

OOPs Interview Questions

1. What is OOPS?

OOPS is abbreviated as Object Oriented Programming system in which programs are considered as a collection of objects. Each object is nothing but an instance of a class.

2. Write basic concepts of OOPS?

1. Abstraction. 2. Encapsulation. 3. Inheritance. 4. Polymorphism.

3. What is a class?

A class is simply a representation of a type of object. It is the blueprint/ plan/ template that describe the details of an object.

| Java Class | C++ Class |
|---|--|
| <pre>public class Interview { // Attributes private int ID; private String name; // Functions public void functionName() { } }</pre> | <pre>class Interview { private: int id; string name; public: void functionName() { } };</pre> |

4. What is an object?

Object is termed as an instance of a class, and it has its own state, behavior and identity.

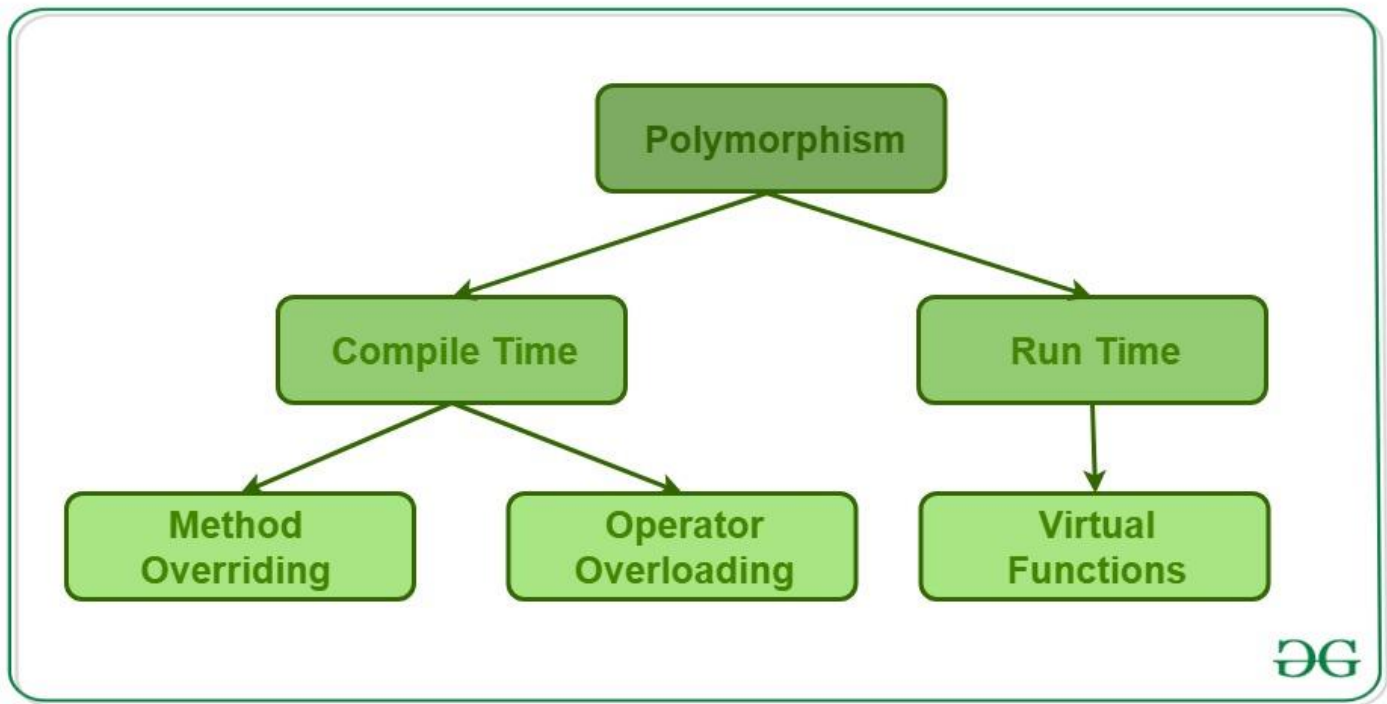
| Java Object | C++ Object |
|--|----------------------------------|
| <pre>Interview objectName = new Interview();</pre> | <pre>Interview objectName;</pre> |

5. 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.

6. What is Polymorphism?

Polymorphism is nothing but assigning behavior or value in a subclass to something that was already declared in the main class. Simply, polymorphism takes more than one form.



