**First see :https://www.w3schools.com/cpp/cpp\_oop.asp**

**Object Oriented Programming**

**Difference between structure and class**

|  |  |
| --- | --- |
| **Structure** | **Class** |
| Members of a structure are public by default | Members of a class are private by default |
| It does not support inheritance | Support inheritance |
| Structure are value type, They are stored as a stack on memory | class are reference type. They are stored as heap on memory. |
| It can have only parameterized constructor | It can have all the types of constructor and destructor |
| Generally used for smaller amounts of data. | Generally used for large amounts of data. |

**Class:**

A Class is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data membrs and member functions defines the properties and behavior of the objects in a Class. Class is a blueprint of data and functions or methods. There are two ways to define class member functions:

* Inside Class Defination
* OutSide Class Definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

RETURN-TYPE CLASS-NAME :: FUNCTION-NAME(PARAMETERS)

**Why is the size of an empty class not zero in C++?**

Size of an empty class is not zero. It is 1 byte generally. It is nonzero to ensure that the two different objects will have different addresses.

**Object:**

Objects are basic run-time entities in an object-oriented system, objects are instances of a class these are defined user defined data types.

**Encapsulation:**

Wrapping up (combing) of data and functions into a single unit is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapping in the class can access it. This insulation of the data from direct access by the program is called data hiding or information hiding.

**Data abstraction:**

Data abstraction refers to, providing only needed information to the outside world and hiding implementation details. Using abstract class we can hide the implementation details

# Characteristics of Abstruct class - A class is abstract if it has at least one pure virtual function. - 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.

**Inheritance:**

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. Inheritance provides re usability.

#### Why And When To Use "Inheritance"?

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

**Polymorphism:**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

polymorphism means ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation. C++ supports operator overloading and function overloading. The process of making an operator to exhibit different behaviors in different instances is known as operator overloading. Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.

In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. Real life example of polymorphism, a person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee. So a same person posses have different behavior in different situations. This is called polymorphism.

In C++ polymorphism is mainly divided into two types:

* Compile time polymorphism:
  + Function Overloading: When there are multiple functions with same name but different parameters then these functions are said to be overloaded.
  + Operator Overloading: C++ also provide option to overload operators. For example, we can make the operator (‘+’) for string class to concatenate two strings. We know that this is the addition operator whose task is to add to operands. So a single operator ‘+’ when placed between integer operands, adds them and when placed between string operands, concatenates them.
* Run time Polymorphism:
  + Function overriding: on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

**Dynamic Binding:**

In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

**Message Passing:**

Objects communicate with one another by sending and receiving information to each other. A messagefor an object is a re quest for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

**Access modifier:**

Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are three type of access modifiers:

* Public: All the class members declared under public will be available to everyone.
* Private: The class members declared as private can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.
* Protected: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any sub-class (derived class) of that class.

You learned from the [Access Specifiers](https://www.w3schools.com/cpp/cpp_access_specifiers.asp) chapter that there are three specifiers available in C++. Until now, we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class). The third specifier, protected, is similar to private, but it can also be accessed in the **inherited** class

**Friend class and function in C++**

Friend Class: A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

Friend Function: Like friend class, a friend function can be given special grant to access private and protected members.

A friend function can be:

* A method of another class
* A global function

Rules for declaring friend function:

1. Create forward reference for the class, which member variables are accessed.
2. Create the class which funtion is going to friend function of another class
3. Create the class which forward reference is being created first
4. Define the friend function with :: resulation operator outside the class. Not define inline function

Some important points about friend functions and classes:

1. Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
2. Friendship is not mutual. If a class A is friend of B, then B doesn’t become friend of A automatically.
3. Friendship is not inherited
4. The concept of friends is not there in Java.

**Basic OOPs Interview Questions:**

**1. What is the difference between OOP and SOP?**

|  |  |
| --- | --- |
| Object-Oriented Programming | Structural Programming |
| Object-Oriented Programming is a type of programming which is based on objects rather than just functions and procedures | Provides logical structure to a program where programs are divided functions |
| Bottom-up approach | Top-down approach |
| Provides data hiding | Does not provide data hiding |
| Can solve problems of any complexity | Can solve moderate problems |
| Code can be reused thereby reducing redundancy | Does not support code reusability |

**2. What is OOPs?**

OOPs (Object-Oriented Programming) is a type of programming which is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

**3. Why use OOPs?**

* OOPs allows clarity in programming thereby allowing simplicity in solving complex problems
* Code can be reused through inheritance thereby reducing redundancy
* Data and code are bound together by encapsulation
* OOPs allows data hiding, therefore, private data is kept confidential
* Problems can be divided into different parts making it simple to solve
* The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms

**4. What are the main features of OOPs?**

* Encapsulation
* Inheritance
* Abstraction
* Polymorphism

**Classes and Objects OOPs Interview Questions:**

**5. What is an object?**

object is an instance of a class

An object is a real-world entity which is the basic unit of OOPs for example chair, cat, dog, etc. Different objects have different states or attributes, and behaviors.

