What is Object Oriented Programming (OOP)?

OOPs, is a Object-oriented programming that implements the concept of **objects** in the program.

It aims to provide an easier solution to real-world problems by implementing real-world entities such as inheritance, abstraction, polymorphism, etc. in programming.

OOPs concept is widely used in many popular languages like Java, Python, C++, etc.

**Classes and Objects**: Classes are used to specify the structure of the data.

It is an user-defined data type which holds its own data members and member funtion, which can be accessed and used by creating an instance/object of that class.

You can create any number of objects from a class. class take up no memory and are called templates for objects.

**Objects** are the instances of classes.

---

**Access Modifiers:**

Access Modifiers in a class are used to assign access to the class members. It sets some restrictions on the class members not to get directly accessed by the outside functions.

It allows us to determine which class members are accessible to other classes and functions and which are not.

There are three types of access modifiers available in C++:

Public: All the class members with public modifier can be accessed from anywhere(inside and outside the class).

Private: All the class members with private modifier can only be accessed by the member function inside the class.

Protected: The access level of a protected modifier is within the class and outside the class through child class. If you do not make the child class, it cannot be accessed from outside the class.

Note: If we do not specify any access modifiers for the members inside the class, then by default, the access modifier for the members will be Private.

---

**Constructor**:

Constructor is automatically invoked when object is created. It is used to initialize the objects of its class. It is a special member function with the same name as of class.

Charachterstic of Contructor:

1. It should be declared in the public section of the class.

2. They are automaticallly invoked whenever object is created.

3. They can't return value or do have returb type.

4.It can have default arguments.

5. We can't refer to the address.

Types::>>

1.Default

2.Paramaterized

3.Copy constructor-

A copy constructor is a member function that initializes an object using another object of the same class.

The copy constructor is created when the new object is initialized with the existing object. The object is passed as an argument to the function.

It returns the object. Both the existing object and new object shares the different memory locations.'

Copy Constructor is of two types:

Default Copy constructor: The compiler defines the default copy constructor. If the user defines no copy constructor, compiler supplies its constructor.

User Defined constructor: The programmer defines the user-defined constructor.

Two types of copies are produced by the constructor:

Shallow copy: The default copy constructor can only produce the shallow copy.

A Shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.

Deep copy: Deep copy requires us to write the user-defined constructor.

Deep copy dynamically allocates the memory for the copy and then copies the actual value, both the source and copy have distinct memory locations.

In this way, both the source and copy are distinct and will not share the same memory location.

---

1.**Encapsulation**: Binding (or wrapping) data member(properties/fields) and function(method/behavior) together into a single entity is known as encapsulation.

In Encapsulation, we can hide the data's internal information, which is better for security concerns.

By encapsulation, we can make the class read-only.

Encapsulated code is better for unit testing.

to prevent the access to the data directly

It is achieved using the access specifiers, i.e., public, private and protected.

Real life example of Encapsulation: Consider a real life example of encapsulation, in a company there are different sections like the accounts section, finance section, sales section etc.

The finance section handles all the financial transactions and keep records of all the data related to finance. Similarly the sales section handles all the sales related activities and keep records of all the sales.

Now there may arise a situation when for some reason an official from finance section needs all the data about sales in a particular month.

In this case, he is not allowed to directly access the data of sales section.

He will first have to contact some other officer in the sales section and then request him to give the particular data.

This is what encapsulation is.

Here the data of sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

---

**2.Abstraction**: It is one of the most important features of OOP. It allows us to show only essential data or information to the user and hides the implementation details from the user.

Abstraction means displaying only essential information and hiding the details.

1)A real-world example of abstraction: Driving a car. When we drive a car, we do not need to know how the engine works (implementation) we only know how ECG works.

2)Real-life example: When you send an email to someone, you just click send, and you get the success message; what happens when you click send,

how data is transmitted over the network to the recipient is hidden from you (because it is irrelevant to you).

---

**3. Inheritance**: Inheritance is the process of inheriting the properties and behavior of an existing class into a new class.

When we inherit the class, we can reuse the existing class’s methods and fields into a new class. Inheritance can also be defined as the Is-A relationship, which is also known as the parent-child relationship.

Why we use inheritance:

● The main advantage of inheritance is code reusability. When we inherit the class, we can reuse the existing class’s methods and fields into a new class.

● The runtime polymorphism (method overriding) can be achieved by inheritance only.

Real life example of inheritance: For any bird, there are a set of predefined properties which are common for all the birds and there are a set of properties which are specific for a particular bird.

