Interview Questions

08 November 2022 14:24

1.Explain OOPS?
Answer :-

Object oriented programming language is computer programming mode that organizes software design around data or object rather than function and logic.

An object can be define as a data field that has unique attributes and be havier.

2.Explain an abstraction? Real life example?
Answer :-

Abstraction is used to hide complexity and showing only necessary detail.Abstraction can be achieved with either abstract classes or interfaces .

Example :-

Car is the best example for abstraction we know only how to drive a car but we don't know internal functionality of a car.

3.Explain encapsulation? Real life example.
Answer :-

Binding of data and function together inside class is called encapsulation.the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.

Example :-

As a car driver we know how to start a car but we don't know what is functionality to start a car.

4.Explain the relationship among abstraction and encapsulation?
Answer :-
Abstraction is the method of hiding the unwanted information This important concept in object-oriented programming will reduce the complexity of the code and increases the readability.
whereas encapsulation is a method to hide the data is a single entity or unit along with a method to protect information from outside.Encapsulation minimizes your code's part revealed to the user. The user can be anyone who uses your published code or perhaps your code's remaining part.

5.Explain polymorphism?
Answer :-

One interface multiple method or one interface to be used for a general class of action. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

Example :-

The best example of polymorphism is human behavior. One person can have different behavior. For example, a person acts as an employee in the office, a customer in the shopping mall, a passenger in bus/train, a student in school, and a son at home.

6.Explain Inheritance?
Answer :-

One object acquires the properties of another object.

There is various type of inheritance.
Single Inheritance:In Single Inheritance one class extends another class.
Multiple Inheritance:Multiple Inheritance is one of the inheritance in Java types where one class extending more than one class. Java does not support multiple inheritance.
Multilevel Inheritance:In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.
Hierarchical Inheritance:In Hierarchical Inheritance, one class is inherited by many sub classes.

7.How composition is better than inheritance?
Answer :-

Benefit of composition over inheritance is testing scope. Unittesting is very easy in composition because we know what all method we are using from another class. we can mock it up for testing whereas in heritance we depend heavily on superclass and don’t know what all method of super class.

8.Which OOPS concept is used as a reuse mechanism?
Answer :-
Inheritance is the feature that provides a reuse mechanism. While abstraction, encapsulation, and dynamic binding have different functionalities in the OOP paradigm.

Which OOPS concept exposes only the necessary information to the calling functions?
Answer :-
Data hiding is the concept of oops which means exposing necessary information to Clint. Data hiding is a technique used in object-oriented programming. It means hiding the internal details. Data hiding makes sure that the internal details are restricted to class members.Data hiding makes sure that the internal details are restricted to class members. Data integrity is maintained in data hiding. Data hiding reduces the complexities and increases the robustness. Another main advantage is that it reduces the interdependencies between two software.

10.Explain a class? Create a class.
Answer :-
A class is a blueprint from which instance of class is created.
Create class :-
Tap classroom.
Tap add + create class.
Enter the class name.
To enter a short description, grade, or class time tap section and enter the detail.
To enter the location for the class tap room and enter the detail.
To add a subject tap subject and enter a name

11. Using above created class, Write in brief abstraction and encapsulation
-
Definition of Abstraction
Abstraction is an OOP concept that focuses only on relevant data of an object. It hides the background details and emphasizes the essential data points for reducing the complexity and increase efficiency. It generally retains only information which is most relevant for that specific process. Abstraction method mainly focusses on the idea instead of actual functioning.
Definition of Encapsulation
Encapsulation is a method of making a complex system easier to handle for end users. The user need not worry about internal details and complexities of the system.
Encapsulation is a process of wrapping the data and the code, that operate on the data into a single entity. You can assume it as a protective wrapper that stops random access of code defined outside that wrapper.

12. Explain difference among class and object?
-
- A class is a template for creating objects in a program, whereas the object is an instance of a class.
- A class is a logical entity, while an object is a physical entity.
- A class does not allocate memory space; on the other hand, an object allocates memory space.
- You can declare a class only once, but you can create more than one object using a class.
- Classes can’t be manipulated, while objects can be manipulated.
- Classes don’t have any values, whereas objects have their own values.
- You can create a class using “class” keyword, while hand you can create an object using “new” keyword in Java.

13. Define access modifiers?
-
Access modifiers (or access specifiers) are **keywords in object-oriented languages that set the accessibility of classes, methods, and other members**.
Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components.

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

An object is a **real-world entity**. An object is a runtime entity. The object is an entity which has state and behavior. The object is an instance of a class. A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

15. Give real life examples of object.

-

An entity with some state and behaviour is referred to as an object. Objects can be tangible or intangible. Object Examples : **pen, car, bike, table, chair, mobile**, etc. The characteristics an object defines include state, behavior, and identity.

16. Explain a Constructor.

-

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
It is a special type of method which is used to initialize the object.
Every time an object is created using the new() keyword, at least one constructor is called.
It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

a. Constructor name must be the same as its class name
b. A Constructor must have no explicit return type
c. A Java constructor cannot be abstract, static, final, and synchronized

17. Define the various types of constructors?

-

There are two types of constructors in Java:
Default constructor (no-arg constructor)
Parameterized constructor

Java Default Constructor
A constructor is called "Default Constructor" when it doesn't have any parameter.

Java Parameterized Constructor
A constructor which has a specific number of parameters is called a parameterized constructor.
Why use the parameterized constructor?
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

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

-

Restrictions in Static Methods:
Non-static data members or non-static methods cannot be used by static methods, and static methods cannot call non-static methods directly.
In a static environment, this and super aren't allowed to be used.

19. Explain Destructor?

-

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed. Destructor is also a special member function like constructor

20. Explain an Inline function?

-

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. This eliminates call-linkage overhead and can expose significant optimization opportunity

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.
By default, all the instance methods in Java are considered as the Virtual function except final,

static, and private methods as these methods can be used to achieve polymorphism.

The virtual keyword is not used in Java to define the virtual function; instead, the virtual functions and methods are achieved using the following techniques:

We can override the virtual function with the inheriting class function using the same function name. Generally, the virtual function is defined in the parent class and override it in the inherited class.
The virtual function is supposed to be defined in the derived class. We can call it by referring to the derived class's object using the reference or pointer of the base class.
A virtual function should have the same name and parameters in the base and derived class.
For the virtual function, an IS-A relationship is necessary, which is used to define the class hierarchy in inheritance.
The Virtual function cannot be private, as the private functions cannot be overridden.
A virtual function or method also cannot be final, as the final methods also cannot be overridden.
Static functions are also cannot be overridden; so, a virtual function should not be static.
By default, Every non-static method in Java is a virtual function.
The virtual functions can be used to achieve oops concepts like runtime polymorphism.