**6. What is a class?**

a class is a template for objects

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" for creating objects.

A class is a prototype that consists of objects in different states and with different behaviors. It has a number of methods that are common the objects present within that class.

**7. What is the difference between a class and a structure?**

**Class:**User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.

**Structure:**A structure is basically a user-defined collection of variables which are of different data types.

**8. Can you call the base class method without creating an instance?**

Yes, you can call the base class without instantiating it if:

* It is a static method
* The base class is inherited by some other subclass

**9. What is the difference between a class and an object?**

|  |  |
| --- | --- |
| Object | Class |
| A real-world entity which is an instance of a class | A class is basically a template or a blueprint within which objects can be created |
| An object acts like a variable of the class | Binds methods and data together into a single unit |
| An object is a physical entity | A class is a logical entity |
| Objects take memory space when they are created | A class does not take memory space when created |
| Objects can be declared as and when required | Classes are declared just once |

## Constructors:

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 the same name as the class, it is always public, and it does not have any return value.

**Interview Questions on Features of OOPs**

**Inheritance:**

**10. What is inheritance?**

Inheritance is a feature of OOPs which allows classes inherit common properties from other classes. For example, if there is a class such as ‘vehicle’, other classes like ‘car’, ‘bike’, etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

**11. What are the different types of inheritance?**

* Single inheritance
* Multiple inheritance
* Multilevel inheritance
* Hierarchical inheritance
* Hybrid inheritance

**12. What is the difference between multiple and multilevel inheritance?**

|  |  |
| --- | --- |
| Multiple Inheritance | Multilevel Inheritance |
| Multiple inheritance comes into picture when a class inherits more than one base class | Multilevel inheritance means a class inherits from another class which itself is a subclass of some other base class |
| Example: A class defining a child inherits from two base classes Mother and Father | Example: A class describing a sports car will inherit from a base class Car which inturn inherits another class Vehicle |

**13. What is hybrid inheritance?**

Hybrid inheritance is a combination of multiple and multi-level inheritance.

**14. What is hierarchical inheritance?**

Hierarchical inheritance refers to inheritance where one base class has more than one subclasses. For example, the vehicle class can have ‘car’, ‘bike’, etc as its subclasses.

**15. What are the limitations of inheritance?**

* Increases the time and effort required to execute a program as it requires jumping back and forth between different classes
* The parent class and the child class get tightly coupled
* Any modifications to the program would require changes both in the parent as well as the child class
* Needs careful implementation else would lead to incorrect results

**16. What is a superclass?**

A superclass or base class is a class that acts as a parent to some other class or classes. For example, the Vehicle class is a superclass of class Car.

[[](https://www.edureka.co/data-science-python-certification-course)](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

**[Python Certification Training for Data Science](https://www.edureka.co/data-science-python-certification-course" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

**17. What is a subclass?**

A class that inherits from another class is called the subclass. For example, the class Car is a subclass or a derived of Vehicle class.

**Polymorphism**

**18. What is polymorphism?**

Polymorphism refers to the ability to exist in multiple forms. Multiple definitions can be given to a single interface. For example, if you have a class named Vehicle, it can have a method named speed but you cannot define it because different vehicles have different speed. This method will be defined in the subclasses with different definitions for different vehicles.

**19. What is static polymorphism?**

Static polymorphism (static binding) is a kind of polymorphism that occurs at compile time. An example of compile-time polymorphism is method overloading.

**20. What is dynamic polymorphism?**

Runtime polymorphism or dynamic polymorphism (dynamic binding) is a type of polymorphism which is resolved during runtime. An example of runtime polymorphism is method overriding.

**21. What is method overloading?**

Method overloading is a feature of OOPs which makes it possible to give the same name to more than one methods within a class if the arguments passed differ.

**22. What is method overriding?**

Method overriding is a feature of OOPs by which the child class or the subclass can redefine methods present in the base class or parent class. Here, the method that is overridden has the same name as well as the signature meaning the arguments passed and the return type.

