
---

# OOP Interview Questions Compilation

---

This note is prepared for aspiring candidates interviewing for positions such as Software Engineer, SQA, faculty roles in private or public universities, govt. jobs etc. It covers both theoretical and some coding questions commonly asked in these interviews.

**Prepared By**

**Salman Farsi**

**CUET, CSE'18**

**Contact:** salman.cuet.cse@gmail.com

**LinkedIn:** salmanfarsi0

**GitHub:** Salman1804102

# 1. What do you understand by OOP?

OOP stands for object-oriented programming. It is a programming paradigm that revolves around the object rather than function and procedure. In other words, it is an approach for developing applications that emphasize on objects. An object is a real word entity that contains data and code. It allows binding data and code together.

# 2. What is the purpose of using OOPs concepts?

The aim of OOP is to implement real-world entities like inheritance, hiding, polymorphism in programming. The main purpose of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

# 3. What are the advantages and disadvantages of OOP?

**Advantages of OOP**

- It follows a bottom-up approach.
- It models the real word well.
- It allows us the reusability of code.
- Avoids unnecessary data exposure to the user by using the abstraction.
- OOP forces the designers to have a long and extensive design phase that results in better design and fewer flaws.
- Decompose a complex problem into smaller chunks.
- Programmer are able to reach their goals faster.
- Minimizes the complexity.
- Easy redesign and extension of code that does not affect the other functionality.

**Disadvantages of OOP**

- Proper planning is required.
- Program design is tricky.
- Programmer should be well skilled.
- Classes tend to be overly generalized.

## 4. What are the limitations of OOPs?

- Requires intensive testing processes.
- Solving problems takes more time as compared to Procedure Oriented Programming.
- The size of the programs created using this approach may become larger than the programs written using the procedure-oriented programming approach.
- Software developed using this approach requires a substantial amount of pre-work and planning.
- OOP code is difficult to understand if you do not have the corresponding class documentation.
- In certain scenarios, these programs can consume a large amount of memory.
- Not suitable for small problems.
- Takes more time to solve problems.

## 5. What are the differences between object-oriented programming and structural programming?

| Object-oriented Programming | Structural Programming |
|---|---|
| **It follows a bottom-up approach.** | It follows a top-down approach. |
| **It provides data hiding.** | Data hiding is not allowed. |
| **It is used to solve complex problems.** | It is used to solve moderate problems. |
| **It allows reusability of code that reduces redundancy of code.** | Reusability of code is not allowed. |
| **It is based on objects rather than functions and procedures.** | It provides a logical structure to a program in which the program is divided into functions. |
| **It provides more security as it has a data hiding feature.** | It provides less security as it does not support the data hiding feature. |
| **More abstraction more flexibility.** | Less abstraction less flexibility. |
| **It focuses on data.** | It focuses on the process or logical structure. |

## 6. What do you understand by pure object-oriented language? Why Java is not a pure object-oriented programming language?

The programming language is called pure object-oriented language that treats everything inside the program as an object. The primitive types are not supported by the pure OOPs language. There are some other features that must satisfy by a pure object-oriented language:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction
- All predefined types are objects
- All user-defined types are objects
- All operations performed on objects must be only through methods exposed to the objects.

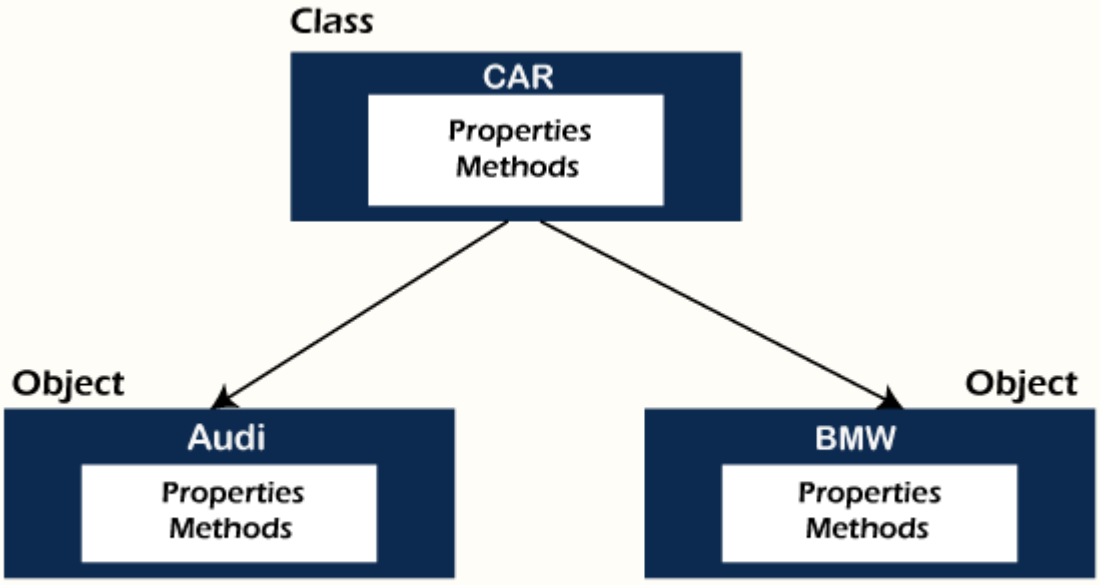
**Java is not a pure object-oriented programming language** because pre-defined data types in Java are not treated as objects. Hence, it is not an object-oriented language.

## 7. What do you understand by class and object? Also, give example.

**Class:** A class is a blueprint or template of an object. It is a user-defined data type. Inside a class, we define variables, constants, member functions, and other functionality. It does not consume memory at run time. Note that classes are not considered as a data structure. It is a logical entity. It is the best example of data binding.

**Object:** An object is a real-world entity that has attributes, behavior, and properties. It is referred to as an instance of the class. It contains member functions, variables that we have defined in the class. It occupies space in the memory. Different objects have different states or attributes, and behaviors.

The following figure best illustrates the class and object.




## 8. What are the differences between class and object?

| Class | Object |
|---|---|
| **It is a logical entity.** | It is a real-world entity. |
| **It is conceptual.** | It is real. |
| **It binds data and methods together into a single unit.** | It is just like a variable of a class. |
| **It does not occupy space in the memory.** | It occupies space in the memory. |
| **It is a data type that represents the blueprint of an object.** | It is an instance of the class. |
| **It is declared once.** | Multiple objects can be declared as and when required. |
| **It uses the keyword class when declared.** | It uses the new keyword to create an object. |
| **A class can exist without any object.** | Objects cannot exist without a class. |

## 9. What are the key differences between class and structure?

| Class | Structure |
|---|---|
| **Class is a group of common objects that shares common properties.** | The structure is a collection of different data types. |
| **It deals with data members and member functions.** | It deals with data members only. |
| **It supports inheritance.** | It does not support inheritance. |
| **Member variables cannot be initialized directly.** | Member variables can be initialized directly. |
| **It is of type reference.** | It is of a type value. |
| **It's members are private by default.** | It's members are public by default. |
| **The keyword class defines a class.** | The keyword struct defines a structure. |
| **An instance of a class is an object.** | An instance of a structure is a structure variable. |
| **Useful while dealing with the complex data structure.** | Useful while dealing with the small data structure. |

## 10. What is the concept of access specifiers when should we use these?

In OOPs language, **access specifiers** are reserved keyword that is used to set the accessibility of the classes, methods and other members of the class. It is also known as **access modifiers**. It includes **public, private,** and **protected**. There is some other access specifier that is language-specific. Such as Java has another access specifier **default**. These access specifiers play a vital role in achieving one of the major functions of OOP, i.e. encapsulation. The following table depicts the accessibility.

| Specifiers | Within Same Class | In Derived Class | Outside the Class |
|---|---|---|---|
| Private | Yes | No | No |
| Protected | Yes | Yes | No |
| Public | Yes | Yes | Yes |

## 11. What are the manipulators in OOP and how it works?

Manipulators are helping functions. It is used to manipulate or modify the input or output stream. The modification is possible by using the **insertion** (<<) and **extraction** (>>) operators. Note that the modification of input or output stream does not mean to change the values of variables. There are two types of manipulators with **arguments** or **without arguments.**

The example of manipulators that do not have arguments is **endl, ws, flush,** etc. Manipulators with arguments are **setw(val), setfill(c), setbase(val), setiosflags(flag).** Some other manipulators are **showpos, fixed, scientific, hex, dec, oct,** etc.

## 12. What are the rules for creating a constructor?

- It cannot have a return type.
- It must have the same name as the Class name.
- It cannot be marked as static.
- It cannot be marked as abstract.
- It cannot be overridden.
- It cannot be final.