22. Explain a friend function?
Ans:

A C++ friend functions are special functions which can access the private members of a class. They are considered to be a loophole in the Object Oriented Programming concepts, but logical use of them can make them useful in certain cases.
For instance: when it is not possible to implement some function, without making private members accessible in them. This situation arises mostly in case of operator overloading.
Or another example when you want to compare two private data members of two different classes in that case you need a common function which can make use of both the private variables of different class. In that case you create a normal function and make friend in both the classes, as to provide access of theirs private variables.

23. Explain function overloading?
Ans:

Function Overloading in Java occurs when there are functions having the same name but have different numbers of parameters passed to it, which can be different in data like int, double, float and used to return different values are computed inside the respective overloaded method.

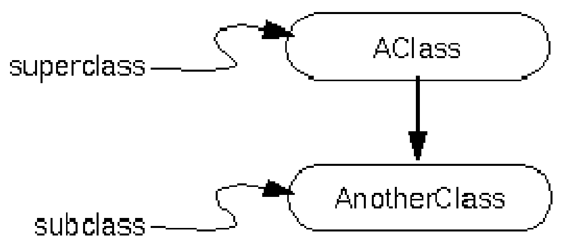
```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}
```

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

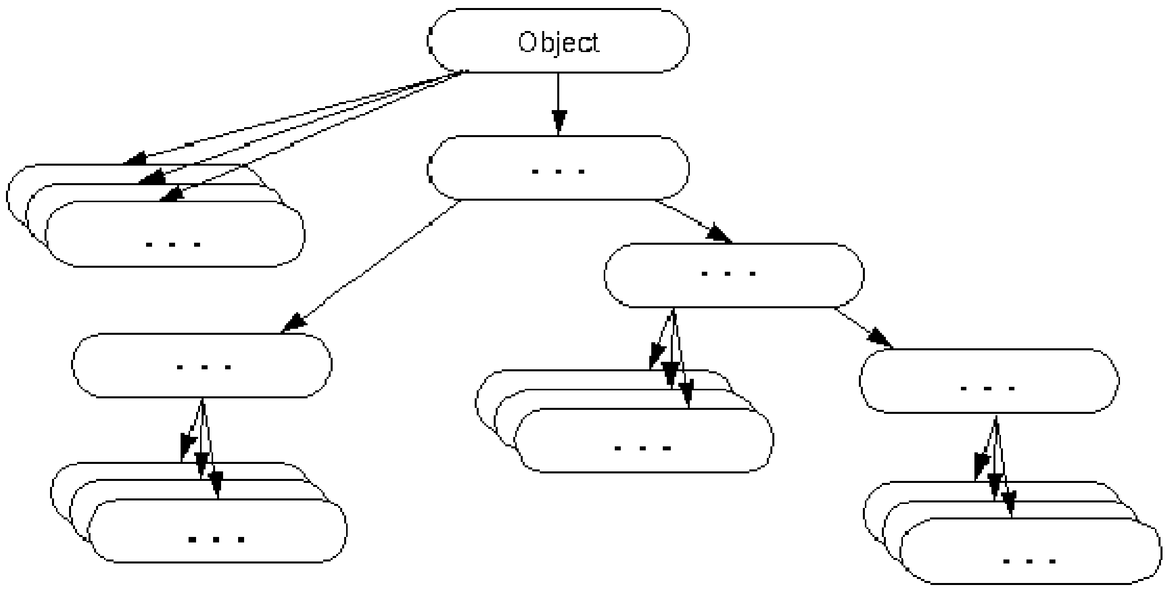
Ans:

Subclasses, Superclasses, and Inheritance

In Java, as in other object-oriented programming languages, classes can be derived from other classes. The derived class (the class that is derived from another class) is called a subclass. The class from which its derived is called the superclass.



In fact, in Java, all classes must be derived from some class. Which leads to the question "Where does it all begin?" The top-most class, the class from which all other classes are derived, is the Object class defined in java.lang. Object is the root of a hierarchy of classes.



The subclass inherits state and behavior in the form of variables and methods from its superclass. The subclass can just use the items inherited from its superclass as is, or the subclass can modify or override it. So, as you drop down in the hierarchy, the classes become more and more specialized:

Definition: A subclass is a class that derives from another class. A subclass inherits state and behavior from all of its ancestors. The term superclass refers to a class's direct ancestor as well as all of its ascendant classes.

Now would be a good time to review the discussion in What Is Inheritance?

.
Creating Subclasses
To create a subclass of another class use the extends clause in your class declaration. (The Class Declaration explains all of the components of a class declaration in detail.) As a subclass, your class inherits member variables and methods from its superclass. Your class can choose to hide variables or override methods inherited from its superclass.
Writing Final Classes and Methods
Sometimes, for security or design reasons, you want to prevent your class from being subclassed. Or, you may just wish to prevent certain methods within your class from being overriden. In Java, you can achieve either of these goals by marking the class or the method as final.
Writing Abstract Classes and Methods
On the other hand, some classes are written for the sole purpose of being subclassed (and are not intended to ever be instantiated). These classes are called abstract classes and often contain abstract methods.
The Object Class
All objects in the Java environment inherit either directly or indirectly from the Object class. This section talks about the interesting methods in Object--methods that you may wish to invoke or override.

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

REFER Que 24

26. Explain an abstract class?
Ans
An abstract class is a template definition of methods and variables of a class (category of objects) that contains one or more abstracted methods.
Points to Remember
An abstract class must be declared with an abstract keyword.
It can have abstract and non-abstract methods.
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.

27. Explain operator overloading?
Ans:

In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big Integer, etc.
Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
Example:

```
 int a;
float b,sum;
sum=a+b;
```

Here, variables "a" and "b" are of types "int" and "float", which are built-in data types. Hence the addition operator '+' can easily add the contents of "a" and "b". This is because the addition operator "+" is predefined to add variables of built-in data type only.

Now, consider another example

```
class A
{
};
int main()
{
    A   a1,a2,a3;
    a3= a1 + a2;
    return 0;
}
```

In this example, we have 3 variables "a1", "a2" and "a3" of type "class A". Here we are trying to add two objects "a1" and "a2", which are of user-defined type i.e. of type "class A" using the "+" operator. This is not allowed, because the addition operator "+" is predefined to operate only on built-in data types. But here, "class A" is a user-defined type, so the compiler generates an error. This is where the concept of "Operator overloading" comes in.
Now, if the user wants to make the operator "+" to add two class objects, the user has to redefine the meaning of the "+" operator such that it adds two class objects. This is done by using the concept "Operator overloading". So the main idea behind "Operator overloading" is to use c++ operators with class variables or class objects. Redefining the meaning of operators really does not change their original meaning; instead, they have been given additional meaning along with their existing ones.

```
#include<iostream>
usingnamespacestd;

classComplex {
private:
    intreal, imag;
public:
    Complex(intr = 0, inti = 0) {real = r;   imag = i;}

    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex const&obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        returnres;
    }
    voidprint() { cout<< real <<" + i"<<imag<<'\n'; }
};

intmain()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2;
    c3.print();
}
```

Can we overload all operators?
Almost all operators can be overloaded except a few. Following is the list of operators that cannot be overloaded.

