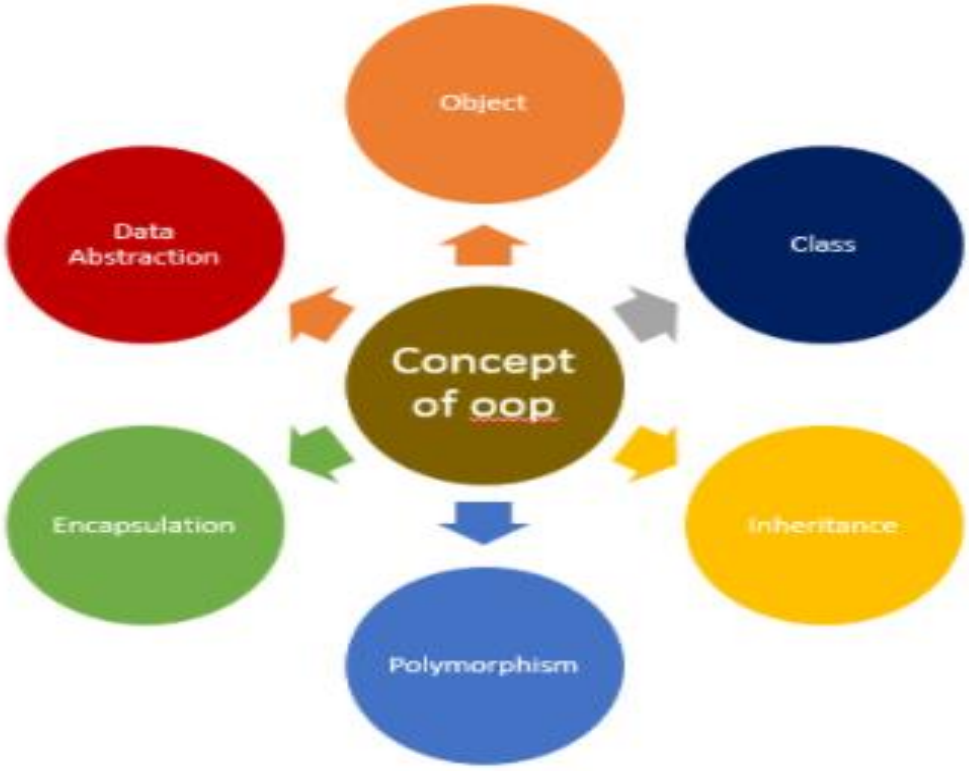
Q1.Explain OOPS?

Answer: Object-oriented programming is a method used for designing a program using classes and objects. Object-oriented programming is also called the core of java. Object-oriented programming organizes a program around objects and well-defined interfaces. This can also be characterized as data controlling for accessing the code. In this type of approach, programmers define the data type of a data structure and the operations that are applied to the data structure. This implies software development and maintenance by using some of the concepts:

- Object
- Class
- Abstraction
- Inheritance
- Polymorphism
- Encapsulation



Q2. Explain an abstraction? Real life example.

Answer: Abstraction is the process of extracting the relevant properties of an object while ignoring nonessential details relative to the perspective of the viewer.

e.g.

• A doctor sees (abstracts) the person as patient. The doctor is interested in name, height, weight, age, blood group, previous or existing diseases etc of a person whereas An employer sees (abstracts) a person as Employee. The employer is interested in name, age, health, degree of study, work experience etc of a person.

• For a sales tracking system relevant attributes of a salesperson might be: name, number of vehicles sold, value of vehicles sold, list of customers, commission rate, total commissions.

• A car dealer might view a car from the standpoint of its selling features. Relevant properties include price, color, optional equipment, and length of warranty. On the other hand , mechanic views the car from the standpoint of systems that require maintenance. Here relevant properties include the type of oil, the size of oil filter, and the number and type of spark plug.

Q3. Explain encapsulation? Real life example.

Answer: Encapsulation is the process of separating the aspects of an object into external and internal aspects. The external aspects of an object need to be visible or known , to other objects in the system. The internal aspects are details that should not affect other parts of the system. Hiding the internal aspects of an object means that they can be changed without affecting external aspects of the system.

Eg .

1. Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, In this case, anyone can know your account balance. So, would you like it? Obviously No.

So, they declare balance variable as private for making your account safe, so that anyone cannot see your account balance.

The person who has to see his account balance, will have to access only private members through methods defined inside that class and this method will ask your account holder name or user Id, and password for authentication.

Thus, we can achieve security by utilizing the concept of data hiding. This is called Encapsulation in Java.

Q4. Explain the relationship among abstraction and encapsulation?

Answer: abstraction focuses upon the observable behavior of an object, whereas encapsulation focuses upon the implementation that gives rise to this behavior. abstraction is a perspective and encapsulation is implementation to achieve that perspective.

Q5. Explain polymorphism ?

Answer: Polymorphism refers to many forms, or it is a process that performs a single action in different ways. It occurs when we have many classes related to each other by inheritance. Polymorphism is of two different types, i.e., compile-time polymorphism and runtime polymorphism. One of the examples in Compile time polymorphism is that when we overload a static method in java. Run time polymorphism is also called a dynamic method dispatch is a method in which a call to an overridden method is resolved at run time rather than compile time. In this method, the overridden method is always called through the reference variable. By using method overloading and method overriding, we can perform polymorphism. Generally, the concept of polymorphism is often expressed as one interface, multiple methods. This reduces complexity by allowing the same interface to be used as a general class of action.