## 13. What are the differences between the constructor and the method in Java?

| Constructor | Method |
|---|---|
| **Constructor has the same name as the class name.** | The method name and class name are not the same. |
| **It is a special type of method that is used to initialize an object of its class.** | It is a set of instructions that can be invoked at any point in a program. |
| **It creates an instance of a class.** | It is used to execute Java code. |
| **It is invoked implicitly when we create an object of the class.** | It gets executed when we explicitly called it. |
| **It cannot be inherited by the subclass.** | It can be inherited by the subclass. |
| **It does not have any return type.** | It must have a return type. |
| **It cannot be overridden in Java.** | It can be overridden in Java. |
| **It cannot be declared as static.** | It can be declared as static. |
| **Java compiler automatically provides a default constructor.** | Java compiler does not provide any method by default. |

## 14. How does procedural programming be different from OOP differ?

| Procedural Oriented Programming | Object-Oriented Programming |
|---|---|
| **It is based on functions.** | It is based on real-world objects. |
| **It follows a top-down approach.** | It follows a bottom-up approach. |
| **It is less secure because there is no proper way to hide data.** | It provides more security. |
| **Data is visible to the whole program.** | It encapsulates the data. |
| **Reuse of code is not allowed.** | The code can be reused. |
| **Modification and extension of code are not easy.** | We can easily modify and extend code. |

| | |
|---|---|
| **Examples of POP are C, VB, FORTRAN, Pascal, etc.** | Examples of OOPs are C++, Java, C#, .NET, etc. |

## 15. What are the differences between error and exception?

| **Basis of Comparison** | **Exception** | **Error** |
|---|---|---|
| **Recoverable/ Irrecoverable** | Exception can be recovered by using the try-catch block. | An error cannot be recovered. |
| **Type** | It can be classified into two categories i.e. checked and unchecked. | All errors in Java are unchecked. |
| **Occurrence** | It occurs at compile time or run time. | It occurs at run time. |
| **Package** | It belongs to java.lang.Exception package. | It belongs to java.lang.Error package. |
| **Known or unknown** | Only checked exceptions are known to the compiler. | Errors will not be known to the compiler. |
| **Causes** | It is mainly caused by the application itself. | It is mostly caused by the environment in which the application is running. |
| **Example** | **Checked Exceptions:** SQLException, IOException **Unchecked Exceptions:** ArrayIndexOutOfBoundException, NullPointerException, ArithmaticException | Java.lang.StackOverFlow, java.lang.OutOfMemoryError |

## 16. What are the characteristics of an abstract class?

An abstract class is a class that is declared as abstract. It cannot be instantiated and is always used as a base class. The characteristics of an abstract class are as follows:

- Instantiation of an abstract class is not allowed. It must be inherited.
- An abstract class can have both abstract and non-abstract methods.

- An abstract class must have at least one abstract method.
- You must declare at least one abstract method in the abstract class.
- It is always public.
- It is declared using the **abstract**

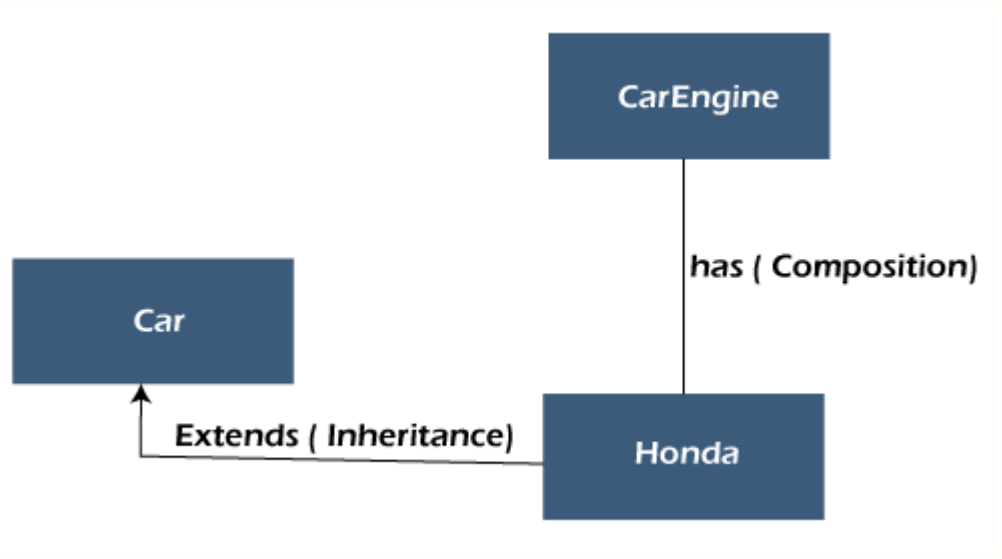
The purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share.

## 17. Is it possible for a class to inherit the constructor of its base class?

No, a class cannot inherit the constructor of its base class.

## 18. What is composition?

Composition is one of the vital concepts in OOP. It describes a class that references one or more objects of other classes in instance variables. It allows us to model a has-a association between objects. We can find such relationships in the real world. For example, a car has an engine. the following figure depicts the same




The main benefits of composition are:

- Reuse existing code
- Design clean APIs
- Change the implementation of a class used in a composition without adapting any external clients.

## 19. What are the differences between copy constructor and assignment operator?

The copy constructor and the assignment operator (=) both are used to initialize one object using another object. The main difference between the two is that the copy constructor allocates separate memory to both objects i.e. existing object and newly created object while the assignment operator does not allocate new memory for the newly created object. It uses the reference variable that points to the previous memory block (where an old object is located).

**Syntax of Copy Constructor**

```
class_name (const class_name &obj)
{
//body
}
```

**Syntax of Assignment Operator**

```
class_name obj1, obj2;
obj1=obj2;
```

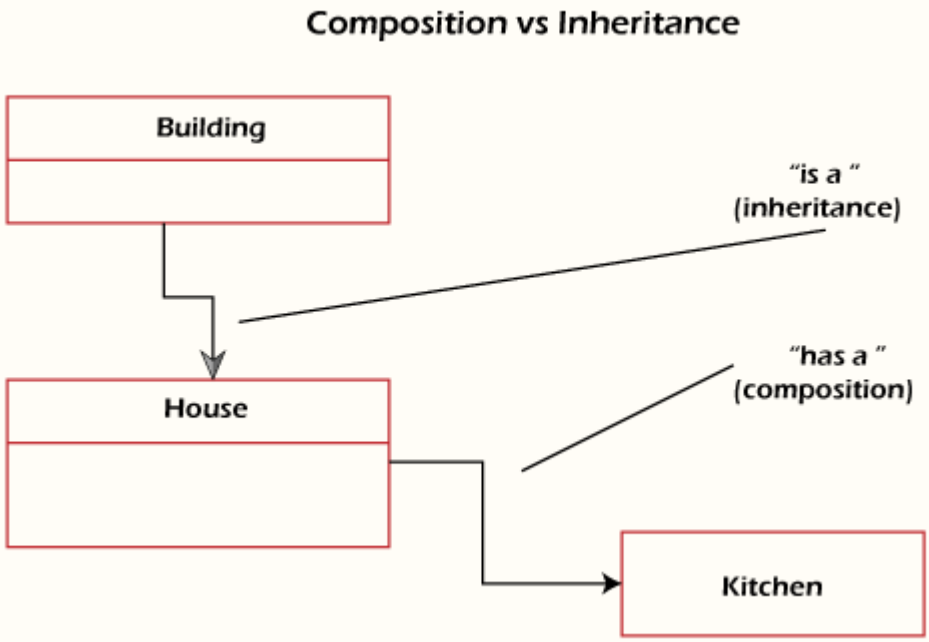
| Copy Constructor | Assignment Operator |
|---|---|
| **It is an overloaded constructor.** | It is an operator. |
| **It creates a new object as a copy of an existing object.** | It assigns the value of one object to another object both of which already exist. |
| **The copy constructor is used when a new object is created with some existing object.** | It is used when we want to assign an existing object to a new object. |
| **Both the objects use separate memory locations.** | Both objects share the same memory but use the two different reference variables that point to the same location. |
| **If no copy constructor is defined in the class, the compiler provides one.** | If the assignment operator is not overloaded then the bitwise copy will be made. |

## 20. Define Destructor?

A destructor is a method which is automatically called when the object is made of scope or destroyed. Destructor name is also same as class name but with the tilde symbol before the name.

