Assignment chapter9

Student

|

-----------------------------

| |

Undergraduate Student Graduate Student

|

-------------------------

| |

Freshman Sophomore

### 2. 1. Private Access (Strong Encapsulation)

**Definition:**

* **Private access** means that members (variables or methods) of a class cannot be accessed or modified directly by any class outside of the defining class, not even by subclasses.

**Merits:**

* **Encapsulation and Data Hiding**:
  + **Private members** provide the highest level of encapsulation. The internal state of the class is completely hidden from outside access, ensuring that objects are self-contained.
  + **Protects invariants**: By restricting direct access, private members help ensure that the class's invariants (i.e., valid internal state) cannot be altered improperly by subclasses or external code.
* **Increased Control**:
  + By using private access, the superclass has complete control over its data. Any changes to private members can only occur through public methods (e.g., getters and setters). This makes it easier to add validation or complex logic for state changes.
* **Better Maintainability**:
  + With private members, the internal implementation of a class can change freely without worrying about breaking subclasses. External code that interacts with the class only knows about the public interface (methods), not the internal structure.
  + This makes future refactoring and changes to the class less likely to impact subclasses or other dependent code.

### 2.2 Protected Access (Limited Encapsulation)

**Definition:**

* **Protected access** allows subclasses (and classes in the same package, depending on language specifics) to directly access or modify the member variables or methods. It provides a middle ground between **private** and **public** access.

**Merits:**

* **Flexible Inheritance**:
  + Protected members allow subclasses to directly access and modify the superclass's data. This can be convenient when subclassing, as the subclass can easily reuse and extend the functionality of the superclass without needing to go through getters and setters.
* **Avoiding Boilerplate Code**:
  + By using protected access, a subclass can directly manipulate superclass fields or methods without having to write excessive getter/setter methods. This can make subclassing more intuitive and reduce the amount of code written.
* **Better Performance**:
  + Accessing protected members is generally faster than using getters and setters, as there's no need for extra method calls. This might be particularly important in performance-sensitive applications.