Polymorphism in java can be classified into two types:

1. Static / Compile-Time Polymorphism

   Compile-Time polymorphism in java is also known as Static Polymorphism. to resolved at compile-time which is achieved through Method Overloading.

2. Dynamic / Runtime Polymorphism

   Runtime polymorphism in java is also known as Dynamic Binding which is used to call to an overridden method that is resolved dynamically at runtime rather than at compile-time.

   eg.

   A Person who knows more than two languages he can speak in a language which he knows. Here person is Object and speak is polymorphisam.

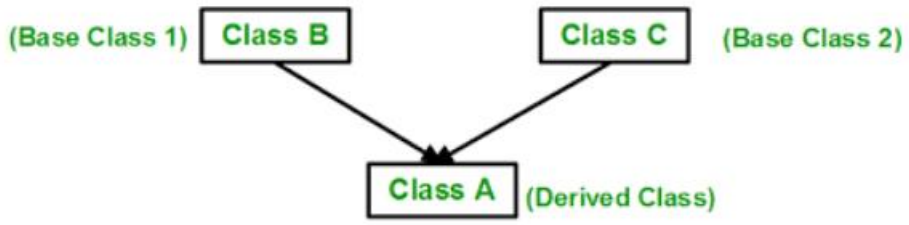
Q6. Explain Inheritance?

Answer: Inheritance is a method in which one object acquires/inherits another object's properties, and inheritance also supports hierarchical classification. The idea behind this is that we can create new classes built on existing classes, i.e., when you inherit from an existing class, we can reuse methods and fields of the parent class. Inheritance represents the parent-child relationship.

For example, a whale is a part of the classification of marine animals, which is part of class mammal, which is under that class of animal. We use hierarchical classification, i.e., top-down classification. If we want to describe a more specific class of animals such as mammals, they would have more specific attributes such as teeth; cold-blooded, warm-blooded, etc. This comes under the subclass of animals where animals come under superclass. The subclass is a class which inherits properties of the superclass. This is also called a derived class. A superclass is a base class or parental class from which subclass inherits properties.

We use inheritance mainly for method overriding and R:

To inherit a class, we use the extend keyword.



There are five types of inheritance single, multilevel, multiple, hybrid and hierarchical.

1. **Single level**

In this one class i.e., derived class inherits properties from its parental class. This enables code reusability and also adds new features to the code. Example: class b inherits properties from class a.

Class A is base or parental class and class b is derived class

Syntax:

```
Class a {
…
}
Class b extends class a {
…
}
```

2. **Multilevel**

This one class is derived from another class which is also derived from another class i.e., this class has more than one parental class, hence it is called multilevel inheritance.

Syntax:

```
Class a {
….
}
Class b extends class a {
….
}
Class c extends class b {
…
}
```

3. **Hierarchical level**

In this one parental class has two or more derived classes or we can say that two or more child classes has one parental class.

Syntax:

```

Class a {
…
}
Class b extends class a {
..
}
Class c extends class a {
..
```

4. **Hybrid inheritance**

This is the combination of multiple and multilevel inheritance and in java multiple inheritance is not supported as it leads to ambiguity and this type of inheritance can only be achieved through interfaces.Consider that class a is the parental or base class of class b and class c and in turn class b and class c are parental or base class of class d. Class b and class c are derived classes from class a and class d is derived class from class b and class c.

Q7. How composition is better than inheritance?

Answer: 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.In inheritance 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 where 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.

Q8. Which OOPS concept is used as a reuse mechanism?

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

Q9. Which OOPS concept exposes only the necessary information to the calling functions?

Answer: Data Hiding / Abstraction is the OOPS concept that which exposes only the necessary information to the calling functions.

Q10. Explain a class? Create a class

Answer: Classes are like object constructors for creating objects. The collection of objects is said to be a class. Classes are said to be logical quantities. Classes don't consume any space in the memory. Class is also called a template of an object. Classes have members which can be fields, methods and constructors. A class has both static and instance initializers.

A class declaration consists of:

1. **Modifiers**:Can be public or default access.
2. **Class name:** Initial letter.
3. **Superclass:** A class can only extend (subclass) one parent.
4. **Interfaces:** A class can implement more than one interface.
5. **Body:** Body surrounded by braces, { }.

A class keyword is used to create a class. A simplified general form of the class definition is given below:

```
class classname {
type instance variable 1;
type instance variable 2;
.
.
.
type instance variable n;
type method name 1 (parameter list) {
// body of method
}
type method name 2 (parameter list) {
// body of method
}
type method name n (parameter list) {
// body of method
}
 }
```

The variables or data defined within a class are called as instance variables. Code is always contained in the methods. Therefore, the methods and variables defined within a class are called members of the class. All the methods have the same form as main () these methods are not specified as static or public.

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. See Virtual function in C++.

22. Explain a friend function?

Ans: friend function of a class is defined outside that class's 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.