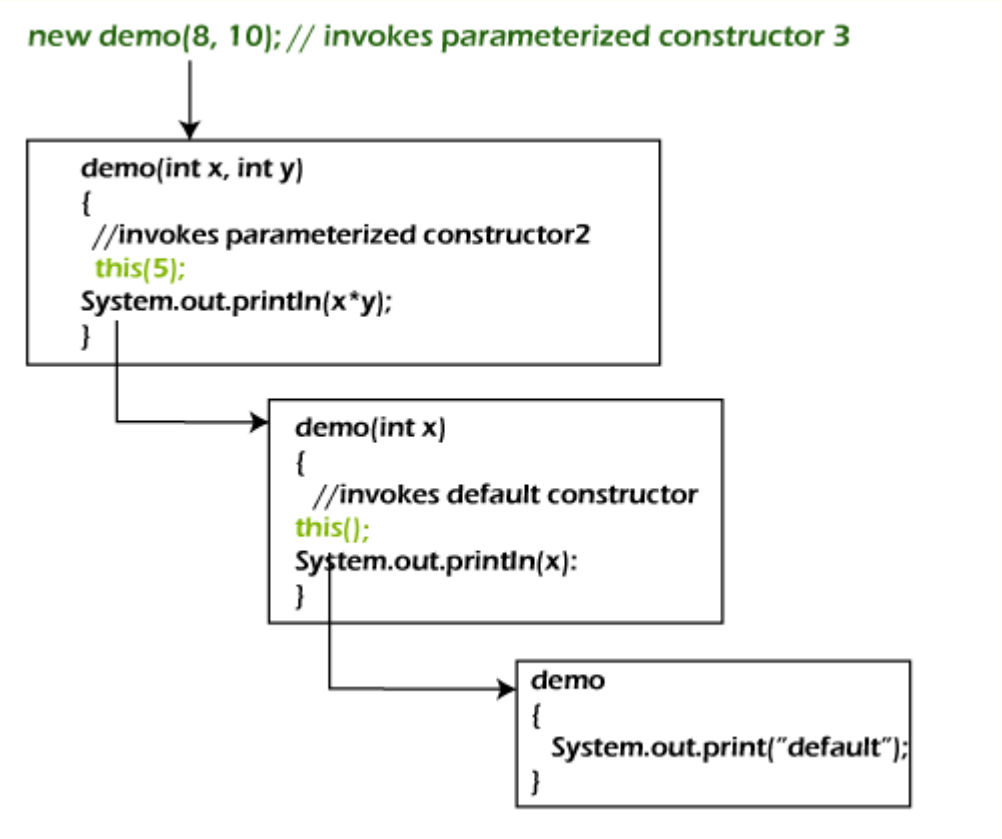
## 21. What is the difference between Composition and Inheritance?

Inheritance means an object inheriting reusable properties of the base class. Compositions mean that an object holds other objects. In Inheritance, there is only one object in memory (derived object) whereas, in Composition, the parent object holds references of all composed objects. From a design perspective, inheritance is "is a" relationship among objects whereas Composition is "has a" relationship among objects.




## 22. What is constructor chaining?

In OOPs, constructor chaining is a sequence of invoking constructors (of the same class) upon initializing an object. It is used when we want to invoke a number of constructors, one after another by using only an instance. In other words, if a class has more than one constructor (overloaded) and one of them tries to invoke another constructor, this process is known as constructor chaining. In C++, it is known as constructor delegation and it is present from C++ 11.




## 23. What are the limitations of inheritance?

- The main disadvantage of using inheritance is two classes get tightly coupled. That means one cannot be used independently of the other. If a method or aggregate is deleted in the Super Class, we have to refactor using that method in SubClass.
- Inherited functions work slower compared to normal functions.
- Need careful implementation otherwise leads to improper solutions.

## 24. What are the differences between Inheritance and Polymorphism?

| Inheritance | Polymorphism |
|---|---|
| **Inheritance is one in which a derived class inherits the already existing class's features.** | Polymorphism is one that you can define in different forms. |
| **It refers to using the structure and behavior of a superclass in a subclass.** | It refers to changing the behavior of a superclass in the subclass. |

| | |
|---|---|
| **It is required in order to achieve polymorphism.** | In order to achieve polymorphism, inherence is not required. |
| **It is applied to classes.** | It is applied to functions and methods. |
| **It can be single, hybrid, multiple, hierarchical, multipath, and multilevel inheritance.** | There are two types of polymorphism compile time and run time. |
| **It supports code reusability and reduces lines of code.** | It allows the object to decide which form of the function to be invoked at run-time (overriding) and compile-time (overloading). |

## 25. What is Encapsulation?

Encapsulation is an attribute of an object, and it contains all data which is hidden. That hidden data can be restricted to the members of that class. Levels are Public, Protected, Private, Internal, and Protected Internal.

Encapsulation is basically the idea of bundling data (attributes) and methods (functions) that work on the data into a single unit, which is called an object. This allows the data to be hidden from the outside world and accessed only through the methods of the object, which is known as data hiding.

## 26. What is Coupling in OOP and why it is helpful?

In programming, separation of concerns is known as **coupling**. It means that an object cannot directly change or modify the state or behavior of other objects. It defines how closely two objects are connected together. There are two types of coupling, **loose** coupling, and **tight** coupling.

Objects that are independent of one another and do not directly modify the state of other objects is called loosely coupled. Loose coupling makes the code more flexible, changeable, and easier to work with.

Objects that depend on other objects and can modify the states of other objects are called tightly coupled. It creates conditions where modifying the code of one object also requires changing the code of other objects. The reuse of code is difficult in tight coupling because we cannot separate the code.

Since using loose coupling is always a good habit.

## 27. Name the operators that cannot be overload.

- Scope Resolution Operator (::)
- Ternary Operator (? :)
- Member Access or Dot Operator (.)
- Pointer to Member Operator (.*)
- sizeof operator

## 28. What is the difference between new and override?

The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

**virtual:** indicates that a method may be overridden by an inheritor

**override:** Overrides the functionality of a virtual method in a base class, providing different functionality.

**new:** Hides the original method (which doesn't have to be virtual), providing different functionality. This should only be used where it is absolutely necessary.

When you hide a method, you can still access the original method by upcasting to the base class. This is useful in some scenarios, but dangerous.

## 29. What is the difference between new and override?

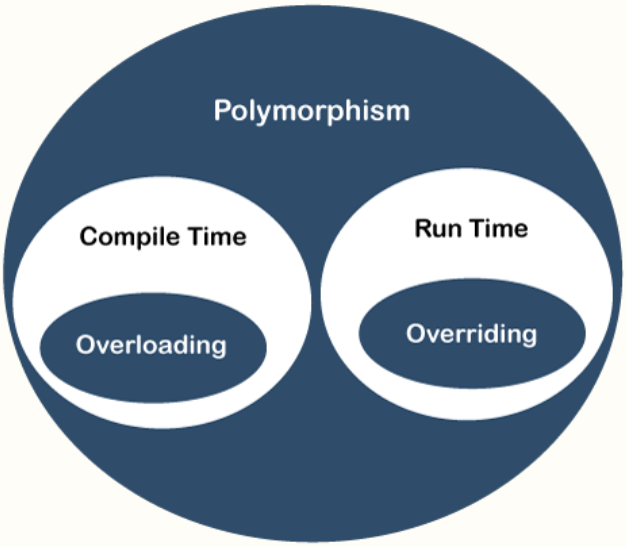
The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

## 30. Explain overloading and overriding with example?

**Overloading**

**Overloading** is a concept in OOP when two or more methods in a class with the same name but the method signature is different. It is also known as **compile-time polymorphism**. For example, in the following code snippet, the method **add()** is an overloaded method.

```
public class Sum
{
int a, b, c;
public int add();
{
c=a+b;
return c;
}
add(int a, int b);
{
//logic
}
add(int a, int b, int c);
{
//logic
}
add(double a, double b, double c);
{
//logic
}
//statements
}
```




### Overriding

If a method with the same method signature is presented in both child and parent class is known as method **overriding**. The methods must have the same number of parameters and the same type of parameter. It overrides the value of the parent class method. It is also known as **runtime polymorphism**. For example, consider the following program.

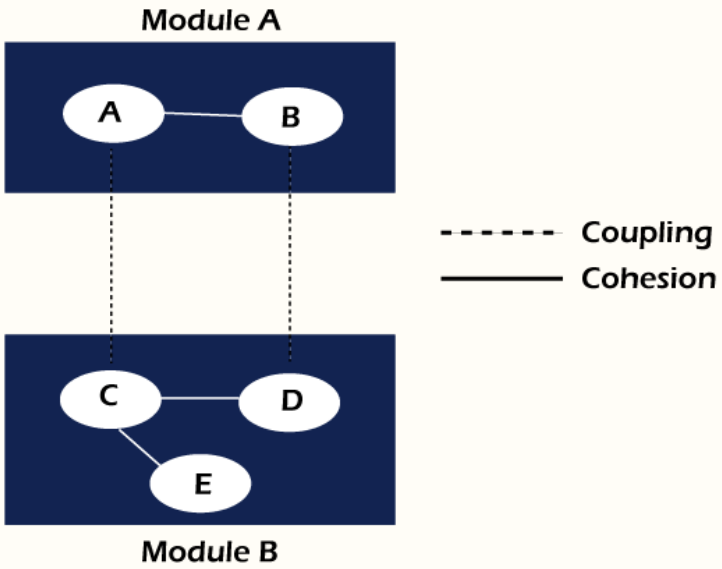
```
class Dog
{
public void bark()
```

```
{
System.out.println("woof ");
}
}
class Hound extends Dog
{
public void sniff()
{
System.out.println("sniff ");
}
//overrides the method bark() of the Dog class
public void bark()
{
System.out.println("bowl");
}
}
public class OverridingExample
{
public static void main(String args[])
{
Dog dog = new Hound();
//invokes the bark() method of the Hound class
dog.bark();
}
}
```

## 31. What is Cohesion in OOP?

In OOP, **cohesion** refers to the degree to which the elements inside a module belong together. It measures the strength of the relationship between the module and data. In short, cohesion represents the clarity of the responsibilities of a module. It is often contrasted with coupling.