Therefore, we can say that all the birds inherit the common features like wings, legs, eyes, etc.

Therefore, in the object-oriented way of representing the birds, we first declare a bird class with a set of properties which are common to all the birds.

By doing this, we can avoid declaring these common properties in every bird which we create. Instead, we can simply inherit the bird class in all the birds which we create.

---

**4. Polymorphism**:

Polymorphism is the important features of Object-Oriented Programming that allows you to perform a single action in different ways.

polymorphism means many forms.

There are two types of polymorphism in C++:

1. Compile-time polymorphism: Compile time polymorphism is also known as static polymorphism. This type of polymorphism can be achieved through function overloading or operator overloading.
    - Function Overloading: When there are multiple functions in a class with the same name but different parameters, these functions are overloaded.

The main advantage of function overloading is it increases the readability of the program.

Functions can be overloaded by using different numbers of arguments and by using different types of arguments.

2. Runtime polymorphism: Runtime polymorphism is also known as dynamic polymorphism. Method overriding is a way to implement runtime polymorphism.
    - Method Overiding/Function Overiding: Basically function overriding means redefine a function which is present in the base class, also be defined in the derived class.

The best real-world example of polymorphism is a person that plays different roles at different palaces or situations.

At home a person can play the role of father, husband, and son.

At the office the same person plays the role of boss or employee.

In public transport, he plays the role of passenger.

In the hospital, he can play the role of doctor or patient.

So the same person exhibits different behavior in different situations. This is called polymorphism.

---

Interfaces:

The C++ interfaces are implemented using abstract classes,

In that class we use pure virtual function. A pure virtual function is represented as "= 0" in its declaration.

virtual void fun() = 0;

---

Friend Function:

A friend function is not a member of any class which is declared in the outside that class's scope, but it has the right to access all private and protected members of the class.

A friend function can be declared in the private or public section of the class.

Why we use? There is a situation like two classes having different member, Now I want access both classes data through one member function.

at this case, we use friend function.

```
class class_name {

   friend data_type function_name(argument); // syntax of friend function.  https://www.javatpoint.com/cpp-friend-function

};
```

---

Difference between Errors and Exceptions:

Exceptions can be handled during run-time whereas errors cannot be because exceptions occur due to some unexpected conditions during run-time

whereas about errors compiler is sure and tells about them during compile-time.

To overload main() function in C++, it is necessary to use class and declare the main as member function.

Note that main is not reserved word in programming languages like C, C++, Java and C#.

---

Virtual Function:

A virtual function is a member function which is declared within a base class and is redefined (overridden) by a derived class.

It is declared using the virtual keyword.

Virtual Funstion are mainly used to achieve Runtime polymorphism.

The classes which are containing virtual functions are not abstract classes.

All the derived classes may or may not redefine the virtual function.

When we use the same function name in both base and derived class, the function in base class is declared with a keyword virtual.

When you refer to a derived class object using a pointer to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

In Virtual Funtion, Base class pointer can point to the objects of base class as well as to the objects of derived class

https://www.javatpoint.com/cpp-virtual-function

Rules of a virtual function: o The virtual functions should be a member of some class.

o The virtual function cannot be a static member.

o Virtual functions are called by using the object pointer.

o It can be a friend of another class.

o C++ does not contain virtual constructors but can have a virtual destructo

---

Pure Virtual Function:

A pure virtual function is a member function which is declared within a base class and implemented in a derived class.

The classes which are containing pure virtual function are the abstract classes.

---

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

---

Abstraction vs encapsulation:>>

Abstraction is the method of hiding the unwanted information.

Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.

We can implement abstraction using abstract class and interfaces.

---

Early Binding(compile-time time polymorphism):

If the compiler knows at the compile-time which function is called, it is called early binding.

By default early binding happens in C++.

Late binding(Run time polymorphism):

If a compiler does not know at compile-time which functions to call up until the run-time, it is called late binding.

Late binding is achieved with the help of virtual keyword.

---

static Keyword: We use the static keyword to define the static data member or static member function inside and outside of the class that is used to access only static data members.

It is declared using the static keyword.

It helps to call functions that using class without using objects.

Constant Keyword: It is the const keywords used to define the constant value that cannot change during program execution.

( or constant is the keyword where we can use only for read-only operation )

It is declared using the const keyword.

It helps us to avoid modifying objects.

---

What is a class?