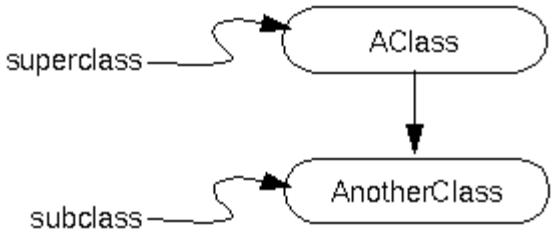
23. Explain function overloading?

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.

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

Ans: A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class.



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

Ans: 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.

26. Explain an abstract class?

Ans: A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Important 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: No, Java doesn't support user-defined operator overloading. The only aspect of Java which comes close to "custom" operator overloading is the handling of + for strings, which either results in compile-time concatenation of constants or execution-time concatenation using StringBuilder/String Buffer. You can't define your own operators which act in the same way though.

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

Ans: Call by Value means calling a method with a parameter as value. Through this, the argument value is passed to the parameter.

While Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.

In call by value, the modification done to the parameter passed does not reflect in the caller's scope while in the call by reference, the modification done to the parameter passed are persistent and changes are reflected in the caller's scope.

29. 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.

USAGE OF JAVA 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.

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).

Q31. Difference among overloading and overriding?

Answer:

Method Overloading	Method Overriding
Method overloading is a compile time polymorphism	Method overriding is a run time polymorphism.
It help to rise the readability of the program.	While it is used to grant the specific implementation of the method which is already provided by its parent class or super class.
It is occur within the class.	While it is performed in two classes with inheritance relationship.
Method overloading may or may not require inheritance.	While method overriding always needs inheritance.

In this, methods must have same name and different signature.	While in this, methods must have same name and same signature.
In method overloading, return type can or can not be be same, but we must have to change the parameter.	While in this, return type must be same or co-variant.

Q32. Whether static method can use non-static members?

Answer:

Yes, a static method can access a non-static variable. This is done by creating an object to the class and accessing the variable through the object.

Q33. Explain a base class, sub class, super class?

Answer:

A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

.

Q34. Write in brief linking of base class, sub class and base object, sub object.

Q35. Explain an interface?

Answer:-

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Q36. Explain exception handling

Answer:

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.

Example:- ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Q37. Explain the difference among structure and a class?

Answer

Class	Structure
Classes are of reference types.	Structs are of value types.
All the reference types are allocated on heap memory.	All the value types are allocated on stack memory.
Allocation of large reference type is cheaper than allocation of large value type.	Allocation and de-allocation is cheaper in value type as compare to reference type.
Class has limitless features.	Struct has limited features.
Class is generally used in large programs.	Struct are used in small programs.
Classes can contain constructor or destructor.	Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor.
Classes used new keyword for creating instances.	Struct can create an instance, with or without new keyword.
A Class can inherit from another class.	A Struct is not allowed to inherit from another struct or class.
The data member of a class can be protected.	The data member of struct can't be protected.
Function member of the class can be virtual or abstract.	Function member of the struct cannot be virtual or abstract.
Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable.	Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable.

Q38. Explain the default access modifier in a class?

Answer:

The Default access modifier is package-private (i.e DEFAULT) and it is visible only from the same package. Your constructor's access modifier would be package-private(default). As you have declared the class public, it will be visible everywhere, but the constructor will not.

Q39. Explain a pure virtual function?

Answer:

A pure virtual function is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract. ... No definition is given in base class. Base class having virtual function can be instantiated i.e. its object can be made.

Q40. Explain dynamic or run time polymorphism?

Answer:

Runtime polymorphism in java is also known as Dynamic Binding or Dynamic Method Dispatch. In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time. Runtime polymorphism is achieved through.

Q41. Do we require a parameter for constructors?

Answer: Constructor may have or may not have parameters

There are two types of constructors

1) Default Constructor

Default constructor is created by JVM and has zero parameters.

2) User defined Constructor

User defined Constructors are created by programmer and has parameters.

Ex-one parameter constructor

```
class Test
{
        int num1;
        int num2;
        Test(int num1,int num2)
        {
                this.num1=num;
                this.num2=ch;
        }
        void display()
        {
                System.out.println(num1);
                System.out.println(num2);
        }
        public static void main(String args[])
        {
        Test t1=new Test(102,17);
        t1.display();
        }
}
```

Output:102

17

Ex-Default Constructor

```
class Test
{
        int num;
        int num2;

        void display()
        {
                System.out.println(num1);
                System.out.println(num2);
        }
        public static void main(String args[])
        {
        Test t1=new Test();
        t1.display();
        }
}
```

Output:0

0

Q42. Explain static and dynamic binding?

Answer: **Connecting a method call to the method body is known as binding.**

There are two types of bindings

1. Static Binding (also known as Early Binding or Compile time binding).
2. Dynamic Binding (also known as Late Binding or Run time binding).

Static Binding

When type of the object is determined at compiled time (by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

Example

```
class Dog{
private void eat()
{
System.out.println("dog is eating...");
}
public static void main(String args[]){
 Dog d1=new Dog();
 d1.eat();
 }
}
```

Output: dog is eating...