**23. What is operator overloading?**

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it.

**24. Differentiate between overloading and overriding.**

|  |  |
| --- | --- |
| Overloading | Overriding |
| Two or more methods having the same name but different parameters or signature | Child class redefining methods present in the base class with the same parameters/ signature |
| Resolved during compile-time | Resolved during runtime |

**Encapsulation**

**25. What is encapsulation?**

Encapsulation refers to binding the data and the code that works on that together in a single unit. For example, a class. Encapsulation also allows data-hiding as the data specified in one class is hidden from other classes.

Wrapping up data and function into a single unit. Must declare class variables/attributes as private.  If you want others to read or modify the value of a private member, you can provide public **get** and **set** method

Benefits:

* 1. Provides data hiding.
  2. Reusability
  3. Code can be modify without breaking the code.
  4. Reduces complexity.
* It is considered good practice to declare your class attributes as private (as often as you can). Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts
* Increased security of data

**26. What are ‘access specifiers’?**

Access specifiers or access modifiers are keywords that determine the accessibility of methods, classes, etc in OOPs. These access specifiers allow the implementation of encapsulation. The most common access specifiers are public, private and protected. However, there are a few more which are specific to the programming languages.

In C++, there are three access specifiers:

* public - members are accessible from outside the class
* private - members cannot be accessed (or viewed) from outside the class
* protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about [Inheritance](https://www.w3schools.com/cpp/cpp_inheritance.asp) later.

**27. What is the difference between public, private and protected access**

**modifiers?**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Accessibility from own class | Accessibility from derived class | Accessibility from world |
| Public | Yes | Yes | Yes |
| Protected | Yes | Yes | No |
| Private | Yes | No | No |

**Data abstraction**

Abstraction : https://www.javatpoint.com/abstract-class-in-java

**28. What is data abstraction?**

Data abstraction is a very important feature of OOPs that allows displaying only the important information and hiding the implementation details. For example, while riding a bike, you know that if you raise the accelerator, the speed will increase, but you don’t know how it actually happens. This is data abstraction as the implementation details are hidden from the rider.

**29. How to achieve data abstraction?**

Data abstraction can be achieved through:

* Abstract class
* Abstract method

**30. What is an abstract class?**

An abstract class is a class that consists of abstract methods. These methods are basically declared but not defined. If these methods are to be used in some subclass, they need to be exclusively defined in the subclass.

**31.** **Can you create an instance of an abstract class?**

No. Instances of an abstract class cannot be created because it does not have a complete implementation. However, instances of subclass inheriting the abstract class can be created.

**32. What is an interface?**

javatpoint.com/interface-in-java

It is a concept of OOPs that allows you to declare methods without defining them. Interfaces, unlike classes, are not blueprints because they do not contain detailed instructions or actions to be performed. Any class that implements an interface defines the methods of the interface.

**33. Differentiate between data abstraction and encapsulation.**

|  |  |
| --- | --- |
| Data abstraction | Encapsulation |
| Solves the problem at the design level | Solves the problem at the implementation level |
| Allows showing important aspects while hiding implementation details | Binds code and data together into a single unit and hides it from the world |

**Methods and Functions OOPs interview questions**

**34. What are virtual functions?**

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

**35. What are pure virtual functions?**

Pure virtual functions or abstract functions are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

**36. What is a constructor?**

A constructor is a special type of method that has the same name as the class and is used to initialize objects of that class.

**37. What is a destructor?**

A destructor is a method that is automatically invoked when an object is destroyed. The destructor also recovers the heap space that was allocated to the destroyed object, closes the files and database connec tions of the object, etc.

**38. Types of constructors**

Types of constructors differ from language to language. However, all the possible constructors are:

* Default constructor
* Parameterized constructor
* Copy constructor
* Static constructor
* Private constructor

**39. What is a copy constructor?**

A copy constructor creates objects by copying variables from another object of the same class. The main aim of a copy constructor is to create a new object from an existing one.

**40. What is the use of ‘finalize’?**

Finalize as an object method used to free up unmanaged resources and cleanup before Garbage Collection(GC). It performs memory management tasks.

**41. What is Garbage Collection(GC)?**

GC is an implementation of automatic memory management. The Garbage collector frees up space occupied by objects that are no longer in existence.