Operators that can be overloaded

- Binary Arithmetic -> +, -, *, /, %
- Unary Arithmetic -> +, -, ++, —
- Assignment -> =, +=,*=, /=,-=, %=
- Bit- wise -> & , | , << , >> , ~ , ^
- De-referencing -> (->)
- Dynamic memory allocation and De-allocation -> New, delete
- Subscript -> [ ]
- Function call -> ()
- Logical -> &, | |, !
- Relational -> >, < , = =, <=, >=

there are some operators that cannot be overloaded. They are

Scope resolution operator	: :
Member selection operator	
Member selection through	*

Pointer to member variable

Conditional operator	? :
Sizeof operator	sizeof()

Why can't the above-stated operators be overloaded?

1. sizeof – This returns the size of the object or datatype entered as the operand. This is evaluated by the compiler and cannot be evaluated during runtime. The proper incrementing of a pointer in an array of objects relies on the sizeof operator implicitly. Altering its meaning using overloading would cause a fundamental part of the language to collapse.
2. typeid: This provides a CPP program with the ability to recover the actual derived type of the object referred to by a pointer or reference. For this operator, the whole point is to uniquely identify a type. If we want to make a user-defined type 'look' like another type, polymorphism can be used but the meaning of the typeid operator must remain unaltered, or else serious issues could arise.
3. Scope resolution (::): This helps identify and specify the context to which an identifier refers by specifying a namespace. It is completely evaluated at runtime and works on names rather than values. The operands of scope resolution are note expressions with data types and CPP has no syntax for capturing them if it were overloaded. So it is syntactically impossible to overload this operator.
4. Class member access operators (.(dot), .* (pointer to member operator)): The importance and implicit use of class member access operators can be

28. Define different types of arguments? (Call by value/Call by reference)
Ans:

What is Call by Value?
In this particular parameter passing method, the values of the actual parameters copy into the function's formal parameters. It stores both types of parameters in different memory locations. Thus, if one makes any changes inside the function- it does not show on the caller's actual parameters.
This method passes a copy of an actual argument to the formal argument of any called function. In the case of a Call by Value method, any changes or alteration made to the formal arguments in a called function does not affect the overall values of an actual argument. Thus, all the actual arguments stay safe, and no accidental modification occurs to them.
What is Call by Reference?
In this case, both the formal and actual parameters refer to a similar location. It means that if one makes any changes inside the function, it gets reflected in the caller's actual parameters.
This method passes the address or location of the actual arguments to the formal arguments of any called function. It means that by accessing the actual argument's addresses, one can easily alter them from within the called function. Thus, in Call by Reference, it is possible to make alterations to the actual arguments. Thus, the code needs to handle the arguments very carefully. Or else, there might be unexpected results and accidental errors.

Difference Between Call by Value and Call by Reference

Parameter	Call By Value	Call By Reference
Convention of Naming	In this case, the parameter's value passes for invoking the function. In the case of calling a function, we pass the values of variables directly to a function. Thus, it has its name as Call by Value.	In this case, the parameter's reference passes for the function invocation. Whenever calling a function, rather than passing the variables' values, we pass its address instead (location of variables) to the

		function. Thus, it has its name as Call by Reference.
Effects of Changes	It copies the value of a passed parameter into the function's argument. Thus, any changes that occur in any argument inside the function have no reflection on the passed parameter.	Both the passed parameter and the argument refer to a similar location. Thus, any changes occurring in any argument inside the function also reflects in the passed parameter.
Type of Passing	The method of Call by Value passes a copy of the variable. Here, the values of all the variables copy into their corresponding dummy variables, also called functions.	The method of Call by Reference passes the variable itself. Here, it copies the address of the actual variables in the calling function into the dummy variables called functions.
Memory Location (Referred)	The memory location referred to by the actual arguments and passed parameters of a function are different. Meaning, it creates the formal and actual arguments in different memory locations.	The memory location referred to by the actual arguments and passed parameters of a function are the same. Meaning, it creates the formal and actual arguments in the very same memory location.
Language Supported	Languages like C++, C#. PHP, Visual Basic NET, etc., provide support to the Call by Value and use it as their default method.	Only the JAVA language supports the Call by Reference method in programming.
Value Modificati on	In the Call by Value method, there is no modification in the original value.	In the Call by Reference method, there is a modification in the original value.
Internal Implement ation	In the case of Call by Value, when we pass the value of the parameter during the calling of the function, it copies them to the function's actual local argument.	In the case of Call by Reference, when we pass the parameter's location reference/address, it copies and assigns them to the function's local argument. Thus, both the actual argument and passed parameters refer to one similar location.
Method of Passing	The values of the variables in a Call by Value method pass using a straightforward method or a Simple technique.	Defining the pointer variables in a Call by Reference method is a prerequisite for storing the address of all the variables.
Manipulati on of Variables	Using this method, any changes occurring to the dummy variables in any called function have no effect on the actual variable's values in the calling function. Thus, you cannot alter or manipulate the actual variables' values using the function calls.	Using this method, one can directly use the addresses to access the actual variables. Thus, any user can easily alter or manipulate the variables' values through the function calls.

29. Explain the super keyword?

Ans:

super is used to refer immediate parent class instance variable. We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

30. Explain method overriding?

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

31. Difference among overloading and overriding?

Method overloading	Method overriding
Method overloading is a compile-time polymorphism.	Method overridingis a run-time polymorphism. It is used to grant the specific implementation of the

It helps to increase the readability of the program	method which is already provided by its parent class or superclass.
	Dynamic binding is being used for overriding methods.
Static binding is being used for overloaded methods.	It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime.
Poor Performance due to compile time polymorphism.	Argument list should be same in method overriding.
	Private and final methods can’t be overridden
Argument list should be different while doing method overloading.	
Private and final methods can be overloaded.	

Method Overloading :

In Method Overloading, Methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.
Method Overloading is to “add” or “extend” more to the method’s behavior.
It is a compile-time polymorphism.
The methods must have a different signature.
It may or may not need inheritance in Method Overloading.
Let’s take a look at the example below to understand it better.

```
class Adder {
Static int add(int a, int b)
{
return a+b;
}
Static double add( double a, double b)
{
return a+b;
}
public static void main(String args[])
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Method Overriding:
In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.
Method Overriding is to “Change” existing behavior of the method.
It is a run time polymorphism.
The methods must have the same signature.
It always requires inheritance in Method Overriding.
Let’s take a look at the example below to understand it better.