It focuses on a how single module or class is intended. Higher the cohesiveness of the module or class, better is the object-oriented design.




There are two types of cohesion, i.e. **High** and **Low**.

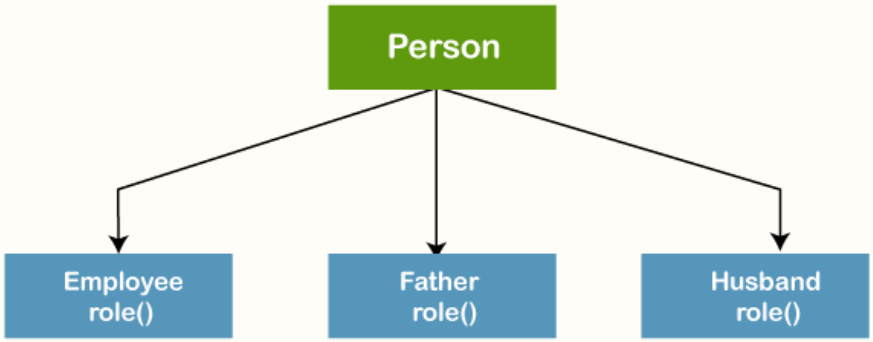
- High cohesion is associated with several required qualities of software including **robustness, reliability**, and **understandability**.
- Low cohesion is associated with unwanted qualities such as being difficult to **maintain, test, reuse,** or even **understand**.

High cohesion often associates with loose coupling and vice versa.

## 32. Give a real-world example of polymorphism?

The general meaning of Polymorphism is one that has different forms. 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.
- At the shop, he plays the role of customer.




Hence, the same person possesses different behavior in different situations. It is called polymorphism.

## 33. What is the difference between a base class and a superclass?

The base class is the root class- the most generalized class. At the same time, the superclass is the immediate parent class from which the other class inherits.

## 34. What is data abstraction and how can we achieve data 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. A real-world example of abstraction is 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.

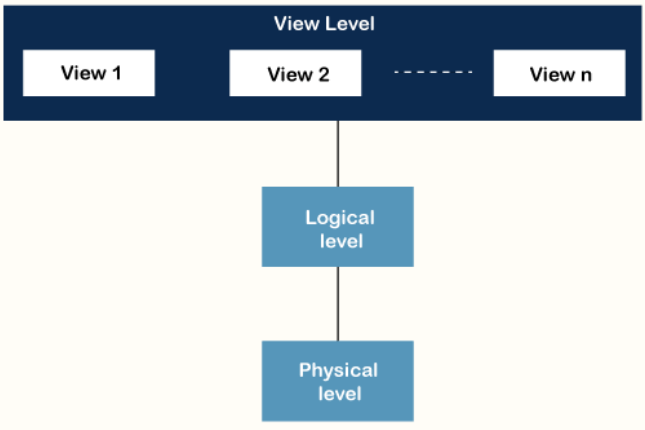
There are two ways to achieve data abstraction

- Abstract class
- Abstract method

## 35. What are the levels of data abstraction?

There are **three** levels of data abstraction:

- **Physical Level:** It is the lowest level of data abstraction. It shows how the data is actually stored in memory.
- **Logical Level:** It includes the information that is actually stored in the database in the form of tables. It also stores the relationship among the data entities in relatively simple structures. At this level, the information available to the user at the view level is unknown.
- **View Level:** It is the highest level of data abstraction. The actual database is visible to the user. It exists to ease the availability of the database by an individual user.

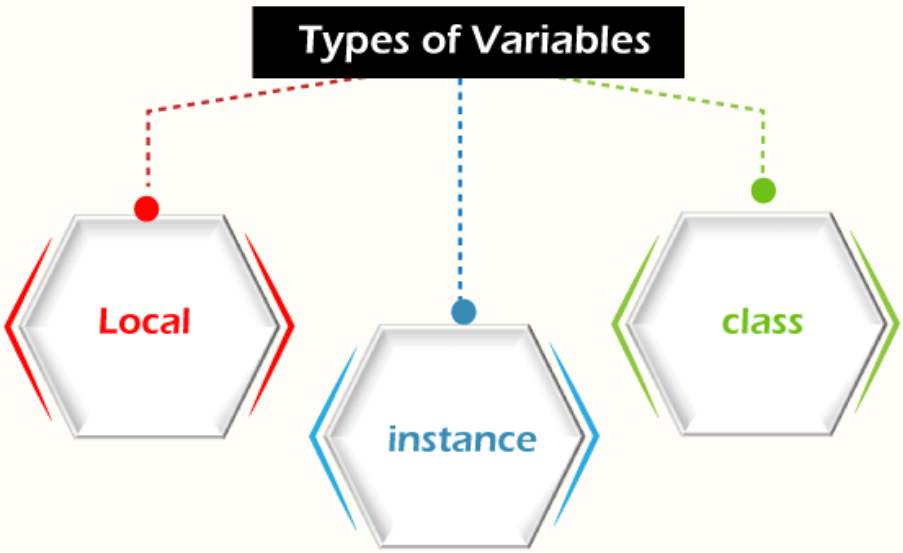


Levels of data abstraction


Salman Farsi salman.cuet.cse@gmail.com LinkedIn

## 36. What are the types of variables in OOP?

There are three types of variables:




**Instance Variable:** It is an object-level variable. It should be declared inside a class but must be outside a method, block, and constructor. It is created when an object is created by using the new keyword. It can be accessed directly by calling the variable name inside the class.

**Static Variable:** It is a class-level variable. It is declared with keyword **static** inside a class but must be outside of the method, block, and constructor. It stores in static memory. Its visibility is the same as the instance variable. The default value of a static variable is the same as the instance variable. It can be accessed by calling the **class_name.variable_name**.

**Local Variable:** It is a method-level variable. It can be declared in method, constructor, or block. Note that the use of an access modifier is not allowed with local variables. It is visible only to the method, block, and constructor in which it is declared. Internally, it is implemented at the stack level. It must be declared and initialized before use.

Another type of variable is used in object-oriented programming is the **reference** variable.

**Reference Variable:** It is a variable that points to an object of the class. It points to the location of the object that is stored in the memory.

## 37. Is it possible to overload a constructor?

**Yes**, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. For example:

```
public class Demo
{
Demo()
```

```
{
//logic
}
Demo(String str) //overloaded constructor
{
//logic
}
Demo(double d) //overloaded constructor
{
//logic
}
//statements
}
```

## 38. Can we overload the main() method in Java also give an example?

Yes, we can also overload the main() method in Java. Any number of main() methods can be defined in the class, but the method signature must be different. Consider the following code.

```
class OverloadMain
{
public static void main(int a)  //overloaded main method
{
System.out.println(a);
}
public static void main(String args[])
{
System.out.println("main method invoked");
main(6);
}
}
```

## 39. If a class Demo has a static block and a main() method. A print statement is presented in both. The question is which one will first execute, static block or the main() method, and why?

JVM first executes the static block on a priority basis. It means JVM first goes to static block even before it looks for the main() method in the program. After that main() method will be executed.

```
class Demo
{
static                  //static block
{
System.out.println("Static block");
}
public static void main(String args[])  //static method
{
System.out.println("Static method");
}
}
```

## 40. What is an Inline function?

An inline function is a technique used by the compilers and instructs to insert complete body of the function wherever that function is used in the program source code.

## 41. What is a virtual function?

A virtual function is a member function of a class, and its functionality can be overridden in its derived class. This function can be implemented by using a keyword called virtual, and it can be given during function declaration.

A virtual function can be declared using a token(virtual) in C++. It can be achieved in C/Python Language by using function pointers or pointers to function

## 42. What is a friend function?

A friend function is a friend of a class that is allowed to access to Public, private, or protected data in that same class. If the function is defined outside the class cannot access such information.

A friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public, or protected.

## 43. What is operator overloading?

Operator overloading is a function where different operators are applied and depends on the arguments. Operator,-,* can be used to pass through the function, and it has its own precedence to execute.

## 44. What is a ternary operator?

The ternary operator is said to be an operator which takes three arguments. Arguments and results are of different data types, and it depends on the function. The ternary operator is also called a conditional operator.

## 45. What is the use of finalize method?

Finalize method helps to perform cleanup operations on the resources which are not currently used. Finalize method is protected, and it is accessible only through this class or by a derived class.

## 46. What are the different types of arguments?

A parameter is a variable used during the declaration of the function or subroutine, and arguments are passed to the function body, and it should match with the parameter defined. There are two types of Arguments.

- Call by Value – Value passed will get modified only inside the function, and it returns the same value whatever it is passed into the function.
- Call by Reference – Value passed will get modified in both inside and outside the functions and it returns the same or different value.