Dynamic Binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example

```
class Animal
{
```

```
        void eat()
    {
        System.out.println("animal is eating...");
    }
}
        class Dog extends Animal
        {
         void eat(){System.out.println("dog is eating...Bro");
        }
        public static void main(String args[])
        {
        Animal a=new Dog();
         a.eat();
         }
        }
```

Output: dog is eating...Bro

Q43. How many instances can be created for an abstract class?

Answer: We can not create instance of an abstract class because it is not possible to create object of an abstract class. But we can create number of instances of the class which is extending that abstract class

Q44. Explain the default access specifiers in a class definition?

Answer: If a class does not have any modifier that means it has default modifier.

Which indicates that that class can be accessed only inside package.

Q45. Which OOPS concept is used as reuse mechanism?

Answer: Inheritance concept is used to reuse the specific code.

- Top inheritance is the process of acquiring properties and behaviour from another class.
- “Extends” keyword is used to establish relationship between two classes.
- The class whose properties are being used is called as Parent class, Super class, Base class.
- The class who is using properties is called as Child class, Sub class, Derives class.

Example-

```
 class abc
{
void m1()
{System.out.println("Method one");}
}

 class Test
{
Void m2()
{ System.out.println("Method two");} }
public static void main(String args[])
        {
abc t1=new abc();
        t1.m1();
        t1.m2();
        }
}
```

Output: Method one

Method two

Q46. Define the Benefits of Object Oriented Programming?

Answer: Some of the advantages of OOPS

Re-usability, Data Redundancy,

Code Maintenance, Security,

Design Benefits, Easy Troubleshooting,

Polymorphism Flexibility, and Problem-solving.

Q47. Explain method overloading?

Answer: when a class contains two or more methods having same name and different number of arguments or if number of arguments are same their order must be different.

```
class DisplayOverloading
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(char c, int num)
    {
        System.out.println(c + " "+num);
    }
}
class Sample
{
   public static void main(String args[])
   {
      DisplayOverloading obj = new DisplayOverloading();
      obj.disp('a');
      obj.disp('a',10);
   }
}
```

Output:a

a 10

Q48. Explain the difference among early binding and late binding?

Answer:

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

Q49. Explain early binding? Give examples.

Answer:

Early Binding:

When type of the object is determined at compiled time (by the compiler), it is known as early binding.

If there is any private, final or static method in a class, there is early binding.

Early Binding (also known as Static Binding or Compile time binding).

Example:

```
class Person
{
public void speak()
{
System.out.println("Person speaks");
}
}
class Teacher extends Person
{
public static void speak()
```

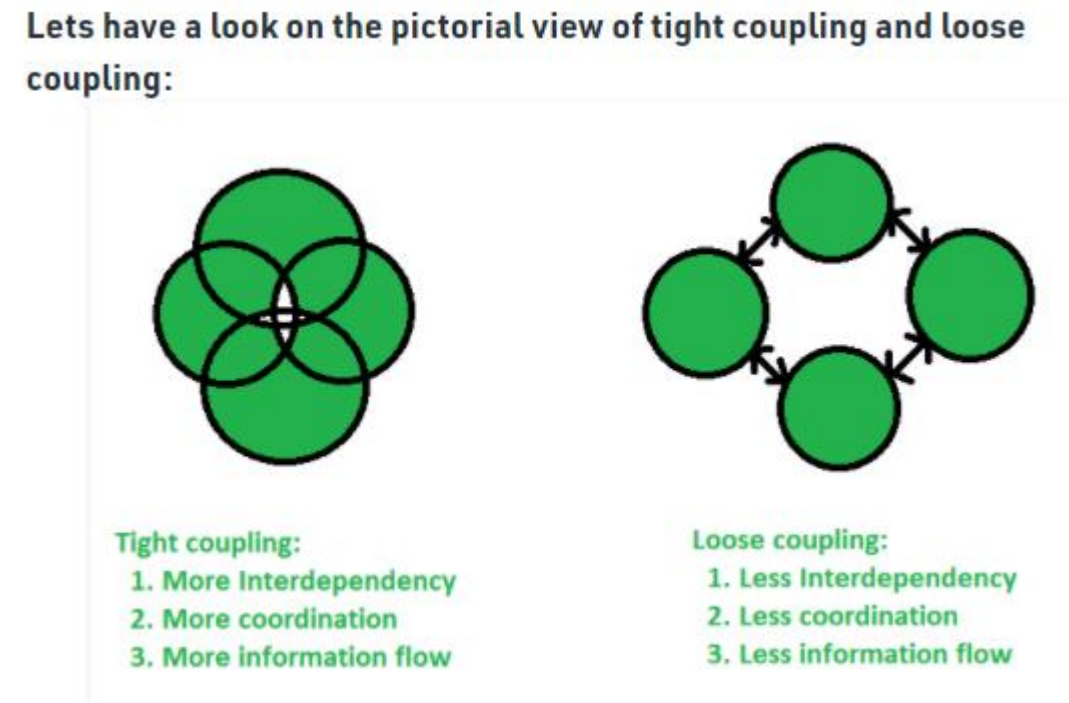
```
{
System.out.println("Teacher speaks");
}
}
public class StaticBinding
{
public static void main( String args[ ])
{
// Reference is of Person type and object is Teacher type
Person obj = new Teacher();
obj.speak();
//Reference and object both are of Person type.
Person obj2 = new Person();
obj2.speak();
}
}
```

Output- person speaks

person speaks

Q50. Explain loose coupling and tight coupling?

