier to maintain and modify as needed.

<https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/>