Homework - Understanding Encapsulation
Due: 03-25-2025 (Tuesday)

Part A: Conceptual Questions

1. Definition

- Encapsulation is when a class keeps its internal data private and only allows access to it through specific methods. This helps avoid mistakes or unwanted changes to the data.

Example: Let's say you have a class with a balance variable. If it's public, anyone could accidentally set it to a negative number. But if it's private, you can control changes using a deposit() method that checks if the amount is valid.

2. Visibility Modifiers

Access Modifier	Benefit	Drawback
Public	Easy to access from outside the class	Can lead to messy code or bugs if misused
Private	Keeps data safe and under control	Can make things harder to modify directly if needed
Protected	Good for inheritance, lets child classes access members	Less secure than private, could still be misused

3. Impact on Maintenance

- Encapsulation makes it easier to find bugs because you know only a few methods can change certain variables. If something goes wrong with the balance in a bank app, you only have to check the deposit() and withdraw() methods instead of the whole codebase.

Example:
If a variable like health in a game character class is public, and some part of the code sets it to 9999 by mistake, you might not even notice until things break. But if health is private, you can control changes through a method that checks the range.

4. Real-World Analogy

- A microwave is a good example. The buttons and screen are the public interface—you use them to start or stop it. But what happens inside (the wiring, the heat control, the timer logic) is the private implementation. It's better that regular users can't mess with that stuff because it keeps the machine working safely and consistently.

Part B: Small-Class Design (Minimal Coding)

Class Skeleton

```
class BankAccount {
private:
   double balance;
   int accountNumber;

public:
   void deposit(double amount);
   void withdraw(double amount);
};
```

Encapsulation Justification
Private Members:

balance: This should be private because we don’t want people to accidentally mess it up by setting it to invalid values.

accountNumber: It’s a sensitive piece of info. Keeping it private prevents mistakes or security issues.

Public Methods:
deposit(double amount): We can make sure the amount is positive before updating the balance.

withdraw(double amount): This lets us check if there’s enough money in the account before taking any out.

Documentation Example
In the comments for the class, I’d include something like:

```
// NOTE: Do not modify 'balance' directly.
// Always use deposit() or withdraw() to safely change account funds.
```
This lets other developers know that balance is meant to be controlled through specific methods only.

Part C: Reflection & Short-Answer

Pros and Cons
Two benefits:
1. Helps prevent bugs or bad data from getting into your program.
2. Makes the code easier to maintain, especially when a project gets bigger.

One limitation:

1. You might have to write more methods just to access or update values, which adds extra code even for simple things.

Encapsulation vs. Abstraction

Encapsulation is about hiding the actual data and only letting it be changed in controlled ways. Abstraction is more about hiding unnecessary details and only showing the important stuff.

Both help reduce complexity, but in different ways. They're both considered "information hiding" because they keep the behind-the-scenes stuff private and only show what's needed to the outside world.

Testing Encapsulated Classes

- Even if the data is private, you can still test the class using its public methods. For example, you could deposit an amount and then withdraw to see if the balance updates correctly. As long as your methods work, you don't need to access private variables directly to test them.