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) What is an abstraction?

= Abstraction is a useful feature of OOPS, and it shows only the necessary details to the client of an object. Meaning, it shows only required details for an object, not the inner constructors, of an object.

Example – When you want to switch on the television, it is not necessary to know the inner circuitry/mechanism needed to switch on the TV. Whatever is required to switch on TV will be shown by using an abstract class.

3) 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.

4) Explain the relationship among abstraction and encapsulation?

= Encapsulation: Wrapping code and data together into a single unit. Class is an example of encapsulation, because it wraps the method and property. Abstraction: Hiding internal details and showing functionality only.

Abstraction focus on what the object does instead of how it does.

In abstraction, implementation complexities are hidden using abstract classes and interfaces. While in encapsulation, the data is hidden using methods of getters and setters. The objects that help to perform abstraction are encapsulated.

Whereas the objects that result in encapsulation need not be abstracted.

5) What is Polymorphism?

= Polymorphism is nothing but assigning behaviour or value in a subclass to something that was

already declared in the main class. Simply, polymorphism takes more than one form.

6) What is Inheritance?

= Inheritance is a concept where one class shares the structure and behavior defined in another

class. If Inheritance applied to one class is called Single Inheritance, and if it depends on

multiple classes, then it is called multiple Inheritance.

7) How composition is better than inheritance ?

= Prefer composition over inheritance as it is more malleable / easy to modify later, but do not use a compose-always approach. With composition, it's easy to change behavior on the fly with Dependency Injection / Setters. Inheritance is more rigid as most languages

do not allow you to derive from more than one type.

8) Which OOPS concept is used as a reuse mechanism?

= In OOP, The concept of inheritance provide the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will

have the combined features of both the classes.

9) Which OOPS Concept exposes only the necessary information to the calling function?

= Abstraction is a useful feature of OOPS, and it shows only the necessary details to the client of an object.

Meaning, it shows only required details for an object, not the inner constructors, of an object. Example – When you want to switch on the television, it is not

necessary to know the inner circuitry/mechanism needed to switch on the TV.

10) What is a class?

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

11. Using above created class, Write in brief abstraction and encapsulation

Ans: In above class Employee is class which is bind the data and member function together which is known is encapsulation and calling which is not showing is called abstraction

12. Explain difference among class and object?

ans:

| Sr.No. | class |
|---|---|
| 1 | Class is collection of object |
| 2 | Class is logical entity |
| 3 | In class definition get memory space |
| 4 | Class has data member and member function |

| Sr.No. | object |
|---|---|
| 1 | Object is instance of class |
| 2 | Object is physical entity |
| 3 | Object is get memory space |
| 4 | Object is used to access data and member function |

13. Define access modifiers?

ans : This type of modifier is used to control the access level of the class, attributes, methods, and constructor.

Types of Access Modifiers:

public Access modifiers: public Access is less restrictive than all other modifiers it can access whole java universe.

Default Access Modifiers: It can access in same package only.

Protected Access Modifiers: In proctected access modifiers we can access within package as well as outside class by using child class only.

Private Access Modifiers: Private access modifiers can only access inside class if we want to access outside class then we need to use setter() and getter() method();

14. Explain an object? Create an object of above class

ans:A Java object is a member (also called an instance) of a Java class.

Each object has an identity, a behavior and a state.

Cricket c = new Cricket();

15. Give real life examples of object.

ans : a car, a person, hard disk, pen, bank account

16. Explain a Constructor.

ans: A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

17. Define the various types of constructors?

ans: Default Constructor

Parameterized Constructor

Copy Constructor

Static Constructor

Private Constructor

18. Whether static method can use nonstatic members?

ans: In static method, the method can only access only static data members and static methods of another class or same class but cannot access non-static methods and variables.

19. Explain Destructor?

ans: A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete.

20. Explain an Inline function?

ans: An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate set of instructions in memory.

21. Explain a virtual function?

Ans=> A virtual function or virtual method in an OOP language is a function or method used to override the behaviour of the function in an inherited class with the same signature to achieve the polymorphism. When the programmers switch the technology from C++ to Java, they think about where is the virtual function in Java.

In C++, the virtual function is defined using the virtual keyword, but in Java, it is achieved using different techniques.