```
class Car {
void run(){
System.out.println(“car is running”);
}
Class Audi extends Car{
void run()
{
System.out.prinltn("Audi is running safely with 100km");
}
public static void main( String args[])
{
Car b=new Audi();
b.run();
}
}
```

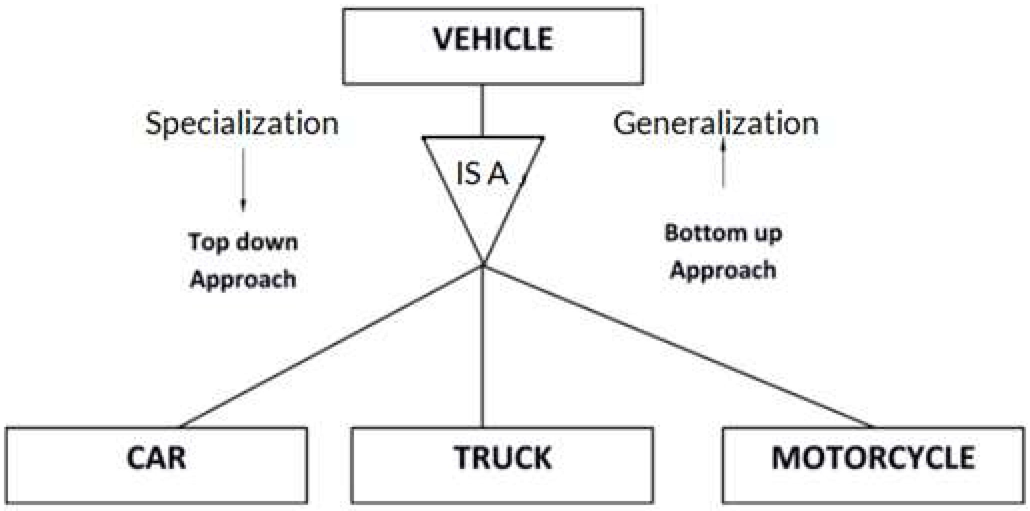
32. Whether static method can use non-static members?

A static method is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class. Every method in java defaults to a non-static method without static keyword preceding it. Non-static methods can access any static method and static variable, without creating an instance of the object. A static method can only access static data members and static methods of another class or same class but cannot access non-static methodsand variables. Also, a static method can rewrite the values of any static data member.A non-static method can access static data members and static methods as well as non-static members and methods of another class or same class, also can change the values of any static data member.

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

Subclasses /Child class=

A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contains attributes of its own. An example is:



Car, Truck and Motorcycle are all subclasses of the superclass Vehicle. They all inherit common attributes from vehicle such as speed, colour etc. while they have different attributes also i.e Number of wheels in Car is 4 while in Motorcycle is 2.

Superclasses /Parent class /Base class=

A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass. The superclass is also known as the parent class or base class.

In the above example, Vehicle is the Superclass and its subclasses are Car, Truck and Motorcycle.

Inheritance=

Inheritance is basically the process of basing a class on another class i.e to build a class on a existing class. The new class contains all the features and functionalities of the old class in addition to its own.

The class which is newly created is known as the subclass or child class and the original class is the parent class or the superclass.

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

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.

So 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 can not hold object of superclass and could access the methods of both superclass and subclass.

35. Explain an interface?

Inheritance in Java is the concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. Inheritance is performed between two types of classes:

Parent class (Super or Base class)
Child class (Subclass or Derived class)
A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.
Types of inheritance in Java
Java supports four types of inheritance which are:
Single Inheritance: In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.
Multilevel Inheritance: When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.
Hierarchical Inheritance: When a class has more than one child classes (subclasses) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.
Hybrid Inheritance: Hybrid inheritance is a combination of two or more types of inheritance.

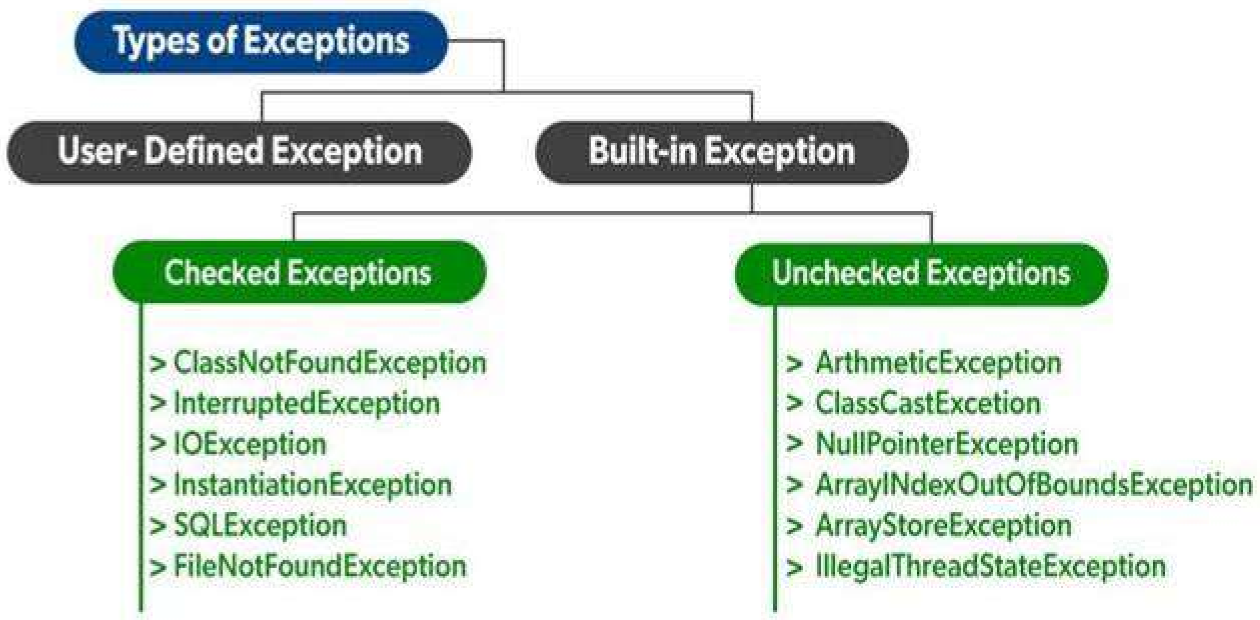
36. Explain exception handling?

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.
Major reasons why an exception Occurs
Invalid user input
Device failure
Loss of network connection
Physical limitations (out of disk memory)
Code errors
Opening an unavailable file
Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc.
Errors are usually beyond the control of the programmer, and we should not try to handle errors.

Differences between Error and Exception
Error: An Error indicates a serious problem that a reasonable application should not try to catch.
Exception: Exception indicates conditions that a reasonable application might try to catch.

Types of Exceptions:



37. Explain the difference among structure and a class?

Class :

Class is a blueprint or set of instructions to build specific types of objects. It consists of a list of data members, and a set of operations are performed on the class.
It is a user-defined data type that holds its own data members.
Class is a basic building block of object-oriented programming (OOP) languages.
Class is a logical abstraction.
It stores multiple types of data and is also used to store functions.
To access the class members, you need to create an instance of the class called objects.
Syntax :