Answer:



Loose Coupling:

When two classes, modules, or components have low dependencies on each other, it is called loose coupling in Java. Loose coupling in Java means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling. If a situation requires objects to be used from outside, it is termed as a loose coupling situation.

Example-

```
interface Food {
  public void display();
}
class Italian {
 Food s;
  public Italian(Food s){
  this.s = s;
  }
  public void display(){
    System.out.println("Italian");
    s.display();
  }
}
class Chinese implements Food {
  public Chinese(){}
  public void display(){
    System.out.println("Chinese");
  }
}
class Mexican implements Food {
  public Mexican(){}
  public void display(){
    System.out.println("Mexican");
  }
}
public class Test {
  public static void main(String args[]) throws IOException {
  Food b = new Chinese();
  Food c = new Mexican();
  Italian a = new Italian(b);
    //a.display() will print Italian and Chinese
    a.display();
  Italian a1 = new Italian(c);
    //a.display() will print Italian and Mexican
    a1.display();
  }
}
```

Output-Italian
Chinese
Italian
Mexican

Tight Coupling:

When two classes are highly dependent on each other, it is called tight coupling. It occurs when a class takes too many responsibilities or where a change in one class requires changes in the other class. In tight coupling, an object (parent object) creates another object (child object) for its usage. If the parent object knows more about how the child object was implemented, we can say that the parent and child object are tightly coupled.

Example-

```
class Volume {
  public static void main(String args[]) {
      Cylinder b = new Cylinder(15, 15, 15);
        System.out.println(b.volume);
  }}
 class Cylinder {
  public int volume;
  Cylinder(int length, int width, int height) {
        this.volume = length * width * height;  }}
```

Output- **3375**

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

Ans- n object oriented design, Coupling refers to the degree of direct knowledge that one element has of another. In other words, how often do changes in class A force related changes in class B.

There are two types of coupling:

Tight coupling : In general, 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. The best example of tight coupling is RMI(Remote Method Invocation).


```
// Java program to illustrate

// tight coupling concept

class Subject {

   Topic t = new Topic();
```

```
    public void startReading()
    {
        t.understand();
    }
}
class Topic {
    public void understand()
    {
        System.out.println("Tight coupling concept");
    }
}
```

Explanation: In the above program the Subject class is dependents on Topic class. In the above program Subject class is tightly coupled with Topic class it means if any change in the Topic class requires Subject class to change. For example, if Topic class understand() method change to gotit() method then you have to change the startReading() method will call gotit() method instead of calling understand() method

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. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling.

Example : If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can't do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

```
// Java program to illustrate
// loose coupling concept
public interface Topic
{
    void understand();
}
class Topic1 implements Topic {
public void understand()
    {
```

```
    System.out.println("Got it");
  }
} class Topic2 implements Topic {
public void understand()
  {
    System.out.println("understand");
  }
} public class Subject {
public static void main(String[] args)
  {
    Topic t = new Topic1();
    t.understand();
  }
}
```

Explanation : In the above example, Topic1 and Topic2 objects are loosely coupled. It means Topic is an interface and we can inject any of the implemented classes at run time and we can provide service to the end user.

Que 52 . Write in brief abstract class.

Ans - Abstract class in Java

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

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.

Que 53. Define the Benefits of oops over pop?

Ans -

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 event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language

Que 54. Explain Generalization and Specialization?

Ans -

1. Generalization :

It works on the principle of bottom up approach. In Generalization lower level functions are combined to form higher level function which is called as entities. This process is repeated further to make advanced level entities.

In the Generalization process properties are drawn from particular entities and thus we can create generalized entity. We can summarize Generalization process as it combines subclasses to form superclass.

Example of Generalization –

Consider two entities Student and Patient. These two entities will have some characteristics of their own. For example Student entity will have Roll_No, Name and Mob_No while patient will have PId, Name and Mob_No characteristics. Now in this example Name and Mob_No of both Student and Patient can be combined as a Person to form one higher level entity and this process is called as Generalization Process.

2. Specialization :

We can say that Specialization is opposite of Generalization. In Specialization things are broken down into smaller things to simplify it further. We can also say that in Specialization a particular entity gets divided into sub entities and it’s done on the basis of it’s characteristics. Also in Specialization Inheritance takes place.

Example of Specialization –

Consider an entity Account. This will have some attributes consider them Acc_No and Balance. Account entity may have some other attributes like Current_Acc and Savings_Acc. Now Current_Acc may have Acc_No, Balance and Transactions while Savings_Acc may have Acc_No, Balance and Interest_Rate henceforth we can say that specialized entities inherits characteristics of higher level entity.

Que 55 .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.

> 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.

As you can see from the example given below, the composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

Que 56 . Write in brief Object Composition vs. Inheritance.

Ans - Inheritance:

When we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. In doing this, we can reuse the fields and methods of the existing class without having to write them ourself.

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass

Composition is also fundamental to every language. Even if the language does not support composition (rare these days!), humans still think in terms of parts and components. It would be impossible to break down complex problems into modular solutions without composition.

Que 57. Explain cohesion?