22. Explain a friend function?

Ans=> A friend function of a class is defined outside that class scope but it has the right to access all private and protected members of the class.

Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

23. Explain function overloading?

Ans=> If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.

Advantage of method overloading:

Method overloading increases the readability of the program.

Different ways to overload the method:

There are two ways to overload the method in java-

1. By changing number of arguments

2. By changing the data type

24. Explain a base class, sub class, super class?

Ans=> The class which inherits the properties of other is known as subclass (derived class, child class). The class whose properties are inherited is known as superclass (base class, parent class). In Java, the superclass, also known as the parent class, is the class from which a child class (or a subclass) inherits its

constructors, methods, and attributes.

For instance, in our above example, Bank Account was the superclass from which our subclass Savings Account inherited its values.

The super keyword is similar to this keyword. Following are the scenarios where the super keyword is used.

-It is used to differentiate the members of superclass from the members of subclass, if they have same names.

-It is used to invoke the superclass constructor from subclass.

25. Write in brief linking of base class, sub class and base object, sub object.

Ans=> Difference between referencing using superclass reference and referencing using subclass reference is use superclass referencing can holds object of subclass and could only access the methods which are defined/overridden by subclass while use subclass referencing cannot hold object of superclass and

could access the methods of both superclass and subclass.

Object relation of Superclass (parent) to Subclass (child) exists while child to parent object relation never exists. This means that reference of parent class could hold the child object while child reference could not hold the parent

object.

In case of overriding of non-static method the runtime object would evaluate that which method would be executed of subclass or of superclass.

While execution of static method depends on the type of reference that object holds. Other basic rule of inheritance is related to static and non-static method overriding that static method in java could not be overridden while non static method can be. However, subclass can define static method of same static method signature as superclass have hut that would not be consider as overriding while known as hiding of static method of superclass.

Q26 Explain an abstract class?

Ans=> An abstract class is a template definition of methods and variables of a class (category of object) that contains one or more abstracted methods. Abstract classes are used in all object-oriented programming (OOP) languages, including Java. Objects or classes may be abstracted, which means that they are summarised into characteristics that are relevant to the current program's operation.

Q27 Explain operator overloading?

Ans=> Operator overloading is a technique by which operators used in a programming language are implemented in user-defined types with customised logic that is based on the types of arguments passed.

Operator overloading facilitates the specification of user-defined implementation for operations wherein one or both operands are of user-defined class or structure type. This helps user-defined types to behave much like the fundamental primitive data types. Operator overloading is helpful in cases where the operators used for certain types provide semantics related to the domain context and syntactic support as found in the programming language. It is used for syntactical convenience, readability and maintainability. Java does not support operator overloading, except for string concatenation for which it overloads the + operator internally.

Q28 Define different types of arguments? (Call by value/Call by reference)

Ans=> Call by value method copies the value of an argument into the formal parameter of that function. Therefore, changes made to the parameter of the main function do not affect the argument.

In this parameter passing method, values of actual parameters are copied to function's formal parameters, and the parameters are stored in different memory locations. So, any changes made inside functions are not reflected in actual parameters of the caller.

Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument.

In this method, the memory allocation is the same as the actual parameters. All the operation in the function is performed on the value stored at the address of the actual parameter, and the modified value will be stored at the same address.

Q29 Explain the super keyword?

Ans=> The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Uses of super keyword:

1. super can be used to refer immediate parent class instance variable.

2. super can be used to invoke immediate parent class method.

3. super() can be used to invoke immediate parent class constructor.

Q.30 Explain method overriding?

Ans=> If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.

In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

◦ Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

◦ Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class

2. The method must have the same parameter as in the parent class.

3. There must be an IS-A relationship (inheritance).

31. Difference among overloading and overriding?

Ans : 1.Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its

definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.

2.Static methods can be overloaded which means a class can have more than one static method of same name. Static methods

cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.

3.The most basic difference is that overloading is being done in the same class while for overriding base and child classes

are required. Overriding is all about giving a specific implementation to the inherited method of parent class.

4.Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.

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

Ans : In static method, the method can only access only static data members and static methods of another class or same class but cannot access non-static methods and variables.

33. Explain a base class, sub class, super class?

Ans : Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).