```
class Bike
 {
   public:
int year;
string model_name;
string brand_name;
};
```

Structure :

The Structure is a collection of various types of variables, and these variables can be built-in or user-defined datatypes.
Structure elements are stored at contiguous memory locations.
A structure can have static data members, access specifiers, and support inheritance.
Members of the Structure can be passed to a function
Members in the Structure can be accessed using the dot (.) operator.

Syntax :

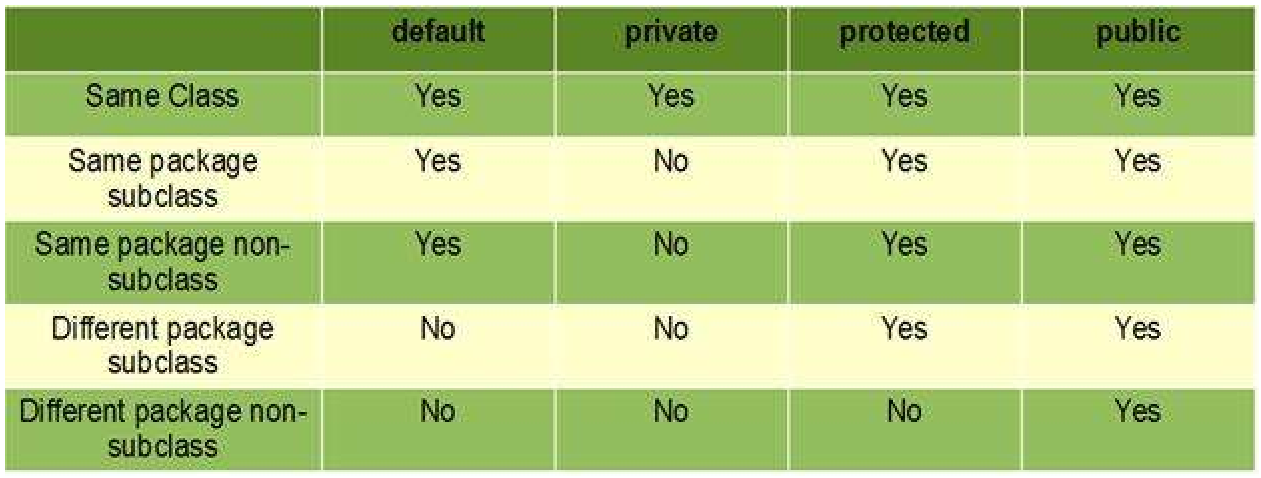
```
struct StructureName
 {
   member 1;
   member 2;
   ……….
   ……….
   member N;
};
```

Differences and similarities between Structure and Class

Structure is a value type, while classes are reference types.
Structure is not inheritable, while classes are.
Classes use Heap allocation while Structure uses stack allocation.
Structure must have at least one non-shared variable while class can be empty.
By default, all structure elements are public, while in-class variables and constants are private, and the others are public.
Both class and Structure are container type, i.e., they can contain others as a member.
Both can implement interfaces and have shared constructors, with or without parameters

38. Explain the default access modifier in a class?

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. The fields in an interface are implicitly public static final and the methods in an interface are by default public.
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.



	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

39. Explain a pure virtual function?

Pure virtual functions are also called ‘do nothing functions’.
Example :
virtual void abc() = 0;

When a pure virtual function is declared in the base class, the compiler necessitates the derived classes to define those functions or redeclare them are pure virtual functions. The classes containing pure virtual functions cannot be used to declare objects of their own. Such classes are called as abstract base classes. 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.

40. Explain dynamic or run time polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable

compile-time polymorphism	**Runtime polymorphism**
• In compile-time polymorphism, call to a method is resolved at compile-time. • It is also known as static binding, early binding, or overloading • Overloading is a way to achieve compile-time polymorphism in which, we	• In runtime polymorphism, call to an overridden method is resolved at runtime. • It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch. • Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class.

can define multiple methods or constructors with different signatures. • It provides fast execution because the type of an object is determined at compile-time • Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time.	• It provides slower execution as compare to compile-time because the type of an object is determined at run-time. • Run-time polymorphism provides more flexibility because all the things are resolved at runtime.

41. Do we require a parameter for constructors?
A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor

42. Explain static and dynamic binding?
Static binding:-
The binding which can be resolved at compile time by the compiler is known as static or early binding. The binding of all the static, private, and final methods is done at compile-time.
Dynamic binding:-
In Dynamic binding compiler doesn’t decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.

43. How many instances can be created for an abstract class?
The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure.

44. Explain the default access specifiers in a class definition?
Default is a keyword that is used as an access modifier for methods and variables.
Using this access modifier will make your class, variable, method or constructor acessible from own class or package, it will be also is set if no access modifier is present.

45. Which OOPS concept is used as reuse mechanism?
Inheritance is the feature that provides a reuse mechanism.
This mechanism provides reusability to the user. While abstraction, encapsulation, and dynamic binding have different functionalities in the OOP paradigm

46. Define the Benefits of Object Oriented Programming?
The Benefits of Object Oriented Programming are as follows:-
Modularity for easier troubleshooting. ...
Reuse of code through inheritance. ...
Flexibility through polymorphism. ...
Effective problem solving. ...
What to know about OOP developer jobs.

47. Explain method overloading?
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.
Advantage of method overloading is that it increases the readability of the program

48. Explain the difference among early binding and late binding?

Early Binding	Late Binding
It is a compile-time process	It is a run-time process
The method definition and method call are linked during the compile time.	The method definition and method call are linked during the run time
Actual object is not used for binding.	Actual object is used for binding.
For example: Method overloading	For example: Method overriding
Program execution is faster	Program execution is slower

49. Explain early binding? Give examples?

Early Binding:-
The binding which can be resolved at compile time by the compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time.
For example: Method overloading

50. Explain loose coupling and tight coupling?

Tight Coupling:-
The tightly coupled object is an object that needs to know about other objects and is usually highly dependent on each other's interfaces.
Changing one object in a tightly coupled application often requires changes to a number of other objects.
In the small applications, we can easily identify the changes and there is less chance to miss anything. But in large applications, these inter-dependencies are not always known by every programmer and there is a chance of overlooking changes.
Loose Coupling
Loose coupling is a design goal to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component.
Loose coupling is a much more generic concept intended to increase the flexibility of the system, make it more maintainable and makes the entire framework more stable.

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

Ans :

Write in brief abstract class.

Ans :

- Abstract class is a class which has been declared abstract by using the ‘abstract’ keyword.
- Ex.

```
        abstract class Test {
                  //Code
                                }
```

- An abstract class may or may not have any abstract method.
- But if any class has any abstract method then class must be declared as abstract.
- We can not create object of an abstract class.
- Abstract class data and methods can be used through child class' objects only.
- 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.

Example of an abstract class that has abstract and non-abstract methods.