C++ Polymorphism

Early binding OR Compile time polymorphism using Function overloading.

```
#include <iostream>
using namespace std;
class Interview    // Compile Time polymorphism
{                // With function overloading.
public:
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl;
    }
};
int main()
{
    Interview in;
    in.func(7);
    in.func(9.132);
    in.func(85, 64);
    return 0;
}
```

Output

```
C:\Users\User\documents\visual studio 2013\Projects\Hellow_world App\Debug\Hellow_world App.exe
value of x is 7
value of x is 9.132
value of x and y is 85, 64
Press any key to continue . . .
```

Early binding OR compile time polymorphism using operator overloading.

```
#include<iostream>
using namespace std;
class Complex
{
private:
    int real, imag;
public:
    Complex(int r = 0, int i = 0) { real = r; imag = i; }
    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex const &obj) {
        Complex result;
        result.real = real + obj.real;
        result.imag = imag + obj.imag;
        return result;
    }
    void print() { cout <<"Real " << real << " Imaginary" << imag << endl; }
};
void main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
    system("pause");
}
```

Output.

```
C:\Users\User\documents\visual studio 2013\Projects\Hellow_world App\Debug\Hello
Real 12 Imaginary 9
Press any key to continue . . .
```

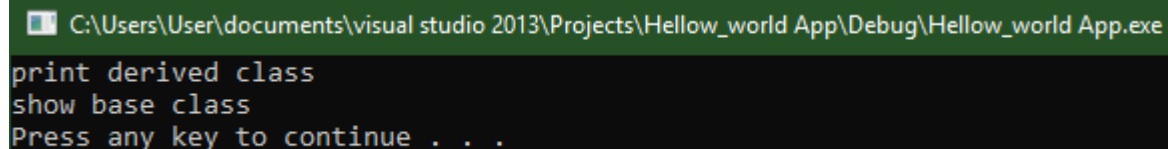
Run time polymorphism OR late binding using virtual functions/ Function Overriding.

```
#include <iostream>
using namespace std;
class base
{
public:
    virtual void print(){
        cout << "print base class" << endl;
    }
    void show(){
        cout << "show base class" << endl;
    }
};

class derived :public base
{
public:
    void print(){
        //print () is already virtual function in derived class,
        //we could also declared as virtual void print () explicitly
        cout << "print derived class" << endl;
    }
    void show(){
        cout << "show derived class" << endl;
    }
};

void main()
{
    base *bptr;
    derived d;
    bptr = &d;
    bptr->print();//virtual function, binded at runtime
    bptr->show();// Non-virtual function, binded at compile time
    system("pause");
}
```

Output



```
C:\Users\User\documents\visual studio 2013\Projects\Hellow_world App\Debug\Hellow_world App.exe
print derived class
show base class
Press any key to continue . . .
```

Compile time polymorphism OR early binding using functions overloading.

```
package com.company;

class MultiplyFun
{
    public int Multiply(int a, int b){
        return a * b;
    }
    public double Multiply(double a, double b){
        return a * b;
    }
}

class Interview
{
    public static void main(String[] args)
    {
        MultiplyFun multiplyFun = new MultiplyFun();
        System.out.println(multiplyFun.Multiply(a: 2, b: 4));
        System.out.println(multiplyFun.Multiply(a: 5.5, b: 6.3));
    }
}
```

Output:

```
8
34.65
```

Compile time polymorphism OR early binding using Operator overloading.

```
package com.company;
class OperatorOVERDDN
{
    void operator(String str1, String str2)    {
        String s = str1 + str2;
        System.out.println("Concatinated String - " + s);
    }
    void operator(int a, int b){
        int c = a + b;
        System.out.println("Sum = " + c);
    }
}
class Interview
{
    public static void main(String[] args){
        OperatorOVERDDN obj = new OperatorOVERDDN();
        obj.operator( a: 2, b: 3);
        obj.operator("joe", "now");
    }
}
```

Output:

```
Sum = 5
Concatinated String - joenow
```

Run time polymorphism OR late binding using functions overriding.

```
package com.company;
class Parent {
    void Print(){
        System.out.println("parent class");
    }
}
class subclass1 extends Parent {
    void Print(){
        System.out.println("subclass1");
    }
}
class subclass2 extends Parent {
    void Print(){
        System.out.println("subclass2");
    }
}
class Interview {
    public static void main(String[] args){
        Parent a;
        a = new subclass1();
        a.Print();
        a = new subclass2();
        a.Print();
    }
}
```

Output:

```
subclass1
subclass2
```

7. What is Inheritance?

Inheritance is a concept where one class shares the structure and behavior defined in another class. If inheritance applied on one class is called Single Inheritance, and if it depends on multiple classes, then it is called multiple Inheritance.