Sub Class: The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class).

The subclass can add its own fields and methods in addition to the superclass fields and methods.

Base class:A base class is a class, in an object-oriented programming language, from which other classes are derived.

It facilitates the creation of other classes that can reuse the code implicitly inherited from the base class (except constructors and destructors)

34. Write in brief linking of base class, sub class and base object, sub object.

Ans : First approach (Referencing using Superclass reference): A reference variable of a superclass can be used to a refer any subclass object derived from that superclass. If the methods are present in SuperClass, but overridden by SubClass, it will be the overridden method that will be executed.

Second approach (Referencing using subclass reference) : A subclass reference can be used to refer its object.

1.Object relation of Superclass (parent) to Subclass (child) exists while child to parent object relation never exists.

This means that reference of parent class could hold the child object while child reference could not hold the parent object.

2.In case of overriding of non static method the runtime object would evaluate that which method would be executed of subclass or of superclass.

While execution of static method depends on the type of reference that object holds.

3.Other basic rule of inheritance is related to static and non static method overriding that static method in java could not be overridden

while non static method can be.However subclass can define static method of same

4.static method signature as superclass have hut that would not be consider as overriding while known as hiding of static method of superclass.

35. Explain an interface?

Ans : An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class.

A Java interface contains static constants and abstract methods. A class can implement multiple interfaces.

In Java, interfaces are declared using the interface keyword. All methods in the interface are implicitly public and abstract.

36. Explain exception handling?

Ans : 1.The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

2.there are three types: 1.Checked Exception

2.Unchecked Exception

3.Error

3.Exception Handling is a mechanism to handle runtime errors such as

ClassNotFoundException, IOException, SQLException, RemoteException, etc.

37. Explain the difference among structure and a class?

Ans : A class is a user-defined blueprint or prototype from which objects are created.

Basically, a class combines the fields and methods(member function which defines actions) into a single unit. A structure is a collection of variables of different data types under a single unit.

38. Explain the default access modifier in a class?

Ans : The access level of a default modifier is only within the package.

It cannot be accessed from outside the package.

If you do not specify any access level, it will be the default.

39. Explain a pure virtual function?

Ans : A pure virtual function or pure virtual method is a virtual function that is required

to be implemented by a derived class if the derived class is not abstract.

Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly.

40. Explain dynamic or run time polymorphism?

Ans : 1.Run-Time Polymorphism: Whenever an object is bound with the functionality at run time, this is known as runtime polymorphism.

2.The runtime polymorphism can be achieved by method overriding.

3.Java virtual machine determines the proper method to call at the runtime, not at the compile time.

Q.41) Do we require a parameter for constructors ?

Ans. No, we don’t require parameter for constructor.

Q.42) Explain static and dynamic Binding?

Ans. 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

Q.43) How Many instances can be created for an abstract class?

Ans. No you can't, instead you can create instance of all other classes extending that abstract class. Because it's abstract and an object is concrete. An abstract class is sort of like a template, or an empty/partially empty structure, you have to extend it and build on it before you can use it.

Q.44) Explain the default access specifier in a class definition?

Ans. It is a Private in a Class.When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class. Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package.

Q.45) Which OOPS Concept is used as reuse Mechanism?

Ans. Inheritance is a OOPs Concept That can be used as reuse Mechanism.

Q.46) Define the Benefits Of Object Oriented Programming?

Ans. 1. Code Reusability

2.Productivity

3.Data Redundancy

4.Code Flexibility

5.Solving problem

Q.47) Explain method Overloading?

Ans. 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 overrides the implementation in the superclass by providing the same method name, same parameter and same return type.

Q.48) Explain the Difference among early Binding and late Binding ?

1. The early binding happens at the compile-time and late binding happens at the run time.

2. In early binding, the method definition and the method call are linked during the compile time. This happens when all information needed to call a method is available at the compile time.

Unlike early binding, the method calls are not resolved until run time in late binding as we cannot determine all information needed for a method call at compile time. So the method definition and a method call are not bound until run time.

3. The normal method calls and overloaded method calls are examples of early binding, while reflection and method overriding (run time polymorphism) are examples of late binding.

