INTERVIEW QUESTIONS for OBJECT-ORIENTED PROGRAMMING

1. Explain OOPs.

ANS: Object oriented programming is a computer programming model that organizes a program around its data rather than the code or function. In this model, data and the functions/methods that operate on them are bind together and instances of this combined unit, called objects, are created and implemented to program large and complex programs and also real world entities like inheritance, encapsulation, polymorphism, etc. Object oriented programming is based on the following four principles:-

- Abstraction

Abstraction is the property by virtue of which only the essential details are displayed to the users. It may also be defined as a process of hiding implementation details and exposes only the functionality to the user. Abstraction deals with ideas and not the events. In java, abstraction is achieved using interface(100%) and abstract classes(0-100%).

- Encapsulation

Encapsulation is the process of wrapping code and the data it manipulates together into a single unit and keeps both safe from outside interference and misuse. Access to the wrapped code and data is tightly controlled through a well-defined interface. In java, the basis of encapsulation is the class. The variable or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.

- Inheritance

Inheritance is the process by which one class/object acquires the properties and methods of another class/object. Code reusability is made possible by inheritance thus reducing development time and ensures a higher level of accuracy.

- Polymorphism

Polymorphism is the ability to perform a single action in different ways. Polymorphism allows different types of objects to pass through the same interface. Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object. There are two types of polymorphism in java:

a. Static/Compile-time Polymorphism
   When the complier is able to determine the actual function, it is called compile-time polymorphism. A call to a method is done at compile time. In java, we can achieved this by method overloading.
b. Dynamic/run-time Polymorphism

When the compiler is not able to determine the actual function and has to pass it on to be determined during rum time is called dynamic polymorphism. In java, it is achieved by method overriding.

When properly applied, polymorphism, encapsulation and inheritance combine to produce a programming environment that supports the development of far more robust and scalable programs than does the process oriented model.

2. Explain an abstraction. Real life example.

ANS:

Abstraction is one of the key concepts of object oriented programming languages. Its main goal is to handle complexity by hiding unnecessary details from the user. This

enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.
One simple real life example of abstraction is the mobile smart phone. In our day-to-day life, it is a generally accepted fact that smart phones have become an inevitable asset in our life. Irrespective of any age groups, everyone is well versed in using a smart phone. People don't need to know about the inner complex working of the smart phone which makes possible all the smart phone things that we can do. All the inner circuitry, microchip, complex architectures and various minute sub-elemental complex functionalities inside the smart phone remain hidden away from the smart phone user. The user just interacts with a simple interface that doesn't require any knowledge about the internal implementation.

3. Explain encapsulation. Real life examples.

ANS:

Encapsulation is to hide the members inside a class, preventing unauthorized parties to use.
One real life example of encapsulation is a driver knows how to start the car by pressing the start button and internal details of the starting operations are hidden from the driver. So we can say that the starting operation is encapsulated from the driver.

4. Explain the relationship among abstraction and encapsulation?

ANS:

There exist an indivisible correlation between abstraction and encapsulation. In oops, encapsulation is carried out to bind data and the methods operating on it and form a single unit thus making things more private and prohibiting unauthorized access from outside. And in the case of abstraction, we know that its aim is all about hiding complex or unnecessary details and expose only the relevant information. With a little speculation, we can see that abstraction can be achieved with encapsulation. Encapsulated data and methods and its working remain hidden from the user and are made available for access or its use only through some public members which at the same time abstraction is also achieved by hiding away the inner working of the methods and details of the data from the user.

5. Explain polymorphism.

ANS:

In object oriented programming, polymorphism is the characteristic of being able to assign a different meaning or usage to something in different context-specifically, to allow an entity such as a variable, a function or an object to have more than one form. Polymorphism in programming gives a program the ability to redefine methods for derived classes.

Imagine that we write a piece of code where we define a base class called shape. We may want to define specific methods such as area and perimeter that can be applied to our base class. But we cannot use the same method to compute the area and perimeter of both a circle and a rectangle. the area and perimeter of a circle depend on its radius and the area and perimeter of a rectangle depend on its width and length. Using polymorphism coding, we could define a circle and rectangle as sub-classes or derived classes of the base class shape. Each sub-class can have its own method for area and perimeter and each method will accept different parameters - length and width for rectangles, and radius for circles. Now the base

class shape has two derived classes - rectangle and circle - each with its own methods for area and perimeter. When we call one of the methods, the provided set of parameters can be used to determine which method should be called. If we provide a single parameter for the radius, our program can call the methods that belong to the class circle. If we provide two parameters for length and width, our program can call the methods that belong to the class rectangle.