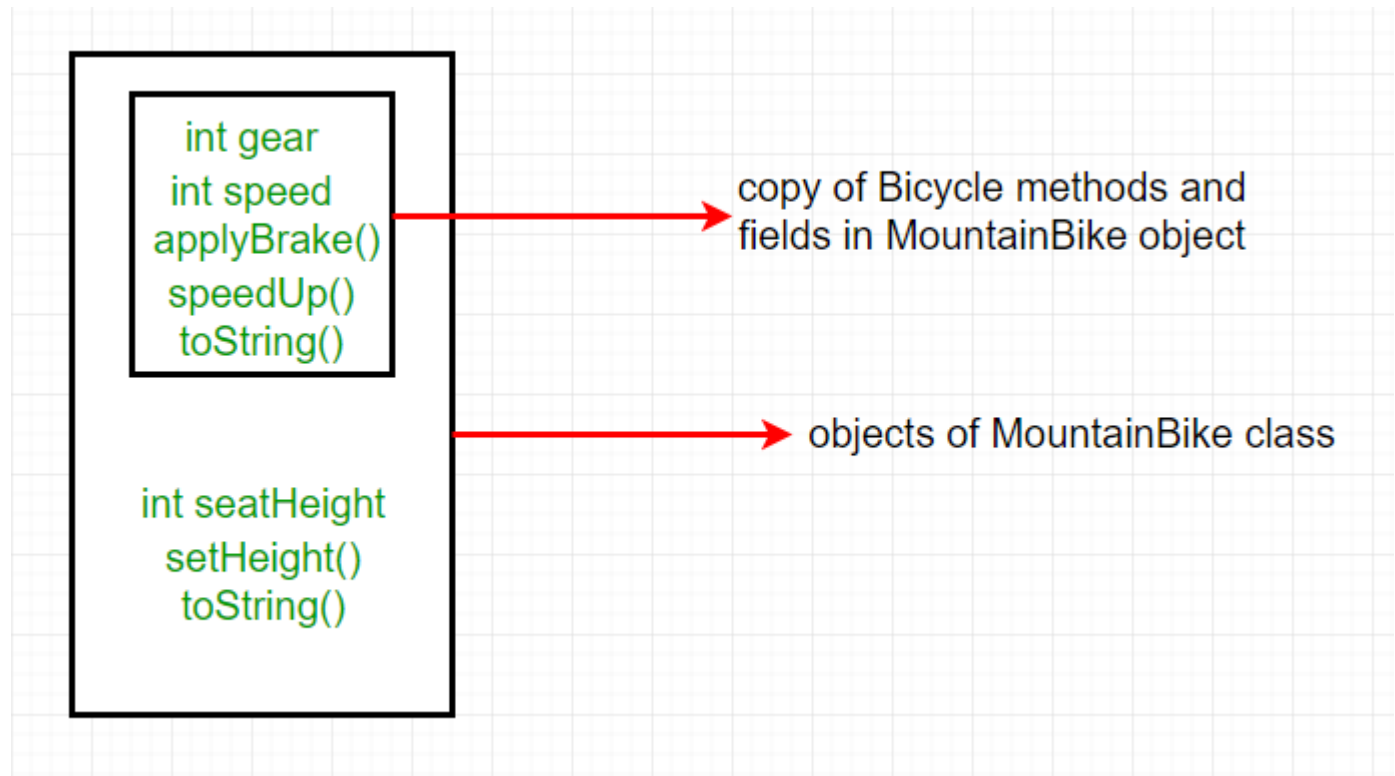
| Inheritance Types | Implementation in Java using | Implementation in C++ using |
|------------------------------|------------------------------|-------------------------------------|
| Single Inheritance | classes | classes |
| Multiple Inheritance | Interfaces | classes |
| Multilevel Inheritance | classes | Classes |
| Hierarchical Inheritance | classes | Classes |
| Hybrid (Virtual) Inheritance | interfaces | Classes (it creates Dimond problem) |

Java: <https://www.geeksforgeeks.org/inheritance-in-java/>

C++ <https://www.geeksforgeeks.org/inheritance-in-c/>

Diagrams of Inheritance

Overall Structure of inheritance.