4. The binding of private, static, and final methods happens at the compile-time as they cannot be overridden.

5. Since all information needed to call a method is available before run time, early binding results in faster execution of a program. While for later binding, a method call is not resolved until run time, resulting in somewhat slower execution of code.

6. The major advantage of late binding is its flexibility since a single method can handle different types of objects at run time. This significantly reduces the size of the codebase and makes the code more readable.

Q.49) Explain early Binding?

Ans. This is compile time polymorphism. Here it directly associates an address to the function call. For function overloading it is an example of early binding.

Q.50) Explain loose Coupling And tight Coupling

Loose Couplnig – When an object gets the object to be used from external sources, we call it loose coupling. In other words, the loose coupling means that the objects are independent. A loosely coupled code reduces maintenance and efforts. This was the disadvantage of tightly coupled code that was removed by the loosely coupled code.

Tight Coupling - It is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its own class. The situation where an object creates another object for its usage, is termed as Tight Coupling.

Q51.Give an example among tight coupling and loose coupling.

ANS=>TIGHT COUPLING: Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled. Example : If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. LOOSE COUPLING: In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. EXAMPLE: f you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling.

Q52. Write in brief abstract class.

ANS=>A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abs tract and non-abstract methods (method with the body). It needs to be extended and its method implemented. It cannot be instantiated. It can have constructors and static methods also. It can have final methods which will force the subclass not to change the body of the method.

Q 53.Define the Benefits of oops over pop?

ANS=> Some of the benefits of object oriented programming(OOPS) over process oriented programming (POP) are as follows: 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere. 3)OOPs provides ability to simulate real-world problems much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

Q54. Explain Generalization and Specialization?

ANS=> GENERALIZATION : Converting a subclass type into a superclass type is called ‘Generalization‘ because we are making the subclass to become more general and its scope is widening. This is also called widening or up casting. Widening is safe because the classes will become more general. SPECIALIZATION: Converting a super class type into a sub class type is called ‘Specialization‘. Here, we are coming down from more general form to a specific form and hence the scope is narrowed. Hence, this is called narrowing or down-casting.

Q55.Write in brief Association, Aggregation and Composition?

ANS=>ASSOCIATION: Association refers to the relationship between multiple objects. It refers to how objects are related to each other and how they are using each other's functionality. Composition and aggregation are two types of association. AGGREGATION: Aggregation is a weak association. An association is said to be aggregation if both Objects can exist independently. For example, a Team object and a Player object. The team contains multiple players but a player can exist without a team. COMPOSITION: The composition is the strong type of association. An association is said to composition if an Object owns another object and another object cannot exist without the owner object. Consider the case of Human having a heart. Here Human object contains the heart and heart cannot exist without Human.

Q56. Write in brief Object Composition vs. Inheritance.

Ans => Inheritance: a class may inherit - use by default - the fields and methods of its superclass. Inheritance is transitive, so a class may inherit from another class which inherits from another class, and so on, up to a base class (typically Object, possibly implicit/absent). Subclasses may override some methods and/or fields to alter the default behaviour. Composition: when a Field's type is a class, the field will hold a reference to another object, thus creating an association relationship between them. Without getting into the nuances of the difference between simple association, aggregation, and composition, let's intuitively define composition as when the class uses another object to provide some or all of its functionality.

Q57. Explain cohesion?

ANS=> In object-oriented design, cohesion refers all about how a single class is designed. Cohesion is the Object-Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose. The more focused a class is, the cohesiveness of that class is more. The advantages of high cohesion is that such classes are much easier to maintain (and less frequently changed) than classes with low cohesion. Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.

Q58. Explain "black-box-reuse" and "white-box-reuse"?

ANS=> Black box reuse is using a class/function/code unmodified in a different project. Black-Box reuse means that you use component without knowing it's internals. All you have is a component interface. White box reuse is taking a class/function/code from one project and modifying it to suit the needs of another project. White-box reuse means that you know how component is implemented. Usually White-box reuse means class inheritance.

Q59. Explain “this”

ANS=> The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameter s with the same name (because a class attribute is shadowed by a method or constructor parameter). If you omit the keyword in the example above, the output would be "0" instead of "5". *this can also be used to: current class constructor Invoke current class method Return the current class object Pass an argument in the method call Pass an argument in the constructor call.