```
 abstract class Bike
{
     Bike()
        {
                System.out.println("bike is created");
        }
  abstract void run();
  void changeGear()
        {
                System.out.println("gear changed");
        }
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike
{
    void run()
```

```
                {
                                System.out.println("running safely..");
                }
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2
{
        public static void main(String args[])
        {
       Bike obj = new Honda();
       obj.run();
       obj.changeGear();
        }
}
```

52.Define the Benefits of oops over pop?

Ans :

● Encapsulation is used to hide the data.

●The existing code can be reused.

● Adding new data and functions is easy.

● Inheritance property is used.

● Used for solving big problems.

53.Explain Generalization and Specialization?

Ans :

● Generalization : The process of converting subclass type into superclass type iscalled generalization in java. This is because we are making the subclass to becomemore general so that its scope can be more widening.
This conversion is also called widening or upcasting in referenced data types.

● Specialization : The conversion of a superclass type into subclass type is called specialization in java.
Specialization means going down from a more general form to a more specific form. Thus, its scope will be narrowed. Hence, this conversion is also called narrowing or down-casting in referenced data types.
Specialization is not safe because classes will be more and more specific. In this case, we will need cast operator.

Ex. Let us consider a superclass Mobile and subclasse Samsung whose subclass is SamsungGalaxy. When we talk about a mobile, In general, it may represent any kind of mobile. So, here, the scope is widened.
Now, suppose we talk about Samsung mobile, then we come down one step in the hierarchy of inheritance and we have eliminated any other kind of mobiles.
Thus, we are becoming more specific. When we still come down to Samsung Galaxy, we are pointing only Samsung Galaxy mobile and not any other Samsung mobile.
Thus, this is very specific. This means that when we move down from superclass to subclasses, we are becoming more and more specific.

54.Write in brief Association, Aggregation and Composition?

Ans :

● Association:
Association relationship is a structural relationship in which different objects are linked within the system. It exhibits a binary relationship between the objects representing an activity. It depicts the relationship between objects, such as a teacher, can be associated with multiple teachers.
It is represented by a line between the classes followed by an arrow that navigates the direction, and when the arrow is on both sides, it is then called a bidirectional association. We can specify the multiplicity of an association by adding the adornments on the line that will denote the association.
The composition and aggregation are two subsets of association.

● Aggregation :
Aggregation is a subset of association, is a collection of different things. It represents has a relationship. It is more specific than an association. It describes a part-whole or part-of relationship. It is a binary association, i.e., it only involves two classes. It is a kind of relationship in which the child is independent of its parent.
For example:Here we are considering a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation

relationship.

● Composition:

The composition is a part of aggregation, and it portrays the whole-part relationship. It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted. It exists between similar objects.
Example: If a file is placed in a folder and that is folder is deleted. The file residing inside that folder will also get deleted at the time of folder deletion.

55.Write in brief Object Composition vs. Inheritance.

Ans :

S.NO	Inheritance	Composition
1.	In inheritance, we define the class which we are inheriting(super class) and most importantly it cannot be changed at runtime	Whereas in composition we only define a type which we want to use and which can hold its different implementation also it can change at runtime. Hence, Composition is much more flexible than Inheritance.
2.	Here we can only extend one class, in other words more than one class can’t be extended as java do not support multiple inheritance.	Whereas composition allows to use functionality from different class.
3.	In inheritance we need parent class in order to test child class.	Composition allows to test the implementation of the classes we are using independent of parent or child class.
4.	Inheritance cannot extend final class.	Whereas composition allows code reuse even from final classes.
5.	It is an is-a relationship.	While it is a has-a relationship

56.Explain cohesion?

Ans :

● Cohesion in Java is the Object-Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose. In object-oriented design, cohesion refers to how a single class is designed.

● The more focused a class is, the more is the cohesiveness of that class.

● The advantage 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.

● Example: Suppose we have a class that multiplies two numbers, but the same class creates a pop-up window displaying the result. This is an example of a low cohesive class because the window and the multiplication operation don’t have much in common. To make it high cohesive, we would have to create a class Display and a class Multiply. The Display will call Multiply’s method to get the result and display it. This way to develop a high cohesive solution.

57.Explain “black-box-reuse” and “white-box-reuse”?

Ans :

Explain “this”

Ans :

● Reference to the current object. That is, this is always a reference to an object on which method was invoked.

● ‘this’ can be used to call the overloaded constructor from the other constructors within the same class. But it should always be the first statement within the constructor.

```
 Ex. Cricketer(int totRun)
        {
                this();
                                //Calling no argument constructor of Cricketer class.
   }
```

● this ( ) can not be used within any other methods other than constructors

```
Ex. void updateTotalRuns(int totRuns)
         {
               this();       //Compilation Error.
         }
```

60.Write in brief static member and member functions.
Ans :

Static Members :
Variables and methods declared using keyword static are called static members of a class. As we know that non-static variables and methods belong to instance. But static members (variables, methods) belong to class. Static members are not part of any instance of the class. Static members can be accessed using class name directly, in other words, there is no need to create instance of the class specifically to use them.
Static members can be of two types:
-- Static Variables :Static variables are also called class variables because they can be accessed using class name, whereas, non static variables are called instance variables and can be accessed using instance reference only. Static variables occupy single location in the memory. These can also be accessed using instance reference
-- Static Methods :Static methods are also called class methods. A static method belongs to class, it can be used with class name directly. It is also accessible using instance references.
Static methods can use static variables only, whereas non-static methods can use both instance variables and static variables.

Q61. How will you relate unrelated classes or how will you achieve polymorphism without using the base class?

As you progress in an object-oriented design, you will likely encounter objects in the problem domain that contain other objects. In this situation you will be drawn to modeling a similar arrangement in the design of your solution. In an object-oriented design of a Java program, the way in which you model objects that contain other objects is with composition, the act of composing a class out of references to other objects. With composition, references to the constituent objects become fields of the containing object.

Q62.What is Diamond problem in java?

The diamond problem is a common problem in Java when it comes to inheritance. Inheritance is a very popular property in an object-oriented programming language, such as C++, Java, etc. There are different types of inheritance such as, single, multiple, multi-level, and hybrid inheritance. But remember that Java does not support the multiple inheritance because of the diamond problem.
As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super-class A, then it is called simple inheritance, and Java supports them.
Multi-level inheritance allows a child class to inherit properties from a class that can inherit properties from some other classes. For example, class C can inherit its property from B class which itself inherits from A class. Java also supports them.
What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the diamond problem. In the above figure, we find that class D is trying to inherit form class B and class C, that is not allowed in Java.
It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPPs languages. It is sometimes referred to as the deadly diamond of death.

Q62. Explain the solution for diamond problem?

The solution to the diamond problem is default methods and interfaces. We can achieve multiple inheritance by using these two things.
The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.
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.

64. Explain the need of abstract class?

Abstract classes permit providing a partial set of default implementations of methods in a class. Since they're incomplete, they can't be instantiated and used as they stand, but they can be subclassed to add the missing details in a way that's specific to that particular implementations, and those subclasses can be instantiated.
Without abstract classes, you would have to provide dummy implementations of the methods you intend to override ... which could be done, but then there'd be the risk of forgetting to implement one of them. Having some methods remain entirely abstract ensures that the real implementations have to fill in the gaps, or continue to be abstract themselves and force their descendents to do so.
It's not something the language couldn't live without. But it's very useful. You'll discover just how useful as you become more proficient in Java and OO design.

65.Why can't we instantiate abstract class?

It is a class that is specifically designed to be only derived from.that class has pure virtual methods that must be overriden by the class inherits abstract class. instantiating a class that has pure virtual methods is pointless and might cause compiler errors ...

66.Can abstract class have constructors?

Yes,The main purpose of the constructor is to initialize the newly created object. In abstract class, we have an instance variable, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variables, therefore abstract classes have a constructor. Also, even if we don't provide any constructor the compiler will add default constructor in an abstract class.
An abstract class can be inherited by any number of sub-classes, thus functionality of constructor present in abstract class can be used by them.
The constructor inside the abstract class can only be called during constructor chaining i.e. when we create an instance of sub-classes. This is also one of the reasons abstract class can have a constructor.

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

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure. For an abstract class in the OOP paradigm, we cannot instantiate it.

68.Which keyword can be used for overloading?

If both parent & child classes have the same method, then the child class would override the method available in its parent class. By using the super keyword we can take advantage of both classes (child and parent) to achieve this.

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

A default access modifier in Java has no specific keyword. Whenever the access modifier is not specified, then it is assumed to be the default. The entities like classes, methods, and variables can have a default access.
A default class is accessible inside the package but it is not accessible from outside the package i.e. all the classes inside the package in which the default class is defined can access this class. Similarly a default method or variable is also accessible inside the package in which they are defined and not outside the package.