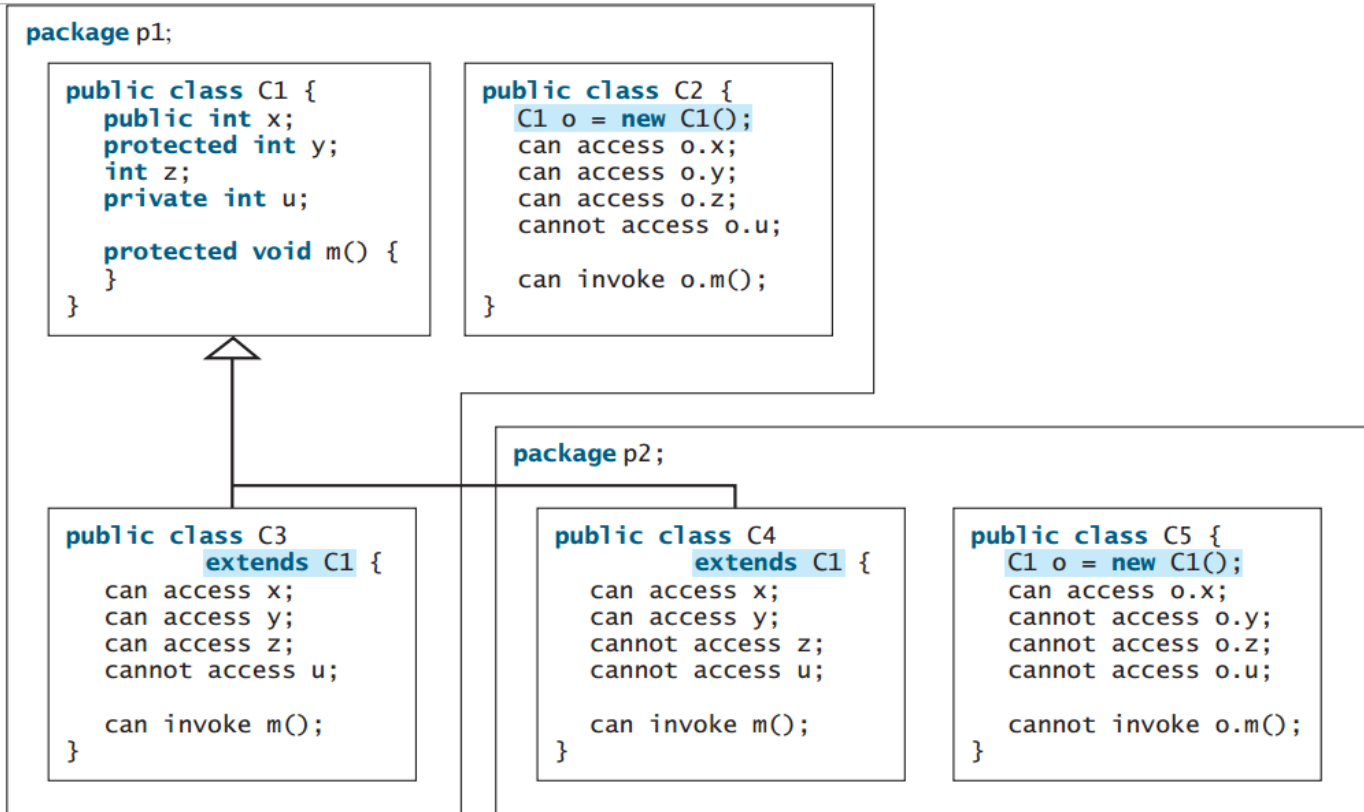


Data and Function Access in base class in c++.

| Base class member access specifier | Type of Inheritance | | |
|------------------------------------|-------------------------|-------------------------|-------------------------|
| | Public | Protected | Private |
| Public | Public | Protected | Private |
| Protected | Protected | Protected | Private |
| Private | Not accessible (Hidden) | Not accessible (Hidden) | Not accessible (Hidden) |

Data and Function Access in base class in different packages and classes in java.

| <i>Modifier on members in a class</i> | <i>Accessed from the same class</i> | <i>Accessed from the same package</i> | <i>Accessed from a subclass in a different package</i> | <i>Accessed from a different package</i> |
|---|---|---|--|--|
| public | ✓ | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | ✓ | - |
| default (no modifier) | ✓ | ✓ | - | - |
| private | ✓ | - | - | - |



8. What are manipulators?

Manipulators are the functions which can be used in conjunction with the insertion (<<) and extraction (>>) operators on an object. Examples are endl and setw.

9. Define a constructor?

Constructor is a method used to initialize the state of an object, and it gets invoked at the time of object creation. Rules for constructor are:

- Constructor Name should be same as class name.
- Constructor must have no return type.