Q60. Write in brief static member and member function.

ANS=>Static Data Member (Class variable): Static Data member has the following properties: It is initialized by zero when first object of class is created. Only one copy of static data member is created for the entire class and all object share the same copy. Its scope is within class but its lifetime is entire program. They are used to store the value that are common for all the objects. Static variable can be declared using the following syntax: static data-type variable-name; The above syntax only declare the static variable. We can initialize static variable by using following syntax: data-type class-name :: variable-name = initial-value ; Along with the definition we can also initialize the static variable. If static data members (variables) are declared under the public section than it can be accessed from outside the class and if it is declared in private section than it can only be accessed within the class itself. Static variables are also known as class variables because they are not the variables of any particular class. Let’s consider the program given below to understand the static variable in which, we are creating static variable ‘counter’ to count the number of objects created for the class ‘MyClass’. Static Member Function: Member variables are known as instance variables in java. Instance variables are declared in a class, but outside a method, constructor or any block. When space is allocated for an object in the heap, a slot for each instance variable value is created. Static member function has following properties: A static function can be access only by other static data member (variables) and function declared in the class. A static function is called using class name instead of object name. Consider the following program (Given in above section). Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class. Instance variables can be declared in a class level before or after use. Access modifiers can be given for instance variables. The instance variables are visible for all methods, constructors, and block in the class. Normally, it is recommended t o make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers. Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor. Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.

Q.61.How will you relate unrelated classes or how will you achieve polymorphism without using base class?

Ans. Yes, polymorphism without using base class. It can be done through method overriding (Inheritance)and Interfaces implementation.

Q.62.Explain Diamond problem?

Ans. The diamond problem in java is related to multiple inheritance. Java does not support the multiple inheritance because of the diamond problem.

Q.63.Explain the solution for diamond problem?

Ans. The solution to the diamond problem is default methods and interfaces. We can achieve multiple inheritance by using these two things.

The advantage of interfaces is that it can have the same default methods with the same name and signature in two different interfaces.

It allows us to implement these two interfaces, from a class. We must override the default methods explicitly with its interface name.

Q.64.Explain the need of abstract class?

Ans. Abstraction is a process of hiding the implementation details and showing only functionality to the user. It shows only essential things to the user and hides the internal details. Abstraction lets you focus on what the object does instead of how it does it.

Q.65.Why can’t we instantiate abstract class?

Ans. An abstract class is a class which doesn’t have an implementation for one or more methods. It means that the jvm doesn’t have any direction of what to do in case if someone calls the method which is abstract. So, if we are able to create an object for the abstract class and call any abstract method of it the jvm will not be able to decide what to do and hence it may be crashed. So to avoid this situation we are restricted to instantiate an abstract class. If we need an object for that abstract class create a concrete subclass and create an object for it and use it.

Q.66.Can abstract class have constructors?

Ans. Yes it can have a constructor and it is defined and behaves just like any other class's constructor. Except that abstract class can't be directly instantiated, only extended, so its use is therefore always from a subclass's constructor.

Q.67.How many instances can be created for an abstract class?

Ans. We cannot create instance of an abstract class.

Q.68.Which keyword can be used for overloading?

Ans. The super keyword can be used for overloading.

We create an object of child class as it can inherit the parent class methods.

In the child class method, we call the method available in its parent class by using super().

Q.69.Explain the default access specifiers in a class definition

Ans. When no access modifier is specified for a class, method, or data member, it is said to be having the default access modifier by default. The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

Q.70. Define all the operators that can be overloaded?

Ans. Java doesn't support operator overloading. Java doesn't provide freedom to programmers, to overload the standard arithmetic operators e.g. +, -, * and / etc. If you have worked previously in C++, then you know that Java has left a lot of features supported in C++ e.g. Java doesn't support multiple inheritances, no pointers in Java, and no pass-by-reference in Java. Rarely is this question asked in Java interviews, to check how a programmer thinks about certain features, which are not supported in Java.

Q71. Explain the difference among structure and a class?

Ans => A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods (member function which defines actions) into a single unit. A structure is a collection of variables of different data types under a single unit. It is almost similar to a class because both are user-defined data types and both hold a bunch of different data types.

