**Object-Orientation (OO)**

• A technique for system modeling (system is set of interacting components)

• OO model consists of several interacting objects

**Class**

• A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" (plan) for creating objects

• A way to map real world objects into programming constructs

**Constructor**

• A constructor in C++ is a special method that is automatically called when an object of a class is created.

• To create a constructor, use the same name as the class, followed by parentheses ()

• The constructor has same name as of class

• The constructor has no return type

• It is always public

**Parameterized constructor**

• It is used to initialize various data elements of different objects with different values when they are created.

• It is used to overload constructors.

• Default constructors are called when constructors are not defined for the classes.

**Destructor**

• Destructor is a special member function that is called when the lifetime of an object ends • The purpose of the destructor is to free the resources that the object may have acquired during its lifetime • It has same name as the class • It is preceded by a tilde ( ~ ) • Destructor does not take arguments, it can never be overloaded

**Abstraction in OOP**

• Abstraction is the concept of object-oriented programming that “shows” only essential attributes and “hides” unnecessary information. The main purpose of abstraction is hiding the unnecessary details from the users. e.g., students’ abstraction, teachers’ abstraction in a university management system.

**Encapsulation**

• Encapsulation is a process of combining data members and functions in a single unit called class. This is to prevent the access to the data directly, the access to them is provided through the functions of the class. It is one of the popular feature of Object Oriented Programming(OOP) that helps in data hiding.

**Access Control and Inheritance**

• A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

• A derived class inherits all base class methods with the following exceptions − • Constructors and destructors of the base class

• Overloaded operators of the base class

• The friend functions of the base class

**Type of Inheritance**

• When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance. The type of inheritance is specified by the access specifier. • We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied –

• **Public Inheritance** − When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

• **Protected Inheritance** − When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

• **Private Inheritance** − When deriving from a private base class, public and protected members of the base class become private members of the derived class

**Constructor and Destructor calling sequence**

• In inheritance, the base class constructor is called before the child class constructor. • The destructors are called in the reverse order of constructors' calls.

**What is Polymorphism?**

• Polymorphism is a combination of two words. Poly means many and morphism means form.

• In OOP, polymorphism is the ability of objects of different types to respond to functions of the same name differently.

**Virtual function**

• A virtual function is a member function of the base class, that is **overridden** in derived class. The compiler binds virtual function at runtime, hence called **runtime polymorphism**. Use of virtual function allows the program to decide at runtime which function is to be called based on the type of the object pointed by the pointer.

• In C++, the member function of a class is selected at runtime using virtual function. The function in the base class is overridden by the function with the same name of the derived class.

• Once a function is declared virtual in a class then for all its derived classes that function will remain virtual.

**Static Binding Vs Dynamic Binding**

• **Static or Compile time or Early Binding:** By Default C++ Compiler do the early binding for all function calls i.e. while linking when compiler sees a function call, then it binds that call with the particular function’s address / definition.

**Operator overloading:**

**Syntax:** return-type **operator** op(){

Function statement;} Ex: int **operator** ++(){ n = n + 1; }

• **Dynamic or Run Time or Late Binding:** When we make a member function virtual then compiler performs run time binding for that function i.e. any call to that virtual function will not be linked to any function’s address during compile time. Actual function’s address to this call will be calculated at run time. To resolve the actual function’s address or definition at run time, C++ compiler adds some additional data structure around virtual functions i.e. • vTable • vPointers

**Abstract class – pure virtual function**

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function. You declare a pure virtual function by using a pure specifier (= 0) in the declaration of a virtual member function in class

**Limitations of Abstract class**

• You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class. You can, however, declare pointers and references to an abstract class

**Abstract class facts**

• A class is abstract if it has at least one pure virtual function. • We cannot instantiate an abstract class type; however, we can have pointers and references of abstract class type. • If we do not override the pure virtual function in derived class, then derived class also becomes abstract class. • An abstract class can have constructors.

**Concrete class**

• A concrete class is an ordinary class which has no purely virtual functions and hence can be instantiated.

• It has an implementation for all of its methods. There cannot have any unimplemented methods.

**Friend function**

Not in scope of class, it can’t be called for object of class. Have access to private members of class.

Can be invoked without any object.

Usually contains objects as arguments.

Can be declared inside public or private part of class.

It can’t access members directly by names and need object-name, member-name to access them.

**File handling:**

Data pattern stored in disk is called file.

1. **fstreambase**
2. **Ifstream** 🡪 derived from fstreambase
3. **Ofstream** 🡪 derived from fstreambase //write operation

**Two ways to open file**

1. Using constructor
2. Using member functiom open()of the class

**why we use constructors in c++**

Constructors in C++ are special member functions used to initialize the object's data members when an instance of a class is created. They ensure that the object starts with a well-defined state.

**This pointer:**

It invokes member function.

Used to point object that called the member function

& store reference address

**Static data member:**

Write outside the class and initialize with static in class, class variable. Only one and all objects share it

**Static function member**

When want function that only access static variables