**42. Differentiate between a class and a method.**

|  |  |
| --- | --- |
| Class | Method |
| A class is basically a template that binds the code and data together into a single unit. Classes consist of methods, variables, etc | Callable set of instructions also called a procedure or function that are to be performed on the given data |

**43. Differentiate between an abstract class and an interface?**

|  |  |  |
| --- | --- | --- |
| Basis for comparison | Abstract Class | Interface |
| Methods | Can have abstract as well as other methods | Only abstract methods |
| Final Variables | May contain final and non-final variables | Variables declared are final by default |
| Accessibility of Data Members | Can be private, public, etc | Public by default |
| Implementation | Can provide the implementation of an interface | Cannot provide the implementation of an abstract class |

**44. What is a final variable?**

A variable whose value does not change. It always refers to the same object by the property of non-transversity.

**Exception Handling**

**45. What is an exception?**

An exception is a kind of notification that interrupts the normal execution of a program. Exceptions provide a pattern to the error and transfer the error to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

**46. What is exception handling?**

Exception handling in Object-Oriented Programming is a very important concept that is used to manage errors. An exception handler allows errors to be thrown and caught and implements a centralized mechanism to resolve them.

[[](https://www.edureka.co/data-science-python-certification-course)](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

**[Python Certification Training for Data Science](https://www.edureka.co/data-science-python-certification-course" \t "_blank)**

[Weekday / Weekend B](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

**47. What is the difference between an error and an exception?**

|  |  |
| --- | --- |
| Error | Exception |
| Errors are problems that should not be encountered by applications | Conditions that an application might try to catch |

**48. What is a try/ catch block?**

A try/ catch block is used to handle exceptions. The try block defines a set of statements that may lead to an error. The catch block basically catches the exception.

**49. What is a finally block?**

A finally block consists of code that is used to execute important code such as closing a connection, etc. This block executes when the try block exits. It also makes sure that finally block executes even in case some unexpected exception is encountered.

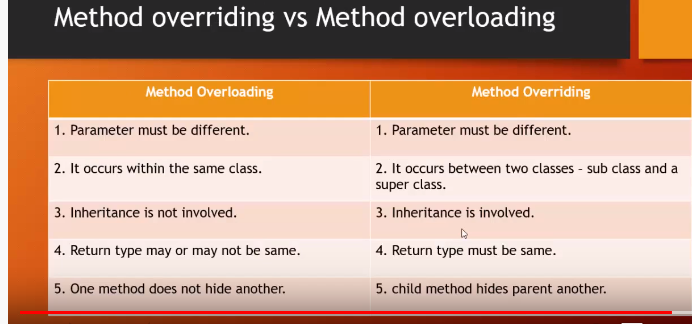
**Limitations of OOPs**

**50. What are the limitations of OOPs?**

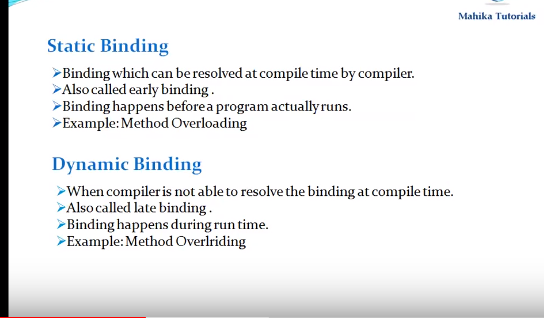
* Usually not suitable for small problems
* Requires intensive testing
* Takes more time to solve the problem
* Requires proper planning
* The programmer should think of solving a problem in terms of objects

Declaring a method in a subclass which is already present in superclass is known as method overridden.

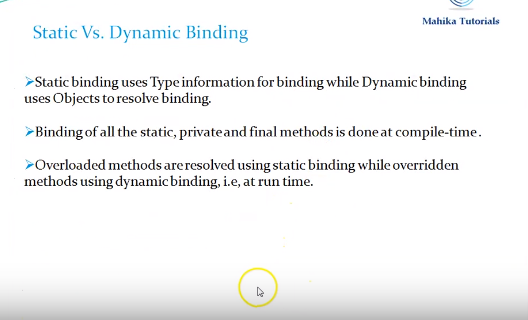
Same parameter.



Binding: Linking between method call and method definition.



Inheritance : A subclass can inherit of state and behavior of its super class is known as Inheritance .



A **thread**, in the context of **Java**, is the path followed when executing a program. All **Java** programs have at least one **thread**, known as the main **thread**, which is created by the **Java** Virtual Machine (JVM) at the program's start

### ****23. What is Interface in Java?****

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface. Read more on [Interface in Java](https://www.softwaretestingmaterial.com/interface-in-java/).