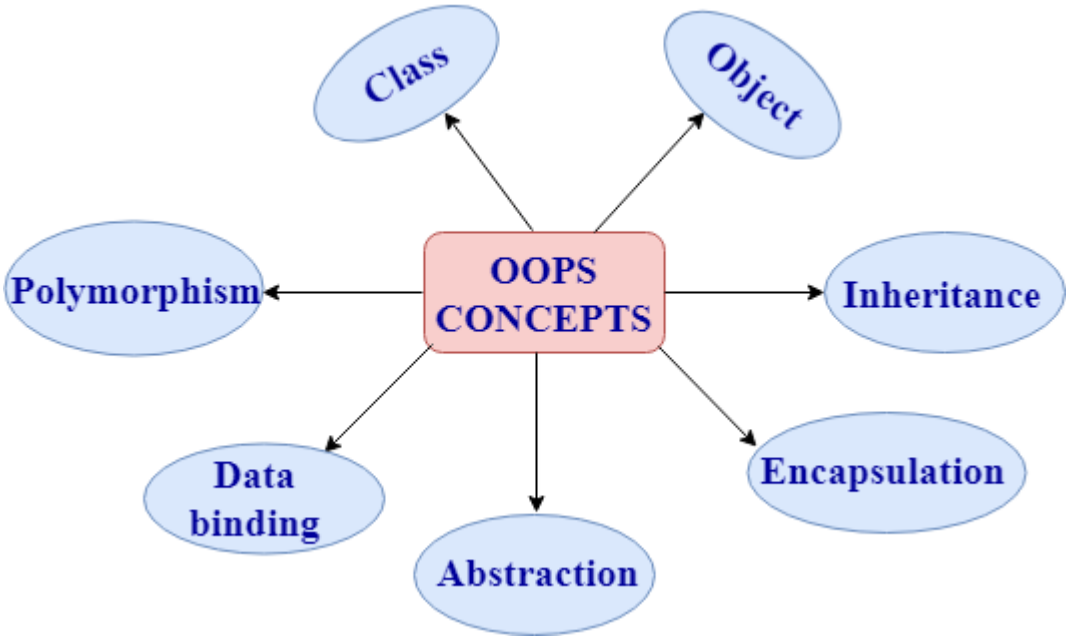
The class is a user-defined data type. The class is declared with the keyword class. The class contains the data members, and member functions whose access is defined by the three modifiers are private, public and protected. The class defines the type definition of the category of things. It defines a datatype, but it does not define the data it just specifies the structure of data.

You can create N number of objects from a class.

---

6) What are the various OOPs concepts in C++?

The various OOPS concepts in C++ are:




- **Class:**

The class is a user-defined data type which defines its properties and its functions. For example, Human being is a class. The body parts of a human being are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space. Therefore, we can say that the class is the only logical representation of the data.

**The syntax of declaring the class:**

```
1. class student
2. {
3. //data members;
4. //Member functions
5. }
```

Test it Now

- **Object:**

An object is a run-time entity. An object is the instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions. The class does not occupy any memory space. When an object is created using a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory. When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

```
1. class Student
2. {
3. //data members;
4. //Member functions
5. }
```

Test it Now

**The syntax for declaring the object:**

1. Student s = **new** Student();

Test it Now

- **Inheritance:**

Inheritance provides reusability. Reusability means that one can use the functionalities of the existing class. It eliminates the redundancy of code. Inheritance is a technique of deriving a new class from the old class. The old class is known as the base class, and the new class is known as derived class.

**Syntax**

1. **class** derived_class :: visibility-mode base_class;

Test it Now

Note: The visibility-mode can be public, private, protected.

- **Encapsulation:**

Encapsulation is a technique of wrapping the data members and member functions in a single unit. It binds the data within a class, and no outside method can access the data. If the data member is private, then the member function can only access the data.

- **Abstraction:**

Abstraction is a technique of showing only essential details without representing the implementation details. If the members are defined with a public keyword, then the members are accessible outside also. If the members are defined with a private keyword, then the members are not accessible by the outside methods.

- **Data binding:**

Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.

- **Polymorphism:**

Polymorphism means multiple forms. Polymorphism means having more than one function with the same name but with different functionalities. Polymorphism is of two types:

