Part A: Conceptual Questions

1. Definition

In your own words, define encapsulation. Include an example of how it can prevent unintended changes to data.

Encapsulation basically hides specific parts of a class. By making attributes private we can control how the data is modified.

2. Visibility Modifiers

Compare public, private, and protected access. For each, list one benefit and one potential drawback (in terms of code flexibility, safety, or maintainability).

Public: Easy access to members from anywhere, but it may lead to unintended changes.

Private: It protects data making sure access is controlled, but it is harder to modify.

Protected: Supports inheritance allowing subclasses to access parent class members, but derived classes may modify members in unintended ways.

Name a scenario in which you might intentionally use protected members instead of private members.

A scenario where you might use protected members could be when designing characters for a game. Each character can come from the same base class, but their attributes will be different.

3. Impact on Maintenance

Explain why encapsulation can reduce debugging complexity when maintaining a large codebase.

Encapsulation prevents accidental data corruption, makes it easier to isolate bugs, controls data access, and it makes large codes more maintainable.

Provide a brief example (no code needed, just a scenario) of how code could break if internal data is made public.

If a bank account class stores a customer's balance it could easily be changed without going through the proper transactions.

4. Real-World Analogy

Think of a real-life object or system. How would you describe its “public interface” vs. its “private implementation”? Why is it helpful to keep the private side hidden?

Sticking with the bank example, its public interface would be the person doing the transaction, while the private implementation is what changes the person’s balance.