10. Define Destructor?

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.

11. What is Inline function?

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.

Note: No, Java does not provide inline functions it is typically done by the JVM at execution time.

Remember, inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining. Compiler may not perform inlining in such circumstances like:

- 1) If a function contains a loop. (for, while, do-while)
- 2) If a function contains static variables.
- 3) If a function is recursive.
- 4) If a function return type is other than void, and the return statement doesn't exist in function body.
- 5) If a function contains switch or goto statement.

```
#include <iostream>
using namespace std;
class operation{
    int a, b, add, sub, mul; float div;
public:
    void get(); void sum(); void difference();
};
inline void operation::get(){
    cout << "Enter two int valeus:";
    cin >> a;   cin >> b;
}
inline void operation::sum(){
    add = a + b;
    cout << "Addition of two numbers: " << a + b << "\n";
}
inline void operation::difference(){
    sub = a - b;
    cout << "Difference of two numbers: " << a - b << "\n";
}
int main(){
    operation s;
    s.get();
    s.sum();
    s.difference();
    system("pause");
    return 0;
}
```

12. What is a virtual function?

Virtual function is a member function of 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. Virtual function can be achieved in C++, and it can be achieved in C Language by using function pointers or pointers to function.

Example: Already provided at pages 4 (In this Document).

13. What is friend function?

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

<https://www.geeksforgeeks.org/friend-class-function-cpp/>

| Global Friend Function | Friend function of a Class |
|--|--|
| <pre>#include <iostream> class A { int a; public: A() { a = 0; } // global friend function friend void showA(A&); }; void showA(A& x){ // Since showA() is a friend, //it can access // private members of A std::cout << "A::a=" << x.a; } int main(){ A a; showA(a); return 0; }</pre> | <pre>#include <iostream> class B; class A { public: void showB(B&); }; class B { private: int b; public: B() { b = 0; } friend void A::showB(B& x); }; void A::showB(B& x){ // Since showB() is friend of B //it can access private members of B std::cout << "B::b = " << x.b; } int main(){ A a; B x; a.showB(x); system("pause"); return 0; }</pre> |

14. What is function overloading?

Function overloading is defined as a normal function, but it has the ability to perform different tasks. It allows creation of several methods with the same name which differ from each other by type of input and output of the function.

Example

```
void add (int& a, int& b);
```

```
void add (double& a, double& b);  
void add (struct bob& a, struct bob& b);
```

15. 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 their own precedence to execute. Example: Already given above at page 4 (in this document).

16. What is an abstract class?

An abstract class is a class which cannot be instantiated. Creation of an object is not possible with an abstract class, but it can be inherited. An abstract class can contain only Abstract method.

17. What is a ternary operator?

Ternary operator is said to be an operator which takes three arguments. Arguments and results are of different data types, and it is depending on the function. Ternary operator is also called as conditional operator.

18. 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.

19. What are 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, 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 it 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.

20. What is super keyword?

Super keyword is used to invoke 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.

21. What is method overriding?

Method overriding is a feature that allows sub class to provide implementation of a method that is already defined in the main class. This will override the implementation in the superclass by providing the same method name, same parameter and same return type.

22. What is an interface?

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

23. What is exception handling?

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

24. What are tokens?

Token is recognized by a compiler 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 – Brackets, Commas, Braces and Parentheses.

25. Difference between overloading and overriding?

Overloading is static binding whereas Overriding is dynamic binding. Overloading is nothing but the same method with different arguments, and it may or may not return the same value in the same class itself. Overriding is the same method names with same arguments and return types associates with the class and its child class.

26. Difference between class and an object?

An object is an instance of a class. Objects hold any information, but classes don't have any information. Definition of properties and functions can be done at class and can be used by the object. Class can have sub-classes, and an object doesn't have sub-objects.

27. What is an abstraction?

Abstraction is a good feature of OOPS, and it shows only the necessary details to the client of an object. Means, it shows only necessary details for an object, not the inner details of an object. Example – When you want to switch on television, it not necessary to show all the functions of TV. Whatever is required to switch on TV will be showed by using abstract class.

28. What are access modifiers?

Access modifiers determine the scope of the method or variables that can be accessed from other various objects or classes. There are 5 types of access modifiers, and they are as follows:

- Private.
- Protected.
- Public.
- Friend.
- Protected Friend.

29. What is sealed modifiers?

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

30. 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 derived class.

31. 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.