Ans - In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Types of Cohession

Functional Cohesion: Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

Sequential Cohesion: A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

Communicational Cohesion: A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

1. Procedural Cohesion: A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

2. Temporal Cohesion: When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

3. Logical Cohesion: A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

4. Coincidental Cohesion: A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at al

Que 58 . 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 i

nterface.

White box reuse is taking a class/function/code from one project and modifying it to suit the needs of another p

roject.

White-box reuse means that you know how component is implemented. Usually White-box reuse means class i

nheritance.

59) 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 i

n 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

60. Write in brief static member and member function.

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 'co

unter' 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 m

ethod, constructor or any block. When space is allocated for an object in the heap, a slot for each instance variable va

lue 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 ob

ject is destroyed.

Instance variables hold values that must be referenced by more than one method, constructor or block, or essential pa

rts 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 ref

erences 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 meth

ods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectRef

erence.VariableName.

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

1)We can relate two unrelated classes with the help of object creation of class in another class i.e. generally, in which main method is defined.For e.g.

Syntax:-

```
Class A{

A(){}    //Constructor
```

```
Void method(){}
}
Class B{
Public static void main(String… args){    //main method
    A aobj = new aobj();     // Creating an object of an class. It will give call to constructor
    aobj.method();    //call to the method inside a Class A
}
}
```

Above e.g. clears that we can relate to classes A and B by creating an object in side Class B and also call a method of Class A.

2)**Achieving polymorphism without using base class:-**

With help of method overloading in between a class or in a same without using base class polymorphism can be achieved without using base class. For e.g.

Syntax:-

```
class OverloadDemo {
 void test() { }      // Overload test for zero parameter.
void test(int a) { } }        // Overload test for one integer parameter.
 void test(int a, int b) { }        // Overload test for two integer
 double test(double a) { }      // Overload test for a double parameter.
}
class Overload {
public static void main(String args[]) {
 OverloadDemo ob = new OverloadDemo();
 double result; // call all versions of test()
ob.test();
 ob.test(10);
 ob.test(10, 20);
result = ob.test(123.25);
 }
```

The above example is a compile time polymorphism without using inheritance or base class.

Q62. Explain Diamond problem?

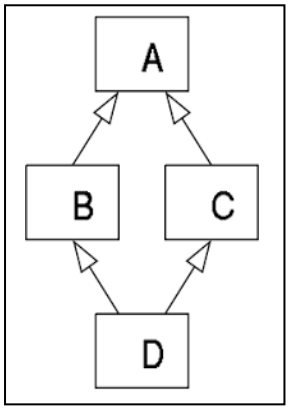
Answer: In Java, the diamond problem is related to multiple inheritance. Sometimes it is also known as the deadly diamond problem or deadly diamond of death**.** In this section, we will learn what is the demand problem in Java and what is the solution to the diamond problem**.**

e.g.

```
class A
{
public void display()
{
System.out.println("class A display() method called");
}
}
class B extends A
{
@Override
public void display()
{
System.out.println("class B display() method called");
}
}
class C extends A
{
@Override
public void display()
{
System.out.println("class C display() method called");
}
}  //not supported in Java
public class D extends B,C
{
public static void main(String args[])
{
D d = new D();
//creates ambiguity which display() method to call
d.display();
}
}
```

Output:-error '{' expected public class D extends B,C

Class B and class C inherits the class A. The display() method of class A is overridden by the class B and class C.Class D inherits the class B and class C (which is invalid in Java). Assume that we need to call the display() method by using the object of class D, in this scenario Java compiler does not know which display() method to call. Therefore, it creates ambiguity. Due to these complications and ambiguities, Java does not support multiple inheritance. It creates problems during various operations, for example, constructor chaining and casting. Hence, it will be good to avoid it for making things simple.



Q63. Explain the solution for diamond problem?

Answer: 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.

For e.g. using interface

```
interface DemoInterface
{
default void display()
{
System.out.println("The dispaly() method invoked");
}
}
interface DemoInterface1 extends DemoInterface {}
interface DemoInterface2 extends DemoInterface {}
public class DemoClass implements DemoInterface1, DemoInterface2
{
public static void main(String args[])
{
DemoClass obj = new DemoClass();
obj.display();
}
}
```

Output : The dispaly() method invoked

Q64. Explain the need of abstract class?

Answer: The need of abstraction process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Q65. Why can’t we instantiate abstract class?

Answer: We have heard that abstract class are classes which can have abstract methods and it can’t be instantiated. We cannot instantiate an abstract class in Java because it is abstract, it is not complete, hence it cannot be used. An abstract class is not complete! The author marked it abstract to tell you that some implementation is missing in the code. The author has done some of the work, but you must fill in some bits yourself to get it working. The abstract keyword ensures that no one would accidentally initiate this incomplete class.

Q66. Can abstract class have constructors?

Answer: 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. For e.g. with help of super keyword it can be call from another class

Q67. How many instances can be created for an abstract class?

Answer: No we can't, instead we 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.

Q68. Which keyword can be used for overloading?

Answer: For function overlading there is no such type of keyword. In function overloading only function name is same but parameter passing are different and also return type.

Q69. Explain the default access specifiers in a class definition?

Answer: 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.

Q70. Define all the operators that cannot be overloaded?