Q72. Explain the default access modifier in a class?

ANS=> If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public. Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same package.

Q73. Can you list out the different types of constructors?

ANS=> A constructor is a special type of function with no return type. Name of constructor should be same as the name of the class. We define a method inside the class and constructor is also defined inside a class. A constructor is called automatically when we create an object of a cl ass. We can't call a constructor explicitly. Constructor Types Default Constructor Parameterized Constructor Copy Constructor Static Constructor Private Constructor

Q74. Explain a friend function?

ANS=> A friend function is a function that is specified outside a class but has the ability to access the class members' protected and private data. A friend can be a member's function, function template, or function, or a class or class template, in which case the entire class and all of its member s are friends. A friend function in C++ is a function that is preceded by the keyword "friend". When the function is declared as a friend, then it can access the private and protected data members of the class. A friend function is declared inside the class with a friend keyword preceding as shown below.

Q75. Explain a ternary operator?

ANS=> The ternary operator is an operator that exists in some programming languages, which takes three operands rat her than the typical one or two that most operators use. It provides a way to shorten a simple if else block. With this type of comparison, you can shorten the code using the ternary operator. operands: a condition followed by a question mark ( ? ), then an expression to execute if the condition is truthy followed by a colon ( : ), and finally the expression to execute if the condition is false.

Q76. Do We Require Parameter For Constructors?

ANS=>No-argument constructor: A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler creates default constructor (with no arguments) for the cl ass. Default constructor provides the default values to the object like 0, null, etc. It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Q77. Explain Sealed Modifiers?

ANS=>When applied to a class, the sealed modifier prevents other classes from inheriting from it. In the following example, class B inherits from class A , but no class can inherit from class B.

Q78. Explain The Difference Among New And Override?

ANS=>virtual keyword must be defined to override the method. The method using override keyword that regardless of reference type (reference of base class or derived class) if it is instantiated with base class, the method of base class runs. Otherwise, the method of derived class runs. new: if the keyword is used by a method, unlike override keyword, the reference type is important. If it is instantiated with derived class and the reference type is base class, the method of base class runs. If it is instantiated with derived class and the reference type is derived class, the method of derived class runs. Namely, it is contrast of override keyword. Enpassant, if you forget or omit to add new keyword to the method, the compiler behaves by default as new keyword is used.

Q79. How Can We Call the Base Method Without Creating an Instance?

ANS=>Its possible If it’s a static method. It’s possible by inheriting from that class also. It’s possible from derived classes using base keyword.

Q80. Define The Various Types Of Constructors?

ANS=>There are two types of constructors in Java: 1.no-arg constructor/Default constructor: A constructor is called "Default Constructor" when it doesn't have any parameter. 2.parameterized constructor: A constructor which has a specific number of parameters is called a parameterized constructor.

Q.81. Define Manipulators.

Ans. Manipulators are helping functions that can modify the input/output stream.

It does not mean that we change the value of a variable, it only modifies the I/O stream

using insertion (<<) and extraction (>>) operators.

For example, if we want to print the hexadecimal value of 100 then we can print it as:

cout<<setbase(16)<<1

Q82. Can you give some examples of tokens?

Ans. The Java compiler breaks the line of code into text (words) is called Java tokens.

These are the smallest element of the Java program. The Java compiler identified these

words as tokens. These tokens are separated by the delimiters. It is useful for compilers

to detect errors. Remember that the delimiters are not part of the Java tokens.

There are nine separators in Java, and the last six (3 pairs of braces) are separators.

Q.83. Explain structured programming and its disadvantage?

Ans. Structured Programming Approach, as the word suggests, can be defined as a programming

approach in which the program is made as a single structure. It means that the code will execute

the instruction by instruction one after the other. It doesn't support the possibility of jumping

from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the

instructions in this approach will be executed in a serial and structured manner.

Disadvantages of Structured Programming Approach:

Since it is Machine-Independent, So it takes time to convert into machine code.

The converted machine code is not the same as for assembly language.

The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.

Usually the development in this approach takes longer time as it is language-dependent.

Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Q.84. Explain the advantage of C++ being a block-structured language?

Ans. Advantages of C++ being a blocked-structured language are as follows:-

C++ is a highly portable language and is often the language of selection for multi-device,

multi-platform app development.

