LSP Midterm Part1

1. User fitness has low-cohesion because it combines unrelated responsibilities throughout the code. In order to improve the code i would suggest having separate classes for different parts of the fitness tracker. For instance i would put things like user name in a different class than i would put user meals . Just so the code is more readable and easy to follow .
2. They should use an Abstract class because you can keep standard student information while keeping the subclasses you want, like the first-year, second-year, etc. The above code is flawed because they're removing things from the list while their iterating over it which can cause some issues. Arthur Riel: Arthur Riel’s Object-Oriented Design: *Do not modify the collection you are iterating over*.
3. Bank accounts need to be better encapsulated because instances like balance, account holder name and account number are all public. Without the protection then these fields can be modified and can lead to some code errors and inconsistencies. The first thing I would do is to make the fields private so that they’re not easily accessible by other code. If that data needs to be reached it can be reached by adding some getters and setters so there is more structure and continuity with each data reference.
4. The structure is rigid and does not allow for the flexibility of changing the “trim level” that users are supposed to be able to do. To fix this instead of using inheritance , composition should be used because it will allow to users to change the trim levels easier.
5. This may be a problem because it increases the risk of errors, due to things like code duplication. Which is why the “dont repeat yourself rule” is used . Riel solves this potential issue my introducing concepts such as inheritance, composition and interface.
6. Arthur Riel believes that “ classes should focus on one responsibility” if we try to make the Car and Bank Account classes do too much outside of their scope it could make the code too reliant on those classes and it would make potential issues or errors that could occur harder to fix and resolve.
7. **Is-a Relationship**: Subclasses must logically be specialized versions of the superclass.

* **Shared Functionality**: Common behavior or data should be abstracted into the superclass to avoid duplication.
* **Polymorphism**: The design should allow subclasses to override or extend superclass methods.
* **Stable Superclass**: The superclass should be stable and unlikely to change frequently.
* **Extensibility**: The hierarchy should allow easy addition of new subclasses without modifying existing ones.

1. Inhertiance references when a new class derives from an existing one creating that is-a relationships i mentioned in the previous question, while composition has an has-a relationship and focuses more on class containing instances of other classes . I would choose composition if i was going for flexibility and i would choose inheritance if there was a clear hierarchy or there was common functionality.
2. Cohesion is focused on how closely related classes or modules are, you want to aim for high cohesion because that means that your code is easy to understand, is more maintainable, and reusable.

Sources:   
  
<https://chatgpt.com/share/67119e8d-d4ac-8011-9707-a26c1d512ad2>

<https://thoughtbot.com/blog/reusable-oo-composition-vs-inheritance>

<https://stackoverflow.com/questions/14000762/what-does-low-in-coupling-and-high-in-cohesion-mean>

<https://chatgpt.com/>