Answer: Java doesn't support operator overloading because it's just a choice made by its creators who wanted to keep the language more simple. Operator overloading allows you to do something extra than what for it is expected for. Java only allows arithmetic operations on elementary numeric types.

Following are some operator which cannot be overloaded in C or C++ language

1. **“.” : -** Member access or dot operator
2. **“? : ” :-** Ternary or conditional operator
3. **“::” :-** Scope resolution operator
4. **“.*”:-** Pointer to member operator
5. **“sizeof”:-** The object size operator
6. **“typeid”:-** Object type operator
7.

These operators cannot be overloaded because if we overload them it will make serious programming issues.

For an example the sizeof operator returns the size of the object or datatype as an operand. This is evaluated by the compiler. It cannot be evaluated during runtime. So we cannot overload it.

Q71. Explain the difference among structure and a class?

Ans : - A class is a user defined data type or a blueprint from which multiple objects can be created, a class can have data members, methods or functions, constructor, getters and setters. All in a single unit.

Structure- A structure is a collection of variable of different data types under a single unit.

Q 72. Explain the default access modifier in a class?

Ans : - Java compiler implicitly adds the default access modifier while defining a data type or methods. The scope of default access specifier is within the package. Means you can not access the default data members and methods outside the package.

Q 73. Can you list out the different type of constructors?

Ans: - There are basically three types of constructors:-

1. Default Constructor.
2. Zero parameter constructor.
3. Parametrized constructor.

Q 74. Explain a friend function?

Ans : - Java does not friend keyword like in C++, 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.

Q 75. Explain a ternary operator?

Ans : - The ternary operator is an operator that exists in some programming languages, which takes three operands rather than the typical one or two that most operators use. It provides a way to shorten a simple if else block.

Q 76. Do we require parameter for constructor?

Ans : - yes, if user wants to initialize the instance data members of a class with user define values then user requires a parameter constructor.

Q 77. Explain sealed Modifiers?

Ans:- A sealed class is a class or interface which restricts which other classes or interfaces may extend it. Sealed classes are also useful for creating secure hierarchies by decoupling accessibility from extensibility, allowing library developers to expose interfaces while still controlling all the implementations.

Q 78. Explain the difference among new and override?

Ans :- In java new keyword is used for dynamic memory allocation of an object, whereas override is an annotation keyword in java which tells the complier that the method or function is overridden form of parent class methods.

Q 79. How Can We Call The Base Method Without Creating An Instance?

Ans : - We can use static methods if we don’t want to create any instance of the class and want to access the methods directly, but static methods can access only static data members and functions.

Q 80. Define the various types of constructors?

Ans : - Types of constructors:-

1. Default constructor: - When user does not define any constructor, java compiler add the default constructor while creating a new object.
2. Zero parameter constructor: - this is a user define constructor, which has no arguments, this constructor is just like default constructor.
3. Parameterised constructor: - When programmer wants to initialize to instance data members with user defined values, then he can define its own constructor, which can and cannot have arguments, this type of constructor is call parameterized constructor.

Q81. Define Manipulators?

Answer : 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.

Q82. Can you give some examples of tokens?

Answer : Tokens are the smallest individual elements which are the building blocks of java program. Examples of tokens are kewords, literals, identifiers, operators and special symbols.

Explain structured programming and its disadvantage?

Answer : In structured programming design, programs are broken into different functions these functions are also known as modules, subprogram, subroutines and procedures.

Disadvantages:

A high level language has to be translated into the machine language by translator and thus a price in computer time is paid.

The object code generated by a translator might be inefficient compared to an equivalent assembly language program.

Q83. Explain the advantage of C++ being a block-structured language?

Answer: The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program.

- 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.

Q84. Can Struct be inherited?

Answer: In C++, 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.

Q85. When to use interface over abstract class.

Answer: When to use an abstract class:

- An abstract class is a good choice if we are using the inheritance concept since it provides a common base class implementation to derived classes.
- An abstract class is also good if we want to declare non-public members. In an interface, all methods must be public.
- If we want to add new methods in the future, then an abstract class is a better choice. Because if we add new methods to an interface, then all of the classes that already implemented that interface will have to be changed to implement the new methods.
- If we want to create multiple versions of our component, create an abstract class. Abstract classes provide a simple and easy way to version our components. By updating the base class, all inheriting classes are automatically updated with the change. Interfaces, on the other hand, cannot be changed once created. If a new version of an interface is required, we must create a whole new interface.

When to use an interface:

- If the functionality we are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated classes.
- 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.
- If we are designing small, concise bits of functionality, use interfaces. If we are designing large functional units, use an abstract class.

Q86. Explain a private constructor? Where will you use it?

Answer: A private constructor in Java is used in restricting object creation. It is a special instance constructor used in static member-only classes. If a constructor is declared as private, then its objects are only accessible from within the declared class. You cannot access its objects from outside the constructor class.

1. You can use it with static members-only classes.
2. You can use it with static utility or constant classes.
3. You can use it to serve singleton classes.
4. You can use it to assign a name, for instance, creation by utilising factory methods.
5. You can use it to prevent subclassing.

Q87. Can you override private virtual methods?

Answer: We cannot override private virtual methods, because any method even if it is virtual is declared private then it cannot be accessed inside other classes.