C++ is an object-oriented programming language and includes concepts like classes, inheritance, polymorphism, data abstraction, and encapsulation which allow code reusability and makes programs very maintainable. It is useful for the low-level programming language and very efficient for general purpose.

C++ gives the user complete control over memory management. This can be seen both as an advantage and a disadvantage as this increases the responsibility of the user to manage memory rather than it being managed by the Garbage collector.

Q.85. Can Struct be inherited?

Ans. Yes, Struct can be inherited.

A structure's inheritance is the same as a class except the following differences:

When deriving a struct from a class/struct, the default access-specifier for a base class/struct is public. And when deriving a class, the default access specifier is private.

Q.86. When to use interface over abstract class.

Ans. An interface can be used to define a contract behaviour and it can also act as a contract between two systems to interact while an abstract class is mainly used to define default behaviour for subclasses, it means that all child classes should have performed the same functionality. interfaces are a good choice when we think that the API will not change for a while.

Interfaces are also good when we want to have something similar to multiple inheritances since we can implement multiple interfaces.

Q.87. Explain a private constructor? Where will you use it?

Ans. Private Constructor is a special instance constructor present in C# language.

Basically, private constructors are used in class that contains only static members. The private constructor is always declared by using a private keyword.

Important points:

It is the implementation of a singleton class pattern.

Use private constructor when class have only static members.

Using private constructor, prevents the creation of the instances of that class.

If a class contains only private constructor without parameter, then it prevents the automatic generation of default constructor.

If a class contains only private constructors and does not contain public constructor, then other classes are not allowed to create instances of that class except nested class.

Q.88. Can you override private virtual methods?

Ans. Ideally No. But, using the tricky code, a subclass can override a private method as well.

Q.89. Can you allow class to be inherited, but prevent from being over-ridden?

Ans. Yes, just leave the class public and make the method sealed.

Thus we can allow class to be inherited, but also prevent from being over-ridden.

Q.90. Why can’t you specify accessibility modifiers for methods inside interface?

Ans. Interface methods are contract with the outside world which specifies that class implementing this interface does a certain set of things.

Interface members are always public because the purpose of an interface is to enable other types to access a class or struct.

Interfaces can have access specifiers like protected or internal etc. Thus limiting 'the outside world' to a subset of 'the whole outside world'.

Q.91.Can static members use non static members? Give reason.

Ans ->The answer here is no. What is the reason? Well, because non-static member variables of a class always belong to an object – meaning that every object has it's own personal copy of non-static member variables (also known as instance variables). And, static functions have no object to work with since they belong to the class as a whole.

Remember that you can call a static function without an object – and for that exact reason a static function cannot call instance variables. For example, the following code will not work:

Ex.

```
class Stocks
{

  int number;

  static void picker( )
  {
    /* this is not allowed because picker is a static
       function and cannot access an instance variable
    */
    number = 6;
    ...
  }
}
```

Q.92.Define different ways a method can be overloaded?

Ans -> Java can distinguish the methods with different method signatures. i.e. the methods can have the same name but with different parameters list (i.e. the number of the parameters, the order of the parameters, and data types of the parameters) within the same class.

1.The number of parameters in two methods.

2.The data types of the parameters of methods.

3.The Order of the parameters of methods.

Overloaded methods are differentiated based on the number and type of the parameters passed as an argument to the methods.

You can not define more than one method with the same name, Order and the type of the arguments. It would be a compiler error.

The compiler does not consider the return type while differentiating the overloaded method. But you cannot declare two methods with the same signature and different return type. It will throw a compile-time error.

If both methods have the same parameter types, but different return type, then it is not possible.

Q.93.Can we have an abstract class without having any abstract method?

Ans -> A class which contains 0 or more abstract methods is known as abstract class. If it contains at least one abstract method, it must be declared abstract.

And yes, you can declare abstract class without defining an abstract method in it. Once you declare a class abstract, it indicates that the class is incomplete and, you cannot instantiate it.

Hence, if you want to prevent instantiation of a class directly you can declare it abstract.

If you want to use the concrete method in an abstract class you need to inherit the class, provide implementation to the abstract methods (if any) and then, you using the subclass object you can invoke the required methods.