```
classBaseClass
{
   voiddisplay()      //no access modifier indicates default modifier
     {
        System.out.println("BaseClass::Display with 'dafault' scope");
     }
}

classMain
{
   publicstaticvoidmain(String args[])
     {
       //access class with default scope
       BaseClass obj = newBaseClass();

       obj.display();    //access class method with default scope
     }
}
```

In the above program, we have a class and a method inside it without any access modifier. Hence both the class and method display has default access. Then we see that in the method, we can directly create an object of the class and call the method.

70. Define all the operators that cannot be overloaded?

In C++ we can overload some operators like +, -, [], -> etc. But we cannot overload any operators in it. Some of the operators cannot be overloaded. These operators are like below
? "." Member access or dot operator
? "? : " Ternary or conditional operator
? "::" Scope resolution operator
? ".*" Pointer to member operator
? "sizeof" The object size operator
? "typeid" Object type operator
These operators cannot be overloaded because if we overload them it will make serious programming issues.

71.Explain the difference among structure and a class?

Ans:Class

It is defined using ‘class’ keyword.
When data is defined in a class, it is stored in memory as a reference.
It gets memory allocated only when an object of that class is created.
The reference type (before creating an object) is allocated on heap memory.
They can have constructors and destructors.
It can use inheritance to inherit properties from base class.
The ‘protected’ access modifier can be used with the data members defined inside the class.

Structure

The ‘struct’ keyword is used to define a structure.
Every member in the structure is provided with a unique memory location.
When the value of one data member is changed, it doesn’t affect other data members in structure.
Total size of the structure is equivalent to the sum of the size of every data member.
It is used to store various data types.
It takes memory for every member which is present within the structure.
Its instance can be created without a keyword.
It doesn’t support protected access modifier.
It doesn’t support inheritance.
It doesn’t have a constructor or destructor.
The values allocated to structures are stored in stack memory.

72.Explain the default access modifier in a class?

There are four types of Java access modifiers:
Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
Default: 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.
Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

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

No-argument constructor: A constructor that has no parameter is known as the No-argument or Zero argument constructor. If we don’t define a constructor in a class, then the compiler creates a constructor(with no arguments) for the class. And if we write a constructor with arguments or no arguments then the compiler does not create a default constructor.
Note: Default constructor provides the default values to the object like 0, null, etc. depending on the type.
Parameterized Constructor: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.
Default Constructor: A constructor that has no parameters is known as default the constructor. A default constructor is invisible. It is taken out.It is being overloaded and called a parameterized constructor. The default constructor changed into the parameterized constructor. But Parameterized constructor can’t change the default constructor.

74.Explain a ternary operator?

In Java, the ternary operator is a type of Java conditional operator. The meaning of ternary is composed of three parts. The ternary operator (? :) consists of three operands. It is used to evaluate Boolean expressions. The above statement states that if the condition returns true, expression1 gets executed, else the expression2 gets executed and the final result stored in a variable.
Example:

```
{ intn1 = 5, n2 = 10, max;

    System.out.println("First num: "+ n1);
    System.out.println("Second num: "+ n2);