## 47. What is the super keyword?

The super keyword is used to invoke the overridden method, which overrides one of its superclass methods. This keyword allows to access overridden methods and also to access hidden members of the superclass.

It also forwards a call from a constructor, to a constructor in the superclass.

## 48. What is an interface?

An interface is a collection of an abstract method. If the class implements an interface, it thereby inherits all the abstract methods of an interface.

Java uses Interface to implement multiple inheritances.

## 49. What is exception handling?

An exception is an event that occurs during the execution of a program. Exceptions can be of any type – Runtime exception, Error exceptions. Those exceptions are adequately handled through exception handling mechanism like try, catch, and throw keywords.

## 50. What are tokens?

A compiler recognizes a token, and it cannot be broken down into component elements. Keywords, identifiers, constants, string literals, and operators are examples of tokens.

Even punctuation characters are also considered as tokens. Example: Brackets, Commas, Braces, and Parentheses.

## 51. What are sealed modifiers?

Sealed modifiers are the access modifiers where the methods can not inherit it. Sealed modifiers can also be applied to properties, events, and methods. This modifier cannot be used to static members.

## 52. How can we call the base method without creating an instance?

Yes, it is possible to call the base method without creating an instance. And that method should be “Static method.”

Doing Inheritance from that class.-Use Base Keyword from a derived class.

## 53. What are the various types of constructors?

There are three types of constructors:

– Default Constructor – With no parameters.

– Parametric Constructor – With Parameters. Create a new instance of a class and also passing arguments simultaneously.

– Copy Constructor – Which creates a new object as a copy of an existing object.

## 54. What is early and late Binding?

Early binding refers to the assignment of values to variables during design time, whereas late Binding refers to the assignment of values to variables during run time.

## 55. What is 'this' pointer?

THIS pointer refers to the current object of a class. THIS keyword is used as a pointer which differentiates between the current object with the global object. It refers to the current object.

## 56. What is the difference between structure and a class?

The default access type of a Structure is public, but class access type is private. A structure is used for grouping data, whereas a class can be used for grouping data and methods. Structures are exclusively used for data, and it doesn't require strict validation, but classes are used to encapsulate and inherent data, which requires strict validation.

## 57. What is the default access modifier in a class?

The default access modifier of a class is Internal and the default access modifier of a class member is Private.

## 58. What is a pure virtual function?

A pure virtual function is a function which can be overridden in the derived class but cannot be defined. A virtual function can be declared as Pure by using the operator =0.

Example –

Virtual void function1() // Virtual, Not pure

Virtual void function2() = 0 //Pure virtual

## 59. What are all the operators that cannot be overloaded?

Following are the operators that cannot be overloaded -.

1. Scope Resolution (::)
2. Member Selection (.)
3. Member selection through a pointer to function (.*)

## 60. What is dynamic or run time polymorphism?

Dynamic or Run time polymorphism is also known as method overriding in which call to an overridden function is resolved during run time, not at the compile time. It means having two or more methods with the same name, same signature but with different implementation.

## 61. Do we require a parameter for constructors?

No, we do not require a parameter for constructors.

## 62. Whether static method can use nonstatic members?

False.

## 63. What are a base class, subclass, and superclass?

The base class is the most generalized class, and it is said to be a root class.

A Subclass is a class that inherits from one or more base classes.

The superclass is the parent class from which another class inherits.

## 64. What is static and dynamic Binding?

Binding is nothing but the association of a name with the class. Static Binding is a binding in which name can be associated with the class during compilation time, and it is also called as early Binding.

Dynamic Binding is a binding in which name can be associated with the class during execution time, and it is also called as Late Binding.

## 65. How many instances can be created for an abstract class?

Zero instances will be created for an abstract class. In other words, you cannot create an instance of an Abstract Class.

## 66. Which keyword can be used for overloading?

Operator keyword is used for overloading.

## 67. What is the default access specifier in a class definition?

Private access specifier is used in a class definition.

## 68. Which OOPS concept is used as a reuse mechanism?

Inheritance is the OOPS concept that can be used as a reuse mechanism.

## 69. Which OOPS concept exposes only the necessary information to the calling functions?

Encapsulation

## 70. Does C++ support multiple inheritance?

Yes, C++ does support multiple inheritance, which allows a class to inherit from more than one base class.

## 71. There is a problem with C++ multiple inheritance, what is that? How do you solve this problem?

The main problem with multiple inheritance in C++ is the Diamond Problem. This occurs when a class inherits from two classes that both inherit from a common base class. It can cause ambiguity about which base class's properties or methods should be used.

To solve this, we use **virtual inheritance**. By declaring the inheritance as virtual, C++ ensures that only one copy of the base class is inherited, preventing ambiguity.

## 72. Why is the final keyword used in C++?

The final keyword is used in C++ to prevent further inheritance. When applied to a class, it prevents the class from being inherited by other classes. When applied to a method, it prevents that method from being overridden in derived classes.

## 73. What is the Diamond Problem? And the Solution of it

The Diamond Problem occurs when a class inherits from two classes that share a common base class. When methods or properties from the common base class are called, the compiler is unsure which instance to use, leading to ambiguity.

The solution is virtual inheritance, which ensures only one instance of the common base class is inherited, eliminating the ambiguity.

## 74. What is the key difference between C++ and Java in terms of OOP?

One key difference is that C++ supports multiple inheritance, whereas Java does not. In Java, a class can implement multiple interfaces but can only extend one class. Another difference is that C++ has destructors, while Java relies on automatic garbage collection.

## 75. What is the destructor alternative in Java?

In Java, there is no direct equivalent to C++ destructors. Java uses **garbage collection** to automatically manage memory and resources. For specific cleanup, the finalize() method was once available, but it's deprecated, and now developers typically use **try-with-resources** or manual cleanup methods (like closing resources in finally blocks).

## 76. Why do we use Encapsulation, and what is the need for getters and setters?

Encapsulation is used to restrict direct access to an object's data and provide controlled access through methods (getters and setters). It helps protect the integrity of the object's data by controlling what is allowed to be changed or accessed. Getters and setters allow us to validate or process data before assignment or retrieval.

## 77. Can abstraction have implementation?

Yes, in C++, an abstract class can have implementation in the form of non-pure virtual functions. A pure virtual function (declared with = 0) must be implemented in derived classes, but non-pure virtual functions in an abstract class can have implementation and can be inherited by derived classes.

## 78. What is the difference between a Java interface and an abstract class?

**Abstract Class**: Can have both implemented and unimplemented (abstract) methods. It can also have member variables and constructors. A class can only extend one abstract class.

**Interface**: Can only have abstract methods (prior to Java 8) but can have default and static methods since Java 8. It cannot have instance variables. A class can implement multiple interfaces.

## 79. Exactly why we should use virtual keyword?

**1. Virtual Functions:**

The virtual keyword, when applied to member functions, supports polymorphism, a fundamental concept in object-oriented programming.

**Purpose of Virtual Functions:**

- **Dynamic Polymorphism:** Allows derived classes to override the function. The function call is resolved at runtime rather than compile-time, enabling polymorphic behavior.
- **Function Overriding:** Ensures that the derived class's version of the function is called, even when using a base class pointer or reference.

**Example:**

```
class Base {
public:
    virtual void show() {
        std::cout << "Base show" << std::endl;
    }
};

class Derived : public Base {
public:
    void show() override { // Override keyword is optional but good for
readability
        std::cout << "Derived show" << std::endl;
    }
};

int main() {
    Base* b = new Derived();
    b->show(); // Calls Derived's show function
    delete b;
    return 0;
}
```

In this example, show is a virtual function in Base. When show is called on a Base pointer pointing to a Derived object, the Derived's version of show is called.

**2. Virtual Inheritance:**

When used with classes, the virtual keyword is used to solve problems related to multiple inheritance, particularly the "Diamond Problem."

**Purpose of Virtual Inheritance:**

- **Avoiding Duplicate Base Classes:** Ensures that a base class is shared among all derived classes, preventing multiple instances of the base class in the inheritance hierarchy.

**Example:**

```
class Base {
public:
    void show() {
        std::cout << "Base show" << std::endl;
    }
};

class Derived1 : virtual public Base {
    // Derived1-specific implementation
};

class Derived2 : virtual public Base {
    // Derived2-specific implementation
};

class Final : public Derived1, public Derived2 {
    // Final-specific implementation
};
```

Without virtual inheritance, Final would have two instances of Base (one from Derived1 and one from Derived2). With virtual inheritance, Final has only one instance of Base.

## 80. What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

- Threads share the same address space.
- The thread is lightweight.
- The cost of communication between the processes is low.

