Encapsulation

Get familiar with important aspects of object-oriented programming called data hiding and encapsulation.

Data hiding

Data hiding is an essential concept in object-oriented programming. In simple terms, it can be defined as masking a class's internal operations and only providing an interface through which other entities can interact with the class without being aware of what is happening within.

The goal is to implement classes in a way that prevents unauthorized access to or modification of the original contents of a class by its instances (or objects). The underlying algorithms of one class need not be known to another class. The two classes can still communicate, though.

Components of data hiding

Data hiding can be divided into two primary components:

- Encapsulation
- Abstraction

When used together, they allow us to make efficient classes for further use in our application.

Encapsulation

Encapsulation is a fundamental programming technique used to achieve data hiding in OOP. Encapsulation in OOP refers to binding data and the methods to manipulate that data together in a single unit—class.

Encapsulation is usually done to hide the state and representation of an object from the outside. A class can be thought of as a capsule with methods and attributes inside it.

When encapsulating classes, a good convention is to declare all variables of a class private. This will restrict direct access by the code outside that class.

At this point, a question can be raised. If the methods and variables are encapsulated in a class, how can they be used outside that class? The answer to this is simple. One has to implement public methods to let the outside world communicate with this class. These methods are called **getters** and **setters**. We can also implement other custom methods.

Implementing encapsulation in programming languages

In this section, we will show how to implement encapsulation using some of the most popular object-oriented programming languages, such as Java, C#, Python, C++, and JavaScript.

For the sake of explanation, we'll start off by creating a simple `Movie` class, which contains the following three data members (attributes):

- `title`

- `year`

- `genre`

Below is the implementation of the `Movie` class in different OOP languages.

```
1   class Movie {
2     // Data members
3     private:
4       string title;
5       int year;
6       string genre;
7
8     public:
9       // Default constructor
10      Movie() {
11        title = "";
12        year = -1;
13        genre = "";
14      }
15
16      // Parameterized constructor
17      Movie(string t, int y, string g) {
18        title = t;
19        year = y;
20        genre = g;
21      }
22    };
```

Data member initialization using the constructor of the "Movie" class

There must be able a way to interact with variables (`title`, `year`, and `genre`). They include all of the movie's information. The question is, how do we access or modify them?

We can create a `getTitle()` method, which will return the title to us. Similarly, the other two members can also have corresponding getters.

We may draw a conclusion by studying the emerging pattern. These functions should be part of the class itself. Let's try it out.

```cpp
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   class Movie {
6     // Data members
7     private:
8       string title;
9       int year;
10      string genre;
11
12    public:
13      // Default constructor
14      Movie() {
15        title = "";
16        year = -1;
17        genre = "";
18      }
19
20      // Parameterized constructor
21      Movie(string t, int y, string g) {
22        title = t;
23        year = y;
24        genre = g;
25      }
26
27      // getters setters
28      string getTitle() {
29        return title;
30      }
31
```

Implementation of encapsulation

The `Movie` class has an interface with public methods for communication.

The private members (variables or functions) cannot be accessed directly from the outside, but public read and write functions allow access to them. This, in essence, is data encapsulation.

Advantages of encapsulation

- Classes are simpler to modify and maintain.
- Which data member we wish to keep hidden or accessible can be specified.
- We choose which variables are read-only and write-only (increases flexibility).

In the next lesson, we will discuss the concept of abstraction in detail.

Mark as Completed