Example:

In the following Java example, the abstract class MyClass contains a concrete method with name display. From another class (AbstractClassExample) we are inheriting the class MyClass and invoking the its concrete method display using the subclass object.

```
abstract class MyClass {
  public void display() {
    System.out.println("This is a method of abstract class");
  }
}
public class AbstractClassExample extends MyClass{
  public static void main(String args[]) {
    new AbstractClassExample().display();
  }
}
```

Q.94.Explain the default access modifier of a class?

Ans -> If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But it is more restrictive than protected, and public.

Example:

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package,

since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java

package pack;

class A{

  void msg(){System.out.println("Hello");}

}


//save by B.java

package mypack;

import pack.*;

class B{

  public static void main(String args[]){

   A obj = new A();//Compile Time Error

   obj.msg();//Compile Time Error

  }

}
```

Q.95.Can function overriding be explained in same class?

Ans -> No, When we have two classes where one extends another and if, these two classes have the same method including parameters and return type (say, sample) the method in the subclass overrides the method in the superclass. i.e. Since it is an inheritance. If we instantiate the subclass a copy of superclass's members is created in the subclass object and, thus both methods are available to the object of the subclass. But if you call the method (sample), the sampling method of the subclass will be executed overriding the super class's method.

Both methods should be in two different classes and, these classes must be in an inheritance relation. Both methods must have the same name, same parameters and, same return type else they both will be treated as different methods.

The method in the child class must not have higher access restrictions than the one in the superclass. If you try to do so it raises a compile-time exception.

If the super-class method throws certain exceptions, the method in the sub-class should throw the same exception or its subtype (can leave without throwing any exception).

Q.96.Does function overloading depends on Return Type?

Ans -> No, It does not depend on Return Type. Because if return type is different and function name as well as parameter is also same. Then it will give compile time error.

Q.97.Define different ways to declare an array?

Ans -> A Java array variable can also be declared like other variables with [] after the data type.

The variables in the array are ordered and each have an index beginning from 0.

Java array can be also be used as a static field, a local variable or a method parameter.

Q.98.Can abstract class have a constructor?

Ans -> Yes, when we define a class to be an Abstract Class it cannot be instantiated but that does not mean an Abstract class cannot have a constructor. Each abstract class must have a concrete subclass which will implement the abstract methods of that abstract class.

When we create an object of any subclass all the constructors in the corresponding inheritance tree are invoked in the top to bottom approach. The same case applies to abstract classes. Though we cannot create an object of an abstract class, when we create an object of a class which is concrete and subclass of the abstract class, the constructor of the abstract class is automatically invoked. Hence, we can have a constructor in abstract classes.

Note: A non-abstract class cannot have abstract methods but an abstract class can have a non-abstract method.

Reason is similar to that of constructors, difference being instead of getting invoked automatically we can call super(). Also, there is nothing like an abstract constructor as it makes no sense at all.

Q.99.Define rules of Function overloading and function overriding?

Ans -> 1) Function Overloading happens in the same class when we declare same functions with different arguments in the same class. Function Overriding is happening in the child class when child class overrides parent class function.

2) In function overloading function signature should be different for all the overloaded functions. In function overriding the signature of both the functions (overriding function and overridden function) should be same.

3) Overloading happens at the compile time that's why it is also known as compile time polymorphism while overriding happens at run time which is why it is known as run time polymorphism.

4) In function overloading we can have any number of overloaded functions. In function overriding we can have only one overriding function in the child class.

Q.100.Explain the concept of Pure Virtual Functions?

Ans -> A pure virtual function is a function which has no definition in the base class. Its definition lies only in the derived class i.e. it is compulsory for the derived class to provide definition of a pure virtual function.

Since there is no definition in the base class, these functions can be equated to zero.

Pure virtual function is the function in the base class with no body. Since no body, you have to add the notation =0 for declaration of the pure virtual function in the base class.

- The base class with pure virtual function can't be instantiated since there is no definition of the function in the base class.

- It is necessary for the derived class to override pure virtual function.

- This type of class with one or more pure virtual function is called abstract class which can't be instantiated, it can only be inherited.

```
class shape
{
  public: virtual void draw() = 0;
};
```