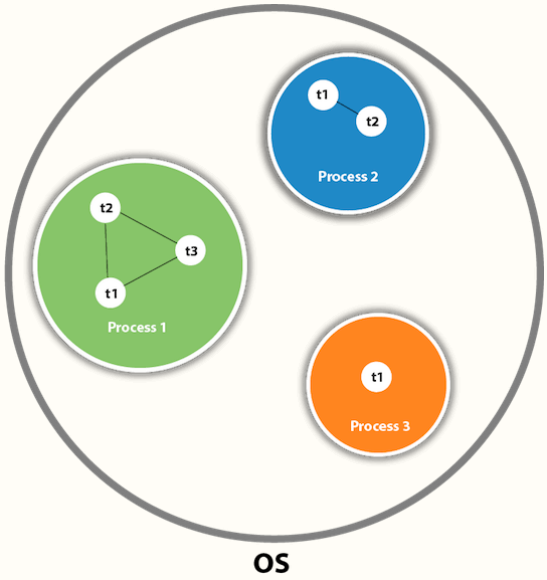
## 81. 2) What is the thread?

A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack frame. A process may contain multiple threads. Threads share the process resources, but still, they execute independently.

## 82. Differentiate between process and thread?

There are the following differences between the process and thread.

- A Program in the execution is called the process whereas; A thread is a subset of the process
- Processes are independent whereas threads are the subset of process.
- Process have different address space in memory, while threads contain a shared address space.
- Context switching is faster between the threads as compared to processes.
- Inter-process communication is slower and expensive than inter-thread communication.
- Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.




## 83. What do you understand by inter-thread communication?

- The process of communication between synchronized threads is termed as inter-thread communication.
- Inter-thread communication is used to avoid thread polling in Java.
- The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.
- It can be obtained by wait(), notify(), and notifyAll() methods.

## 84. What are the advantages of multithreading?

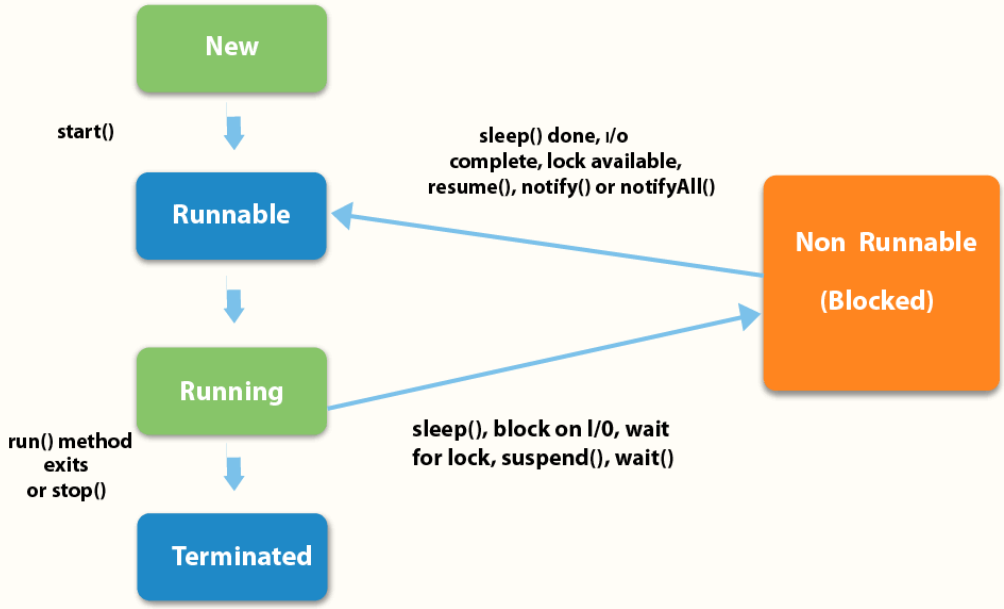
Multithreading programming has the following advantages:

- Multithreading allows an application/program to be always reactive for input, even already running with some background tasks
- Multithreading allows the faster execution of tasks, as threads execute independently.
- Multithreading provides better utilization of cache memory as threads share the common memory resources.
- Multithreading reduces the number of the required server as one server can execute multiple threads at a time.

## 85. What are the states in the lifecycle of a Thread?

A thread can have one of the following states during its lifetime:

1. **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
2. **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
3. **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
4. **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
5. **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.




## 86. What is context switching?

In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resumed from the same point later. Context switching enables the multiple processes to share the same CPU.

---


Salman Farsi salman.cuet.cse@gmail.com LinkedIn

## 87. Differentiate between the Thread class and Runnable interface for creating a Thread?

The Thread can be created by using two ways.

- By extending the Thread class
- By implementing the Runnable interface

However, the primary differences between both the ways are given below:

- By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class (if required).
- By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
- Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

## 88. Is it possible to start a thread twice?

No, we cannot restart the thread, as once a thread started and executed, it goes to the Dead state. Therefore, if we try to start a thread twice, it will give a runtimeException "java.lang.IllegalThreadStateException". Consider the following example.

```
public class Multithread1 extends Thread
{
   public void run()
    {
      try {
          System.out.println("thread is executing now........");
      } catch(Exception e) {
      }
    }
    public static void main (String[] args) {
        Multithread1 m1= new Multithread1();
        m1.start();
        m1.start();
    }
}
```

```
Output
thread is executing now........
Exception                in                thread                "main"
java.lang.IllegalThreadStateException
at java.lang.Thread.start(Thread.java:708)
at Multithread1.main(Multithread1.java:13)
```


Salman Farsi salman.cuet.cse@gmail.com LinkedIn

## 89. When should we interrupt a thread?

We should interrupt a thread when we want to break out the sleep or wait state of a thread. We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

## 90. What is the purpose of the Synchronized block?

The Synchronized block can be used to perform synchronization on any specific resource of the method. Only one thread at a time can execute on a particular resource, and all other threads that attempt to enter the synchronized block are blocked.

- Synchronized block is used to lock an object for any shared resource.
- The scope of the synchronized block is limited to the block on which, it is applied. Its scope is smaller than a method.

## 91. What is the deadlock?

Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.

## 92. How to detect a deadlock condition? How can it be avoided?

We can detect the deadlock condition by running the code on cmd and collecting the Thread Dump, and if any deadlock is present in the code, then a message will appear on cmd.

**Ways to avoid the deadlock condition in Java:**

- **Avoid Nested lock:** Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.
- **Avoid unnecessary locks:** we must avoid the locks which are not required.
- **Using thread join:** Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of the join method.

## 93. Does each thread have its stack in multithreaded programming?

Yes, in multithreaded programming every thread maintains its own or separate stack area in memory due to which every thread is independent of each other.

## 94. What is race-condition?

A Race condition is a problem which occurs in multithreaded programming when various threads execute simultaneously accessing a shared resource at the same time. The proper use of synchronization can avoid the Race condition.

## 95. What is the difference between Synchronous programming and Asynchronous programming regarding a thread?

**Synchronous programming:** In Synchronous programming model, a thread is assigned to complete a task and hence thread started working on it, and it is only available for other tasks once it will end the assigned task.

**Asynchronous Programming:** In Asynchronous programming, one job can be completed by multiple threads and hence it provides maximum usability of the various threads.

# C++ Special

## 96. What is C++? What are the advantages of C++?

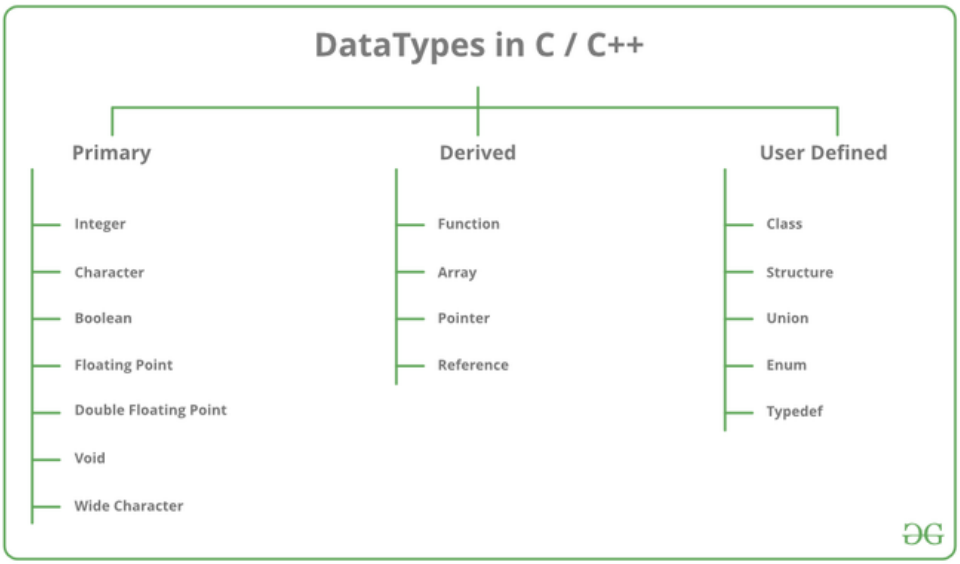
C++ is an object-oriented programming language that was introduced to overcome the jurisdictions where C was lacking. By object-oriented we mean that it works with the concept of *polymorphism*, *inheritance*, *abstraction*, *encapsulation*, *object, and class*.

