OBJECT ORIENTED PROGRAMMING (PCC-CS503)

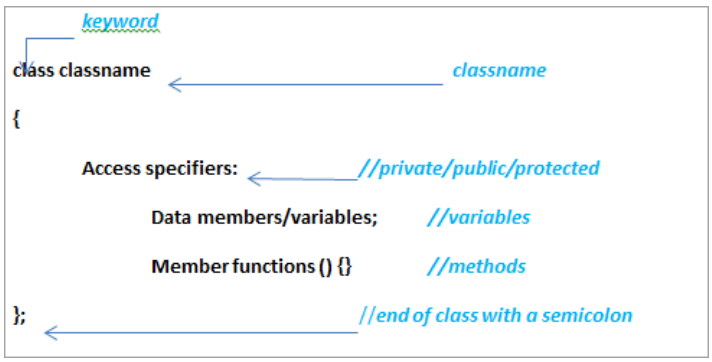
Unit – 3

The Fundamentals of Object Oriented Programming

Class and Object

- **Class** is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance (i.e. objects) of that class.
- The data and function defined within the class spring to life only when an object of type class is created.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.
- Example of class can be a *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.
- **Object** is an instance of a Class.
- The data and function defined within the class can only become usable when an object of type class is created.
- When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.
- A class is termed as a basic unit of encapsulation.

Structure and syntax of a class:



Syntax for creating the objects of a class:

```
ClassName ObjectName;
```

Accessing data members and member functions:

The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is `obj` and you want to access the member function with the name `printName()` then you will have to write `obj.printName()` and to access the member *a*, write `obj.a`

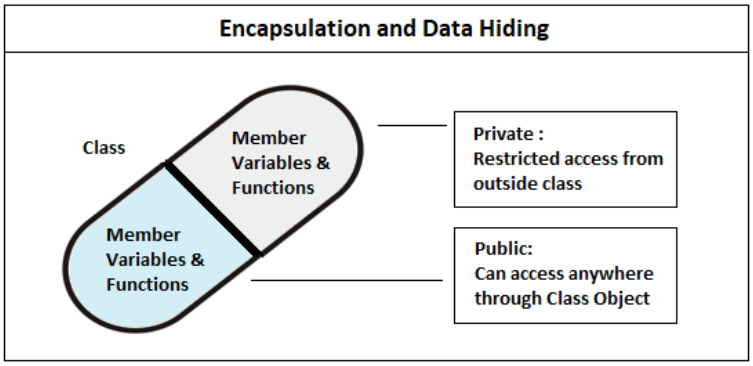
Example of a class:

```
#include <iostream>
using namespace std;

class person      // Defining a class
```

```
{
    //Access - Specifier
      public:

    //Variable Declaration
      string name;
      int number;

    //Function Definition
      void show()
      {
        cout << name << ": " << number << endl;
      }
};

int main()
{
    person obj;          // Object Creation For Class

    //Accessing data-members of the class using object
    cout << "Enter the Name :";
    cin >> obj.name;

    cout << "Enter the Number :";
    cin >> obj.number;

    //Accessing the member function of the class using object
    obj.show();

    return 0;
}
```

Encapsulation

- Encapsulation is the process of combining data and function into a single unit called class.
- Encapsulation is a powerful feature that leads to data hiding.
- They encapsulate all the essential properties of the object that are to be created.
- Using the method of encapsulation the programmer cannot access the class directly and thus it provides protection to the data of the class from the external access.



Data Hiding

- Data hiding is hiding the details of internal data members of a class from the external access by using the **access specifiers**.

- Access specifiers define how a member's variables and member's functions of a class can be accessed from outside the class.
- There are three access specifiers: **public, private and protected**
- Data hiding is also known as Information hiding.
- The data is hidden, so that it will be safe from accidental manipulation.
  - **Private members/methods** can only be accessed by methods defined as part of the class. Data is most often defined as private to prevent direct outside access from other classes. Private members can be accessed by members of the class.
  - **Public members/methods** can be accessed from anywhere in the program. Class methods are usually public which is used to manipulate the data present in the class. As a general rule, data should not be declared public. Public members can be accessed by members and objects of the class.
  - **Protected member/methods** are private within a class and are available for private access in the derived class.

Data Abstraction

- Abstraction is a mechanism to represent only essential features which are of significance and hides the unimportant details.

- To make a good abstraction we require a sound knowledge of the problem domain which we are going to implement using OOP principles.

- As an example of abstraction consider a class Vehicle. When we create the vehicle class, we can decide what function code and data to put in the class like vehicle name, no. of wheels, fuel type, vehicle type etc. and functions like changing the gear, accelerating/ decelerating the vehicle. At this time we are not interested how vehicle works like how acceleration, changing gear take place. We are also not interested in making more parts of vehicle to be part of the class like model number, vehicle color etc.

Procedural Abstraction

- A *procedural abstraction* is when we know *what* a procedure or a function will do but we don't know *how* it will do.

- This idea, that each conceptual unit of behavior should be wrapped up in a procedure, is called **procedural abstraction**.

- Procedural abstraction makes your code easier to read, understand, modify, and reuse.

- When can we go for **procedural abstraction**?

    - It's a big, ugly function and you want to hide the "how it works" details from code that might use it. Giving it a name allows the user to ignore how it's done.

    - It's a common thing to do, and you don't want to have to replicate the code in several places. Giving it a name allows multiple users to rely on the same (common) implementation.

    - It's conceptually a separate "task", and you want to be able to give it a name.

- **Example of procedural abstraction**: pow() function is used to calculate the power of a number without knowing the algorithm the function follows.

Difference between Abstraction and Encapsulation

Abstraction	Encapsulation
Abstraction solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction is about hiding unwanted details while showing most essential information.	Encapsulation means hiding the code and data into a single unit.
Abstraction allows focusing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Necessity for OOP

- Code reusability in terms of inheritance.
- Object oriented system can be easily upgraded from one platform to another.
- Complex projects can be easily divided into small code functions.
- The principle of abstraction and encapsulation enables a programmer to build secure program.
- Software complexity decreases.
- Principle of data hiding helps programmer to design and develop safe programs.
- Rapid development of software can be done in short span of time.
- More than one instance of same class can exist together without any interference.