Imagine you are designing a Student class. In this class, the student's name should be directly accessible (public), but the id should be private and accessible only through a getter function.

(a) How would you declare the Student class with a public name member and a private id member?
(b) Provide a code snippet that demonstrates creating a Student object, directly modifying the name, and accessing the id via its getter function.
(c) Why might a designer choose to allow direct access to some members while restricting others?

(a) Declaring the `Student` Class

```
class Student {
public:
    std::string name;  // Public: Can be accessed directly

    Student(const std::string& studentName, int studentId) : name(studentName), id(studentId) {}

    int getId() const {  // Getter function for id
        return id;
    }

private:
    int id;  // Private: Can only be accessed via getId()
};
```

(b) Code Snippet Demonstrating Usage

```
#include <iostream>

class Student {
public:
    std::string name;

    Student(const std::string& studentName, int studentId) : name(studentName), id(studentId) {}

    int getId() const {
        return id;
    }

private:
    int id;
};

int main() {
    Student student("Alice", 12345);  // Create a Student object

    // Modify name directly
    student.name = "Bob";

    // Access id using getter function
    std::cout << "Student Name: " << student.name << std::endl;
    std::cout << "Student ID: " << student.getId() << std::endl;

    return 0;
}
```

(c) Why Restrict Some Members but Not Others?

Encapsulation & Data Integrity

- Allowing direct access to `name` is reasonable because modifying a name is common and generally safe.
- The `id`, however, is a critical identifier that should not be arbitrarily changed. By making it private, we prevent accidental modifications.

Control Over How Data is Accessed & Modified

- A getter ensures that the `id` is read-only. If needed, additional validation or formatting can be added inside the function.

Future Flexibility

- If `id` storage needs to change (e.g., from `int` to `std::string` or a more complex type), only the class internals need modification, without affecting code that accesses it.

Security & Data Protection

- If IDs are sensitive, restricting direct modification ensures they cannot be altered maliciously or by mistake.