***Advantages of C++:***

1. C++ is an OOPs language that means the data is considered as objects.
2. C++ is a multi-paradigm language; In simple terms, it means that we can program the logic, structure, and procedure of the program.
3. Memory management is a key feature in C++ as it enables dynamic memory allocation

4. It is a Mid-Level programming language which means it can develop games, desktop applications, drivers, and kernels

## 97. What are the different data types present in C++?




## 98. Define 'std'?

'**std'** is also known as Standard or it can be interpreted as a namespace. The command "*using namespace std"* informs the compiler to add everything under the *std namespace* and inculcate them in the *global namespace*. This all inculcation of global namespace benefits us to use "**cout**" and "**cin"** without using "**std::_operator_".**

## 99. What are references in C++?

In C++, references are an alternative way to create an alias for another variable. A reference acts as a synonym for a variable, allowing you to access the variable directly without any additional syntax. They must be initialized when created and cannot be changed to refer to another variable afterward. This feature makes it easier to manipulate variables in functions while avoiding the overhead of copying large objects. A reference variable is preceded with a **'&'** symbol.

**Syntax:**

```
int GFG = 10;
// reference variable
int& ref = GFG;
```


Salman Farsi salman.cuet.cse@gmail.com LinkedIn

## 100. What is the difference between reference and pointer?

| Reference | Pointer |
|---|---|
| The value of a reference cannot be reassigned | The value of a pointer can be reassigned |
| It can never hold a *null* value as it needs an existing value to become an alias of | It can hold or point at a *null* value and be termed as a *nullptr* or *null pointer* |
| It cannot work with arrays | It can work with arrays |
| To access the members of class/struct it uses a ‘ . ‘ | To access the members of class/struct it uses a ‘ -> ‘ |
| The memory location of reference can be accessed easily or it can be used directly | The memory location of a pointer cannot be accessed easily as we have to use a dereference ‘ * ‘ |

## 101. What is the difference between an array and a list?

| Arrays | Lists |
|---|---|
| **Array are contiguous memory locations of homogenous data types stored in a fixed location or size.** | Lists are classic individual elements that are linked or connected to each other with the help of pointers and do not have a fixed size. |
| **Arrays are static in nature.** | Lists are dynamic in nature |
| **Uses less memory than linked lists.** | Uses more memory as it has to store the value and the pointer memory location |

**What is the difference between new and malloc()?**

| new | malloc() |
|---|---|
| new is an operator which performs an operation | malloc is a function that returns and accepts values |
| new calls the constructors | malloc cannot call a constructor |
| new is faster than malloc as it is an operator | malloc is slower than new as it is a function |

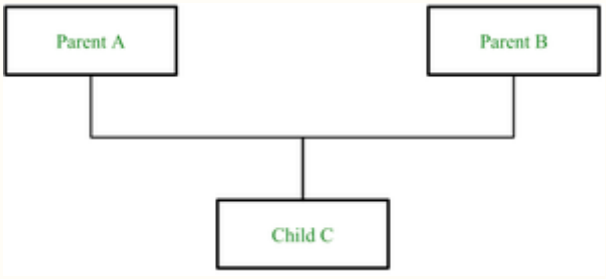
| | |
|---|---|
| new returns the exact data type | malloc returns void* |

## 102. What is the difference between virtual functions and pure virtual functions?

| Virtual Function | Pure Virtual Function |
|---|---|
| A Virtual Function is a member function of a base class that can be redefined in another derived class. | A Pure Virtual Function is a member function of a base class that is only declared in a base class and defined in a derived class to prevent it from becoming an abstract class. |
| A virtual Function has its definition in its respective base class. | There is no definition in Pure Virtual Function and is initialized with a pure specifier **(= 0).** |
| The base class has a virtual function that can be represented or instanced; In simple words, its object can be made. | A base class having pure virtual function becomes abstract that cannot be represented or instanced; In simple words, it means its object cannot be made. |

## 103. When should we use multiple inheritance?

Multiple inheritances mean that a derived class can inherit two or more base/parent classes. It is useful when a derived class needs to combine numerous attributes/contracts and inherit some, or all, of the implementation from these attributes/contracts. To take a real-life example consider your Parents where Parent A is your DAD Parent B is your MOM and Chid C is you.




**21. What is virtual inheritance?**

Virtual inheritance is a technique that ensures only one copy of a base class's member variables is inherited by grandchild-derived classes. Or in simple terms, virtual inheritance is used when

we are dealing with a situation of multiple inheritances but want to prevent multiple instances of the same class from appearing in the inheritance hierarchy.

**27. What is a virtual destructor?**

When destroying instances or objects of a derived class using a base class pointer object, a virtual destructor is invoked to free up memory space allocated by the derived class object or instance.

Virtual destructor guarantees that first the derived class' destructor is called. Then the base class's destructor is called to release the space occupied by both destructors in the inheritance class which saves us from the memory leak. It is advised to make your destructor virtual whenever your class is polymorphic.

**28. Is destructor overloading possible? If yes then explain and if no then why?**

The simple answer is **NO** we cannot overload a destructor. It is mandatory to only destructor per class in C++. Also to mention, Destructor neither take arguments nor they have a parameter that might help to overload.

**29. Which operations are permitted on pointers?**

Pointers are the variables that are used to store the address location of another variable. Operations that are permitted to a pointer are:

1. Increment/Decrement of a Pointer
2. Addition and Subtraction of integer to a pointer
3. Comparison of pointers of the same type

**30. What is the purpose of the "*delete*" operator?**

The delete operator is used to delete/remove all the characteristics/properties from an object by deallocating its memory; furthermore, it returns true or false in the end. In simple terms, it destroys or deallocates array and non-array(pointer) objects which are created by new expressions.

```
int GFG = new int[100];
// uses GFG for deletion
delete[] GFG;
```

## 104.31. How delete [] is different from delete?

| delete[] | delete |
|---|---|
| It is used for deleting a whole array | It is used to delete only one single pointer |

| | |
|---|---|
| It is used for deleting the objects of new[]; By this, we can say that delete[] is used to delete an array of objects | It is used for deleting the objects of **new;** By this, we can say that **delete** is used to delete a single object |
| It can call as many destructors it wants | It can only call the destructor of a class once |

## 105. What do you know about friend class and friend function?

A friend class is a class that can access both the protected and private variables of the classes where it is declared as a friend.

**Example of friend class:**

```
class Class_1st {
    // ClassB is a friend class of ClassA
    friend class Class_2nd;
    statements;
} class Class_2nd {
    statements;
}
```

A friend function is a function used to access the private, protected, and public data members or member functions of other classes. It is declared with a friend keyword. The advantage of a friend function is that it is not bound to the scope of the class and once it is declared in a class, furthermore to that, it cannot be called by an object of the class; therefore it can be called by other functions. Considering all the mentioned points we can say that a friend function is a global function**.**

**Example of friend function:**

```
class GFG {
    statements;
    friend datatype function_Name(arguments);
    statements;
} OR class GFG {
    statements' friend int divide(10, 5);
    statements;
}
```

## 106. What is an Overflow Error?

Overflow Error occurs when the number is too large for the data type to handle. In simple terms, it is a type of error that is valid for the defined but exceeds used the defined range where it should coincide/lie.

For example, the range of int data type is **–2,147,483,648** to **2,147,483,647** and if we declare a variable of size **2,247,483,648** it will generate a overflow error.

## 107. What does the Scope Resolution operator do?

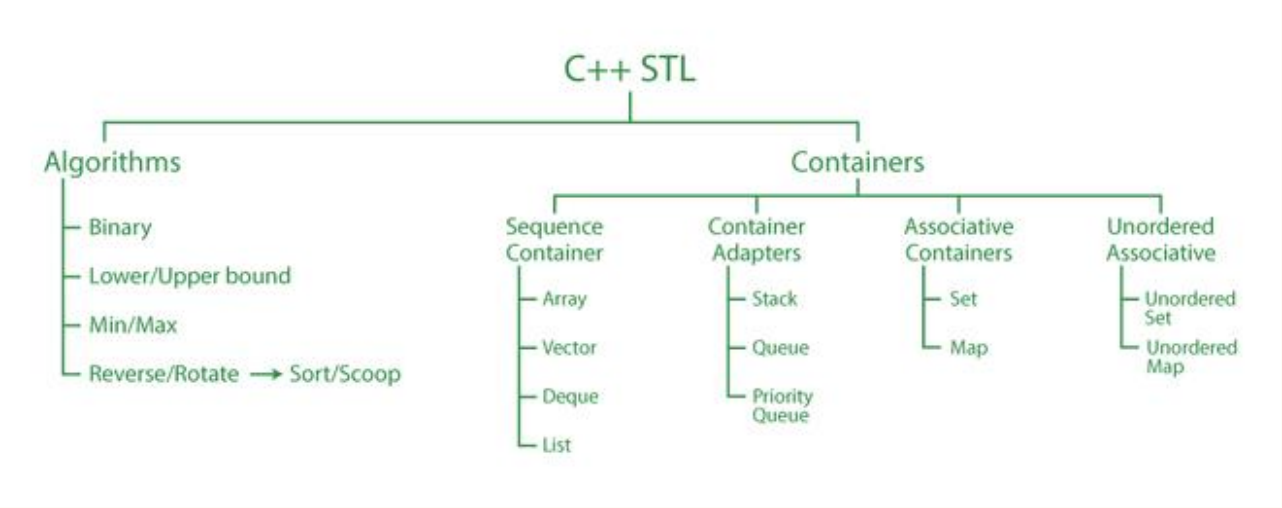
A scope resolution operator is denoted by a '**::**' symbol. Just like its name this operator resolves the barrier of scope in a program. A scope resolution operator is used to reference a member function or a global variable out of their scope furthermore to which it can also access the concealed variable or function in a program.

