LSP Midterm

1. A. The UserFitness class has low cohesion because the methods can’t really be reused for multiple things, and there are a lot of tasks that can’t really be used together. For example, logSteps, and logMeal can only be used to do those specific things, and nothing else.

(used this article to help explain <https://www.linkedin.com/advice/0/how-do-you-design-test-your-objects-low-coupling#:~:text=High%20cohesion%20means%20that%20the,or%20overlapping%20role%20and%20scope>.)

B. One thing I could do is create a logCalories method to access all of the calories burned and consumed, getting rid of the logMeal method and consolidating the logWorkout method. And I would get rid of the logSteps method, and combine it with the logWorkout method.

1. Student should be an abstract class because all three subclasses will most likely share the same behaviors. This way, the class can define all of the properties and methods at once.

The code throws a CurrentModificationException. The code removes elements from the list while iterating through it. (used chatGPT to help with this question)

1. No, accountHolderName, accountNumber, balance, interestRate, addInterest, deposit, and withdraw are all publicly accessible, and can be accessed and modified externally. We don’t want this since it is private information.To Change this, I would make BackAccount private. (used chatGPT to help)
2. A. With this UML, it doesn;t seem like the customer can change the trim level. They would have to choose the trim level, and then build the rest of the car around it. Choosing the trim level is an immutable property, and if the customer decides later in the process that they want to choose a different trim, they would have to choose a different engine, and potentially start all over again.

(used chatGPT, Notebook LM\*, and <https://creately.com/blog/diagrams/uml-diagram-types-examples/> for a and b)

B. Instead of having a fixed trim level, there should be a reference to a trim level object that can be referenced at any time.

1. This can cause code duplication as having similar code in multiple classes can lead to the code being duplicated. There's also poor reusability as the duplicated code is less likely to be reused effectively. To solve this, you can use functionality that doesn’t overlap partially. This way, the code becomes more flexible, and more usable. (Used chatGPT to help with this problem)
2. It may cause increased coupling because it tightly couples those classes to the specific database technology. and reduced cohesion as Car and BankAccount should focus on representing the behaviors of a car and bank account. There would also be a hindered maintainability as the database logic is scattered through domain classes. ( I used Notebook LM for this question)
3. To create a new class hierarchy using inheritance, you need to make sure there is an “is-a” relationship, so that the subclass represents a specialized version of the subclass. Also, when subclasses extend the functionality of the superclass, rather than overriding its core responsibilities. Also when there is no transmutation, utility class extension, and when there is a problem domain focus. (used Notebook LM)
4. Inheritance has an “is-a” relationship that allows a subclass to inherit properties and behaviors from its superclass. Composition models a “has-a” relationship where one class contains an instance of another class as a member. You choose inheritance when a clear “is-a” relationship exists, and the subclass genuinely represents a specialized kind of the superclass, the subclass extends the functionality of the superclass without overriding or nullifying its core responsibilities, dynamic binding and polymorphism are desired, allowing for flexible behavior based on the object’s type at runtime. Use composition when there is an “has-a” relationship and one class contains or uses another class as a part, loose coupling is desired, and greater flexibility is needed.

(used Notebook LM to help with this question)

1. Cohesion refers to the degree to which the elements within a module (such as a class or a method) are functionally related. High cohesion is typically better as it leads to systems that are easier to understand, maintain, and modify. More robust and manageable software systems can be created by adhering to principles that promote high cohesion.

(used Notebook LM to help with this answer)

\*Notebook LM source - I uploaded all of the lecture slides to Google’s Notebook LM, and had that extract information from the slides to help me answer the questions.