Q89. Can you allow class to be inherited, but prevent from being over-ridden?

Answer: Yes to do these we need to use final keywords on the method inside the class which we do not want to get inherited.

Q90.Why can't you specify accessibility modifiers for methods inside interface?

Answer: Because the methods in the interface are already public and abstract hence we don't need to specify the accessibility modifiers for methods inside interface.

91. Can static members use non static members? Give reasons.

Ans.

No. In static method, the method can access only static data members and static methods of another class or same cl ass but cannot access non-static methods and variables. It cannot access non-static data (instance variables).

92. Define different ways a method can be overloaded?

Ans.

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. Method overloading can be done by changing:

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.

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

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

Ans.

Yes we can have an abstract class without Abstract Methods as both are independent concepts. Declaring a class ab stract means that it can not be instantiated on its own and can only be sub classed. Declaring a method abstract mean s that Method will be defined in the subclass. An abstract class means that hiding the implementation and showing th e function definition to the user. An abstract class having both abstract methods and non-abstract methods. For an ab stract class, we are not able to create an object directly. But Indirectly we can create an object using the subclass obj ect. 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 d on'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.

94. Explain the default access modifier of 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. If you don't use any modifier, it is treated as default by def ault. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provi des more accessibility than private. But, it is more restrictive than protected, and public. When we don't use any key word explicitly, Java will set a default access to a given class, method or property. The default access modifier is als o called package-private, which means that all members are visible within the same package but aren't accessible fro m other packages: package com.

95. Can function overriding be explained in same class?

Ans.

While overriding, Both methods should be in two different classes and, these classes must be in an inheritance relat ion. 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 supercl ass. If you try to do so it raises a compile-time exception. If the super-class method throws certain exceptions, the m ethod in the sub-class should throw the same exception or its subtype (can leave without throwing any exception). T herefore, you cannot override two methods that exist in the same class, you can just overload them.

96. Does function overloading depends on Return Type?

Ans.

No, you cannot overload a method based on different return type but same argument type and number in java. In ov erloading it is must that the both methods have same name & different parameters (different type or, different numbe r or both). The return type doesn't matter. If they don't have different parameters, they both are still considered as sa me method and a compile time error will be generated.

Q97. Define different ways to declare an array?

Ans

You can declare an array in different methods, some are mentioned below:

```
int[] myIntArray = new int[3];

int[] myIntArray = {1, 2, 3};

int[] myIntArray = new int[]{1, 2, 3};

int num[] = {1, 2, 3, 4, 5};

int[] num = {1,2,3,4,5};

int num[] = new int[5];

int[] num = new int[5];

String[] myStringArray;

myStringArray = new String[]{"a", "b", "c"};
```

Q98 Can abstract class have a constructor?

Ans

Yes, an Abstract class always has a constructor. If you do not define your own constructor, the compiler will give a default constructor to the Abstract class. Constructor Chaining is a concept when a constructor calls another constructor in the same class or its base class. We know that Abstract classes can have constructors and we can not instantiate an abstract class. We can call the Abstract class's constructor through constructor chaining. den method.

Q99)Define rules of Function overloading and function overriding?

Ans:

Some rules of FUNCTION OVERLOADING are as mentioned below:

Two methods will be treated as overloaded

if both follow the mandatory rules below: Both must have the same method name.

Both must have different argument lists.

And if both methods follow the above mandatory rules, then they may or may not:

Have different return types.

Have different access modifiers.

Throw different checked or unchecked exceptions.

Some rules of FUNCTION OVERRIDING are as mentioned below:

With respect to the method it overrides, the overriding method must follow following mandatory rules:

It must have the same method name.

It must have the same arguments.

It must have the same return type. From Java 5 onward, the return type can also be a subclass (subclasses are a cova

riant type to their parents).

It must not have a more restrictive access modifier (if parent --> protected then child --> private is not allowed).

It must not throw new or broader checked exceptions.

And if both overriding methods follow the above mandatory rules, then they:

May have a less restrictive access modifier (if parent --> protected then child --> public is allowed).

May throw fewer or narrower checked exceptions or any unchecked exception.

Apart from the above rules, there are also some facts to keep in mind:

Only inherited methods can be overridden. That means methods can be overridden only in child classes.

Constructors and private methods are not inherited, so they cannot be overridden.

Abstract methods must be overridden by the first concrete (non-abstract) subclass.

final methods cannot be overridden.

A subclass can use super.overridden_method() to call the superclass version of an overrid

Q100) Explain the concept of Pure Virtual Functions?

Ans:

A pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived c lass if the derived class is not abstract. Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly. A subclass of an abstract class can only be instantiated directly if all inherited pure virtual m ethods have been implemented by that class or a parent class. Pure virtual methods typically have a declaration (sign ature) and no definition (implementation). Although pure virtual methods typically have no implementation in the cl ass that declares them, pure virtual methods in some languages (e.g. C++ and Python) are permitted to contain an im plementation in their declaring class, providing fallback or default behaviour that a derived class can delegate to, if a ppropriate Pure virtual functions can also be used where the method declarations are being used to define an interfac e

- similar to what the interface keyword in Java explicitly specifies. In such a use, derived classes will supply all im plementations.