32. What are the various types of constructors?

There are three various types of constructors, and they are as follows:

- 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.

33. What is early and late binding?

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

34. 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. Basically, it refers to the current object.

35. What is the difference between structure and a class?

Structure default access type is public, but class access type is private. A structure is used for grouping data whereas 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 encapsulates and inherit data which requires strict validation.

36. What is the default access modifier in a class?

The default access modifier of a class is Private by default.

37. What is 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.

38. 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 (. *)

39. 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.

40. Do we require parameter for constructors?

No, we do not require parameter for constructors.

41. What is a copy constructor?

This is a special constructor for creating a new object as a copy of an existing object. There will be always only one copy constructor that can be either defined by the user or the system.

42. What does the keyword virtual represented in the method definition?

It means, we can override the method.

43. Whether static method can use non-static members?

False.

44. What are base class, sub class and super class?

1. Base class is the most generalized class, and it is said to be a root class.
2. Sub class is a class that inherits from one or more base classes.
3. Super class is the parent class from which another class inherits.

45. 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.

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

Zero instances will be created for an abstract class.

47. Which keyword can be used for overloading?

Operator keyword is used for overloading.

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

Private access specifier is used in a class definition.

49. Which OOPS concept is used as reuse mechanism?

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

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

Data Hiding / Abstraction

51. What is difference between Abstract class and interface?

| Abstract class | Interface |
|--|--|
| Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |
| A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| Example: public abstract class Shape{ public abstract void draw(); } | Example: public interface Drawable{ void draw(); } |

52. Difference Between virtual and pure virtual function?

| Virtual functions | Pure Virtual functions |
|--|---|
| It has its definition (function body) in base class. | It has no (function body) definition in base class |
| Virtual Function is declared with keyword ' virtual ' at the declaration or definition. | Pure Virtual Function is declared with keyword ' virtual ' at the declaration. |
| All derived classes may or may not override, virtual function | All derived classes must override, pure virtual function. |
| These are hierarchical in nature so it does not affect if any derived class does not inherit it. | Gives compilation error if derived classes are not inherited. |
| No concept of abstract classes. | If class is pure virtual function, then class is declared as abstract class |
| Example: virtual void name(arguments); | Example: virtual void name(arguments)=0; |
| These are mostly used for polymorphism. | These are mostly used for Abstraction. |

53. Are there Virtual function in java?

In **Java**, all non-static methods are by default "**virtual functions**." Only methods marked with the **keyword final**, which cannot be overridden, along with **private methods**, which are not inherited, are **non-virtual**.

54. What is Abstraction?

Abstraction is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user.

| Abstraction in c++ | Abstraction in java |
|---|--|
| In C++ abstraction can achieved by abstract classes and Header files. | In java abstraction can achieved by interfaces and abstract classes. |
| <p>Abstraction in c++ using abstract classes.</p> <pre>#include<iostream> using namespace std; class Base { protected: int x; public: virtual void fun() = 0; Base(int i) { x = i; } }; class Derived : public Base{ int y; public: Derived(int i, int j) :Base(i) { y = j; } void fun() { cout << "x = " << x << ", y = " << y; } }; int main(void){ Derived d(4, 5); d.fun(); return 0; }</pre> | <p>Abstraction in java using interfaces.</p> <pre>package com.company; interface printable { void print(); } class A6 implements printable { public void print() { System.out.println("Hello"); } public static void main(String args[]) { A6 obj = new A6(); obj.print(); } }</pre> |

Abstraction in c++ using Header files.

One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating power of numbers.

Abstraction in java using Abstraction classes.

```
package com.company;
abstract class Bike
{
    abstract void run();
}
class Honda4 extends Bike
{
    void run()
    {
        System.out.println("running safely");
    }
    public static void main(String args[])
    {
        Bike obj = new Honda4();
        obj.run();
    }
}
```

55. Difference between Abstraction and Encapsulation?

| Parameter | Abstraction | Encapsulation |
|----------------|---|---|
| Use for | Abstraction solves the problem and issues that arise at the design stage. | Encapsulation solves the problem and issue that arise at the implementation stage. |
| Focus | Abstraction allows you to focus on what the object does instead of how it does it | Encapsulation enables you to hide the code and data into a single unit to secure the data from the outside world. |
| Implementation | You can use abstraction using Interface and Abstract Class. | You can implement encapsulation using Access Modifiers (Public, Protected & Private.) |
| Focuses | Focus mainly on what should be done. | Focus primarily on how it should be done. |
| Application | During design level. | During the Implementation level. |