**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 many tasks, that all must be managed separately. Its functions are loosely related. The logworkout, logsteps, and logmeal functions all calculate one aspect of the users’ fitness, but do not work together.

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.)

To split the UserFitness class into more cohesive classes, I start with separating the responsibilities of the class. The class deals with the users personal information, their workout tracking, and their nutrition tracking. With these three distinctions, certain methods can be moved into separate classes or objects, and come together in the UserFitness 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

Student should be an abstract class, because there is no implementation of it at this level. The subclasses will then share the functionality of the Student class and all be able to have their own concrete methods.

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 is flawed due to modifying an element while iterating over it. The method above is attempting to remove the student while it is going through a list of third years. This changes the internal structure of the list that the method is looping through.

Cited: https://www2.ccs.neu.edu/research/demeter/related-work/riel/heuristics2.txt

**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.)

The BankAccount class is not well encapsulated because its variables (accountHolderName, accountNumber, balance, and interestRate) are all public. These variables should be hidden/private, and be accessed through the methods of the class to be well-encapsulated.

**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 current structure does not allow for a customer to change the trim level, only for the selected trim to be seen.

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.)

To allow trim-level changes, there could be a TrimLevel class, which handles the changing of an reference to the selected trim level. This would act like much like the engine class, which the Car superclass would reference.

**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 problem of code duplication would violate the DRY (don’t repeat yourself) principle. This could be solved by using abstraction, through inheritance or composition. This would reduce redundancy and allow the code to be maintained better, due to changes only having to be made in one place.

**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 would be unwise to place the database calls there because it would result in high coupling, which would make the code harder to test and maintain. This would also violate the single responsibility principle, because the any changes would need to occur in the domain classes.

**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.

To use inheritance, there must be a “is-a” relationship. The superclass would be something that is common amongst the identity of the different objects in the requirements. This would be something like there being different animals (superclass), and the subclasses being the specific animal.

**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, while composition is a “has-a” relationship. With inheritance, you can have superclasses, where subclasses can inherit/use the functions of it. This is useful when objects have a common identity. Composition is useful when objects have different behaviors that may change often. This allows more movement within the objects behaviors.

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

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

Cohesion is the level of relation between functions of a class. High cohesion is better because it is easier to maintain, due to classes following the single responsibility principle (not overlapping in function). Low cohesion often results in harder to upkeep and understand code due to the functions not working together.