**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:**

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

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

1. **OPEN BOOK, OPEN NOTES. THERE IS NO COLLABORATION ON THIS EXAM**
2. **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.)

* **This class is low cohesion because the class does a variety of actions. There are multiple responsibilities for managing the user’s information that aren’t exactly related to each other. This will make the class complex and difficult to maintain. Programmers would rather have high cohesion instead of low to make sure the class has one, focused responsibility instead of multiple like this one.**
* **Source: https://stackoverflow.com/questions/3085285/difference-between-cohesion-and-coupling**

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 class into smaller classes. Where there is a break in code and comments that is where I would add a new class. For example, I would have a class for the profile of the user, and then a workout tracking class, nutrition class, and scheduling class.

**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

* Abstract class is the best choice because it allows for common traits and methods of students to be shared. The ‘Student’ list can also have objects from the type of student. With interface I would not be able to store common student data and there would be a lot of repetition. Using a regular class isn’t the greatest neither because we wouldn’t be able to use subclasses for the year-specific state and behaviors. Abstract class is the right approach because it provides the right balance of shared functionality.

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 above method is flawed because it is attempting to remove elements, modifying the list, while iterating through it. This will require an ‘Exception’ and overall it’ll most likely cause errors.

**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.)

* **No, it is not well-encapsulated. All the variables are declared as public which violates the principle of encapsulation. This can lead to misuse or inconsistency if left like this. In order to make it better, all the instance variables can be changed to private.**
* **Source: https://www.geeksforgeeks.org/encapsulation-in-java/**

**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.)

* The trim levels are subclasses of the car class. Changing the trim level requires creating a new instance of a different subclass. There is a reliance on inheritance. Adding or modifying trim levels changes the class hierarchy which makes the system less scalable and adaptable to new requirements.

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.)

* We can refactor using composition by having a TrimLevel interface that defines common behaviors and attributes. Each trim level should have separate classes, Base, Sports, and Luxury. The Car class would have a TrimLevel field instead of inheritance. This allows easy switching between trim levels without creating new objects.

**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.

* The program would become harder to maintain and update. When needing to fix a bug, it’ll need to be done in two places instead of one. Using inheritance would be able to solve this.

**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?

* It is unwise because adding a database functionality to them would make it harder to maintain and test. This approach reduces flexibility and reusability and complicates testing.

**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.

* The conditions that need to meet is that the classes should share common characteristics to form a superclass. Then the subclasses should be more specific for meaningful code reuse and reducing duplication. The classes should also form a form a group within eachother meaning it should fit well together and make sense of what the program needs to do.

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

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

* Inheritance is a parent-child relationship between classes. Child classes inherit traits from parents. Composition combines objects as parts. Inheritance is good for creating specialized versions of classes or sharing common code. Composition is more flexible allowing easier and more changes and looser connections between classes.

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

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

* Cohesion means how closely related the functionalities are within a module or component. High cohesion is better than low cohesion. High cohesion elements work together for a specific purpose. It has better readability, easier testing and reusable. Low cohesion means a module is doing way too many things and it makes the code harder to understand. Good programmers try to have high cohesion.