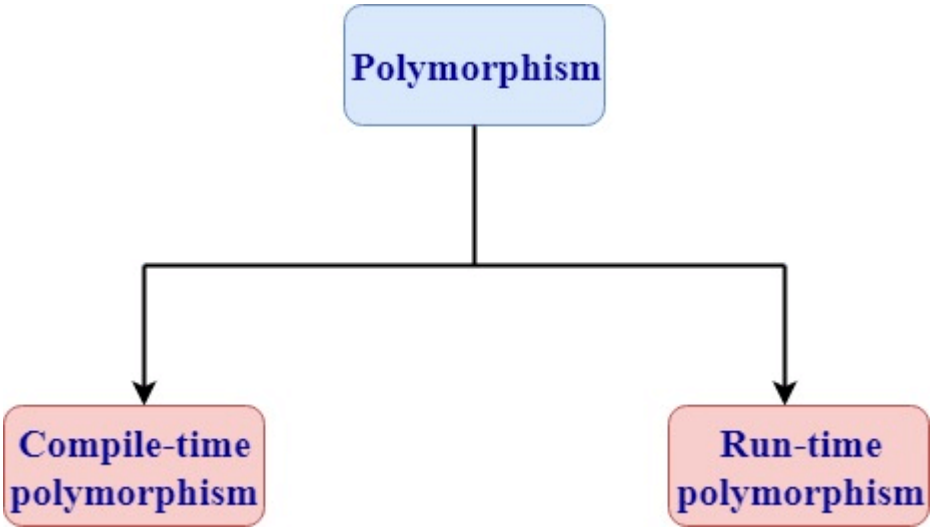
1. Static polymorphism is also known as early binding.
2. Dynamic polymorphism is also known as late binding.

---

7) What are the different types of polymorphism in C++?

Polymorphism: Polymorphism means multiple forms. It means having more than one function with the same function name but with different functionalities.

**Polymorphism is of two types:**




- **Runtime polymorphism**

Runtime polymorphism is also known as dynamic polymorphism. Function overriding is an example of runtime polymorphism. Function overriding means when the child class contains the method which is already present in the parent class. Hence, the child class overrides the method of the parent class. In case of function overriding, parent and child class both contains the same function with the different definition. The call to the function is determined at runtime is known as runtime polymorphism.

**Let's understand this through an example:**

```
#include <iostream>
using namespace std;
class Base
{
    public:
    virtual void show()
    {
        cout<<"javaTpoint";
     }
};
class Derived:public Base
{
    public:
    void show()
    {
        cout<<"javaTpoint tutorial";
    }
};

```

```
20. int main()
21. {
22.     Base* b;
23.     Derived d;
24.     b=&d;
25.     b->show();
26.             return 0;
27. }
```

Test it Now

**Output:**

*javaTpoint tutorial*

- **Compile time polymorphism**

Compile-time polymorphism is also known as static polymorphism. The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Method overloading is an example of compile-time polymorphism.

**Method overloading:** Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality.

Method overloading can be possible on the following basis:

- The return type of the overloaded function.
- The type of the parameters passed to the function.
- The number of parameters passed to the function.

**Let's understand this through an example:**

```
1.  #include <iostream>
2.  using namespace std;
3.  class Multiply
4.  {
5.     public:
6.     int mul(int a,int b)
7.     {
8.        return(a*b);
9.     }
10.    int mul(int a,int b,int c)
11.    {
12.       return(a*b*c);
13.   }
```

```
14.  };
15. int main()
16. {
17.     Multiply multi;
18.     int res1,res2;
19.     res1=multi.mul(2,3);
20.     res2=multi.mul(2,3,4);
21.     cout<<"\n";
22.     cout<<res1;
23.     cout<<"\n";
24.     cout<<res2;
25.     return 0;
26. }
```

Test it Now

**Output:**

*6*

*24*

- In the above example, mul() is an overloaded function with the different number of parameters.

---

8) Define namespace in C++.

- The namespace is a logical division of the code which is designed to stop the naming conflict.
- The namespace defines the scope where the identifiers such as variables, class, functions are declared.
- The main purpose of using namespace in C++ is to remove the ambiguity. Ambiquity occurs when the different task occurs with the same name.
- For example: if there are two functions exist with the same name such as add(). In order to prevent this ambiguity, the namespace is used. Functions are declared in different namespaces.
- C++ consists of a standard namespace, i.e., std which contains inbuilt classes and functions. So, by using the statement "using namespace std;" includes the namespace "std" in our program.
- **Syntax of namespace:**

```
1. namespace namespace_name
2. {
3.  //body of namespace;
4. }
```

Test it Now

Syntax of accessing the namespace variable:

```
1. namespace_name::member_name;
```

Test it Now

**Let's understand this through an example:**

```
1. #include <iostream>
2. using namespace std;
3. namespace addition
4. {
5.     int a=5;
6.     int b=5;
7.     int add()
8.     {
9.         return(a+b);
10.    }
11. }
12.
13. int main() {
14. int result;
15. result=addition::add();
16. cout<<result;
17. return 0;
18. }
```

Test it Now

**Output:**

*10*

---