Scope Resolution is used for numerous amounts of tasks:

1. To access a global variable when there is a local variable with the same name
2. To define the function outside the class
3. In case of multiple inheritances
4. For namespace

**37. What is STL?**

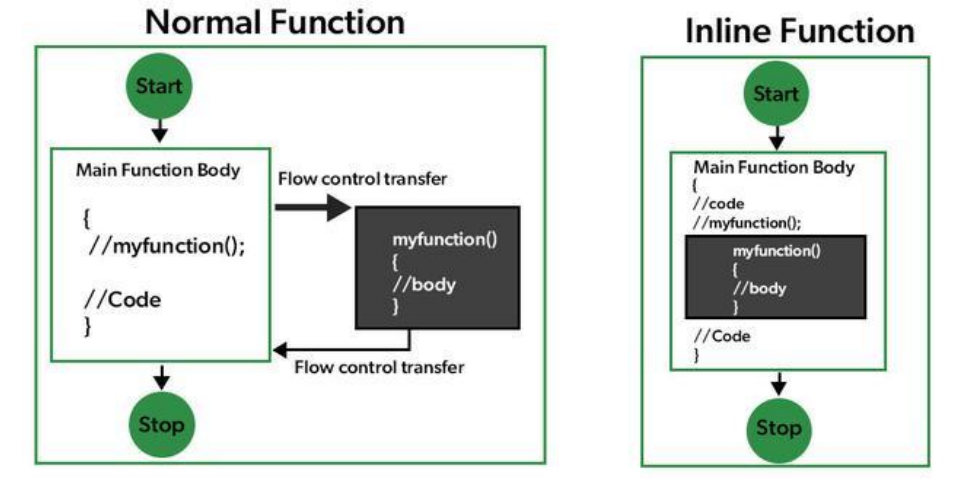
STL is known as Standard Template Library, it is a library that provides 4 components like container, algorithms, and iterators.




## 108. Define inline function. Can we have a recursive inline function in C++?

An inline function is a form of request not an order to a compiler which results in the inlining of our function to the main function body. An inline function can become overhead if the execution time of the function is less than the switching time from the caller function to called

function. To make a function inline use the keyword **inline** before and define the function before any calls are made to the function.




**Syntax:**

```
inline data_type function_name()
{
Body;
}
```

The answer is **No;** It cannot be recursive.

An inline function cannot be recursive because in the case of an inline function the code is merely placed into the position from where it is called and does not maintain a piece of information on the stack which is necessary for recursion.

Plus, if you write an inline keyword in front of a recursive function, the compiler will automatically ignore it because the inline is only taken as a suggestion by the compiler.

## 109. What is an abstract class and when do you use it?

An abstract class is a class that is specifically designed to be 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 the class declaration

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. However, it can be used to declare pointers and references to an abstract class.

An abstract class is used if you want to provide a common, implemented functionality among all the implementations of the component. Abstract classes will allow you to partially implement your class, whereas interfaces would have no implementation for any members whatsoever. In simple words, Abstract Classes are a good fit if you want to provide

implementation details to your children but don't want to allow an instance of your class to be directly instantiated.

## 110. What is the main use of the keyword "Volatile"?

Just like its name, things can change suddenly and unexpectantly; So it is used to inform the compiler that the value may change anytime. Also, the volatile keyword prevents the compiler from performing optimization on the code. It was intended to be used when interfacing with memory-mapped hardware, signal handlers, and machine code instruction.

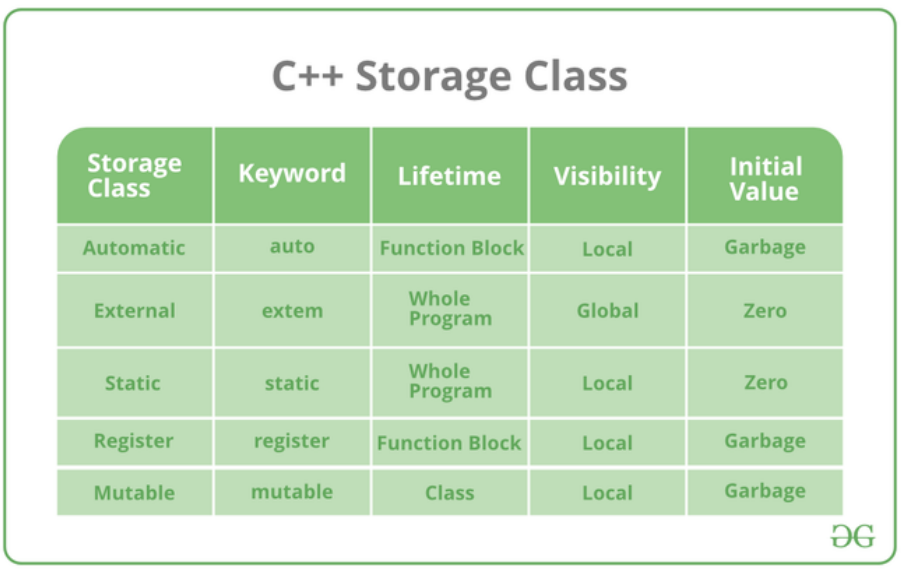
## 111. Define storage class in C++ and name some

Storage class is used to define the features(lifetime and visibility) of a variable or function. These features usually help in tracing the existence of a variable during the runtime of a program.

**Syntax:**

storage_class var_data_type var_name;

**Some types of storage classes:**




## 112. What is a mutable storage class specifier? How can they be used?

Just like its name, the mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const. Static or

const, or reference members cannot use the mutable specifier. When we declare a function as const, this pointer passed to the function becomes const.

## 113. Define the Block scope variable.

So the scope of a variable is a region where a variable is accessible. There are two scope regions, A global and block or local.

A block scope variable is also known as a local scope variable. A variable that is defined inside a function (like main) or inside a block (like loops and if blocks) is a local variable. It can be used ONLY inside that particular function/block in which it is declared. a block-scoped variable will not be available outside the block even if the block is inside a function.

## 114. What is the function of the keyword “Auto”?

The auto keyword may be used to declare a variable with a complex type in a straightforward fashion. You can use auto to declare a variable if the initialization phrase contains templates, pointers to functions, references to members, etc. With type inference capabilities, we can spend less time having to write out things the compiler already knows. As all the types are deduced in the compiler phase only, the time for compilation increases slightly but it does not affect the runtime of the program.

## 115. Define namespace in C++

Namespaces enable us to organize named items that would otherwise have global scope into smaller scopes, allowing us to give them namespace scope. This permits program parts to be organized into distinct logical scopes with names. The namespace provides a place to define or declare identifiers such as variables, methods, and classes.

Or we could say that A namespace is a declarative zone that gives the identifiers (names of types, functions, variables, and so on) within it a scope. Namespaces are used to arrange code into logical categories and to avoid name clashes, which might happen when you have many libraries in your code base.

## 116. What is the difference between shallow copy and deep copy?

| Shallow Copy | Deep Copy |
| --- | --- |

| | |
|---|---|
| In Shallow copy, a copy of the original object is stored and only the reference address is finally copied. In simple terms, Shallow copy duplicates as little as possible | In Deep copy, the copy of the original object and the repetitive copies both are stored. In simple terms, Deep copy duplicates everything |
| A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share individual elements. | A deep copy of a collection is two collections with all of the elements in the original collection duplicated. |
| A shallow copy is faster | Deep copy is comparatively slower. |

## 117. Can we call a virtual function from a constructor?

Yes, we can call a virtual function from a constructor. But it can throw an exception of overriding.

## 118. What are void pointers?

Just like its name a void pointer is a pointer that is not associated with anything or with any data type. Nevertheless, a void pointer can hold the address value of any type and can be converted from one data type to another.


Salman Farsi salman.cuet.cse@gmail.com LinkedIn

# References:


[1] Different Books of Computer Science

[2] www.geeksforgeeks.com

[3] www.tutorialspoint.com

[4] https://www.javatpoint.com/

[5] https://www.interviewbit.com/


# End Notes

I tried to keep the answers to most of the questions descriptive so that the candidates can understand the topic thoroughly. From my experience, I can say that it is very useful to answer the counter questions usually asked immediately after you answer a question. Hope you will find it helpful; I invested a lot of time organizing all these things. As a reward, I will be more than happy if it really helps you in some way.