    // Largest among n1 and n2
    max = (n1 > n2) ? n1 : n2;    }
```

75.Do We Require Parameter For Constructors?

Default Constructor – A constructor that accepts no parameter is called Default Constructor. It is not necessary to have a constructor block in your class definition. If you don't explicitly write a constructor, the compiler automatically inserts one for you.

76.Explain Sealed Modifiers?

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 . You can also use the sealed modifier on a method or property that overrides a virtual method or property in a base class.

77.Explain The Difference Between New And Override.

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.

78.How Can We Call The Base Method Without Creating An Instance?

Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class.

79. Define The Various Types Of Constructors?

No-argument constructor: A constructor that has no parameter is known as the No-argument or Zero argument constructor. If we don't define a constructor in a class, then the compiler creates a constructor(with no arguments) for the class. And if we write a constructor with arguments or no arguments then the compiler does not create a default constructor.
Note: Default constructor provides the default values to the object like 0, null, etc. depending on the type.
Parameterized Constructor: A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.
Default Constructor: A constructor that has no parameters is known as default the constructor. A default constructor is invisible. It is taken out.It is being overloaded and called a parameterized constructor. The default constructor changed into the parameterized constructor. But Parameterized constructor can't change the default constructor.

80.Define Manipulators?

A Java manipulator is your own code in Java that takes records from any number of pipeline components in Forge or, optionally, your source data, and changes it according to your.processing requirements. A Java manipulator can then write any records you choose to its output

Ans 80)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.

Manipulators are special functions that can be included in the I/O statement to alter the format parameters of a stream.
Manipulators are operators that are used to format the data display.
To access manipulators, the file iomanip.h should be included in the program.

For example, if we want to print the hexadecimal value of 100 then we can print it as:
cout<<setbase(16)<<100

Ans 81)A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

Keywords
Identifiers
Constants
Special Symbols
Operators

1.Keywords :Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.
2.Identifier : Identifiers are used as the general terminology for naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value.
3.Constants :Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals.
Constants may belong to any of the data type.

Final datatype variable name;
4.Operators:Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are
· Unary Operator,
· Arithmetic Operator,
· Shift Operator,
· Relational Operator,
· Bitwise Operator,
· Logical Operator,
5. Special symbols :The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.
[] () {}, ; * =

Brackets[]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
Parentheses():These special symbols are used to indicate function calls and function parameters.
Braces{}: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
comma (,): It is used to separate more than one statements like for separating parameters in function calls.
semi colon : It is an operator that essentially invokes something called an initialization list.
asterick (*): It is used to create pointer variable.
assignment operator: It is used to assign values.

A 82)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. The languages that support Structured programming approach are:

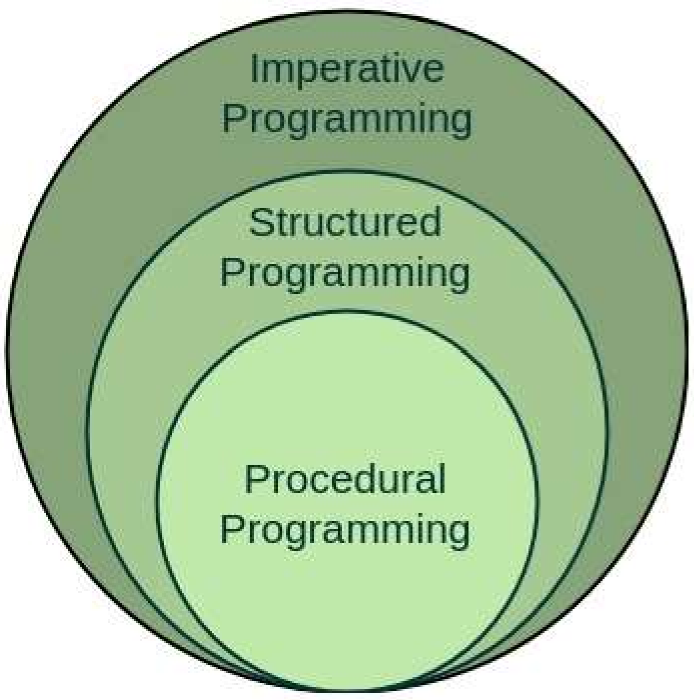
C
C++
Java

Advantages of Structured Programming Approach:

Easier to read and understand
User Friendly
Easier to Maintain
Mainly problem based instead of being machine based
Development is easier as it requires less effort and time
Easier to Debug
Machine-Independent, mostly.

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.



A 83) As we know that abstraction refers to hiding the internal implementation of the feature and only showing the functionality to the users. i.e. what it works (showing), how it works (hiding).
Consider using abstract classes if any of these statements apply to your situation:

In the java application, there are some related classes that need to share some lines of code then you can put these lines of code within the abstract class and this abstract class should be extended by all these related classes.

You can define the non-static or non-final field(s) in the abstract class so that via a method you can access and modify the state of the object to which they belong.

You can expect that the classes that extend an abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).

Consider using interfaces if any of these statements apply to your situation:

It is a total abstraction, all methods declared within an interface must be implemented by the class(es) that implements this interface.

A class can implement more than one interface. It is called multiple inheritances.

You want to specify the behaviour of a particular data type but are not concerned about who implements its behaviour.

Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited forproviding common functionality to unrelated classes. If you are designing small, concise bits of functionality, use interfaces. If you are designing large functional units, use an abstract class.
A 84)There are various scenarios where we can use private constructors. The major ones are

Internal Constructor chaining

Singleton class design pattern

What is a Singleton class?
As the name implies, a class is said to be singleton if it limits the number of objects of that class to one. We can’t have more than a single object for such classes.
Singleton classes are employed extensively in concepts like Networking and Database Connectivity.
Design Pattern of Singleton classes:
The constructor of singleton class would be private so there must be another way to get the instance of that class. This problem is resolved using a class member instance and a factory method to return the class member.
A 85)No, we cannot override private or static methods in Java.
Private methods in Java are not visible to any other class which limits their scope to the class in which they are declared.
In Java, methods declared as private can never be overridden, they are in-fact bounded during compile time.
A 86)Inheritance is a substantial rule of any Object-Oriented Programming (OOP) language but still, there are ways to prevent method overriding in child classes which are as follows:
1.Using static method
This is the first way of preventing method overriding in the child class. If you make any method static then it becomes a class method and not an object method and hence it is not allowed to be overridden as they are resolved at compilation time and overridden methods are resolved at runtime.
2.Using private access modifier

Making any method private reduces the scope of that method to class only which means absolutely no one outside the class can reference that method.
3.Using default access modifier
This can only be used when the method overriding is allowed within the same package but not outside the package. Default modifier allows the method to be visible only within the package so any child class outside the same package can never override it.
4.Using the final keyword method
The final way of preventing overriding is byusing the final keyword in your method. The final keyword puts a stop to being an inheritance. Hence, if a method is made final it will be considered final implementation and no other class can override the behaviour.
A 87) For some user-defined types or type members who have a default access level, you can’t specify them explicitly at all. For example interface and enum members are always public and no access modifiers are allowed. If you are trying to add any access modifiers explicitly, it causes acompile-time error.
A 88)
A 89)If a class has multiple methods having same name but parameters of the method should be different 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 to understand the behaviour of the method because its name differs.

Method overloading in java is based on the number and type of the parameters passed as an argument

to the methods. We can not define more than one method with the same name, Order, and 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 types. It will throw a compile-time error. If both methods have the same parameter types, but different return types, then it is not possible.
Parameters should be different means
1. Type of parameter should be different.

```
void add(int a, int b){
    System.out.println("sum ="+(a+b));


  void  add(double a, double b){
    System.out.println("sum="+(a+b));
```

2.Number of parameter should be different.

```
void add(int a, int b){
    System.out.println("sum ="+(a+b));


   void  add(int a, int b,int c){
    System.out.println("sum="+(a+b+c));
```

A 90) Yes, we can declare an abstract class with no abstract methods in Java.
An abstract class means that hiding the implementation and showing the function definition to the user.
An abstract class having both abstract methods and non-abstract methods.
For an abstract class, we are not able to create an object directly. But Indirectly we can create an object using the subclass object.
A Java abstract class can have instance methods that implement a default behavior.
An abstract class can extend only one class or one abstract class at a time.
Declaring a class as abstract with no abstract methods means that we don't allow it to be instantiated on its own.
An abstract class used in Java signifies that we can't create an object of the class directly.

91 .Explain the default access modifier of a class?
Answer :: Default: 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.

92. Can function overriding be explained in same class?
Answer ::No,Function Overriding can only be defined in different class because it treated as Base and Derived or parent-child relationship

93. Does function overloading depends on Return Type?
Answer::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

94. Can abstract class have a constructor?
Answer::Abstract classes can have constructors! 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.

95. Define rules of Function overloading and function overriding?
Answer :: In function overloading, two or more functions can own the same name, but the parameters will be different. Where as Function overriding permit us to redefine a method with the same name and signature
2 .There is no requirement of the inheritance concept here.In function overriding, we need an inheritance concept.
3-In the case of function overloading, the signatures should be different.
In the case of function overriding, the signatures should be the same.