6. Explain inheritance.

ANS:

Inheritance is one of the most important aspects of Object Oriented Programming (OOP). It provides code re-usability so in place of writing the same code, again and again, we can simply inherit the properties of one class into the other. Inheritance is the procedure in which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class. Along with the inherited properties and methods, a child class can have its own properties and methods. In java, The keyword extends is used by the sub class to inherit the features of super class.

There are broadly five forms of inheritance based on the involvement of parent and child classes.

- Single inheritance

This is a form of inheritance in which a class inherits only one parent class. This is the simplest form of inheritance and hence also referred to as simple inheritance.

- Multiple inheritance

An inheritance becomes multiple inheritances when a class inherits more than one parent class. The child class after inheriting properties from various parent classes has access to all of their objects.

- Multi-level Inheritance

A chain of inheritance is termed as multi-level inheritance. Here, one class inherits from the parent class and the newly created sub-class becomes the base class for another new class.

- Hybrid Inheritance

When there is a combination of more than one form of inheritance, it is known as hybrid inheritance.

7. How composition is better than inheritance?

ANS:

Both composition and inheritance promote code reuse through different approaches. Favouring composition over inheritance is a principle in object oriented programming and the reason can be known from the given below merits of composition over inheritance:-

- Inheritance is tightly coupled whereas composition is loosely coupled. One of Inheritance demerits is that it breaks encapsulation. If the subclass depends on the action of the superclass for its functions, when the super class behaviour changes, sub class functionality can be broken and modification will be then required in the sub class. This can be avoided if composition is used.
- There is no access control in inheritance whereas access can be restricted in composition. We expose all the superclass methods to the other classes having access to subclass. So if a new method is introduced or there are security holes in the superclass, subclass becomes vulnerable. Since in composition we choose which methods to use, it’s more secure than inheritance.
- Multiple inheritance in java can be done using composition but not with inheritance.

- Composition provides flexibility in invocation of methods that is useful with multiple subclass scenario.
- One more benefit of composition over inheritance is testing scope. Unit testing is easy in composition because we know what all methods we are using from another class. We can mock it up for testing whereas in inheritance we depend heavily on superclass and don't know what all methods of superclass will be used. So we will have to test all the methods of the superclass.

8. Which oops concept is used as a reuse mechanism?

ANS: Inheritance

9. Which oops concept exposes only the necessary information to the calling functions?

ANS: Abstraction

10. Explain a class. Create a class.

ANS:

In oops, a class is a user defined data type which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. It is a logical entity.

A class in java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

An example of creating a class named Fun

```
class Fun
{
//member data
int a;
String s;
double d;

//a zero constructor
Fun()
{}
Void run()
{
d=a+d;
System.out.println(d);
System.out.println(s);
}

}
```

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

ANS:

In the example class, class Fun, contains three member variables a,s and d of different data types and one method named run which operates on the member variables.

Here, the data variables and the method is encapsulated inside the class Fun. These encapsulated members will not be accessible from outside the class Fun without following some accessing protocols. If the method run of this class Fun is accessed by an authorised means, the user or caller of this method will simply be using it without any knowledge of the inside code that is contained inside the run method. Thus abstraction is also achieved.

12. Explain difference among class and object.

ANS:

A class is a logical entity , does not have any physical existence and does not occupy any memory. An object is an instance of class and is a physical as well as logical entity.

13. Define access modifier

ANS:

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it. There are four types of Java access modifiers:

1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. 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.
3. 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.
4. 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.

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

ANS:

An object is an instance of a class. It is an entity which has state and behaviour.

```
//creating an object of class Fun
Fun obj1=new Fun();
```

An object of class Fun is created and the object reference is stored in the reference variable obj1 of type Fun.

15. Give real life examples of object.

ANS:

One simple example of class and object is that of cars. Car can be considered as a class which have the general specifications and attributes that describes a car. Baleno, Audi, Mercedez, Cruz are all cars having various characters and attributes of their own. It can be said that they are the instances or object of class car.

16. Explain a Constructor.

ANS:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. 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. Constructors can also be overloaded like Java methods.

There are two rules defined for the constructor.

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

17. Define the various types of constructors.

ANS:

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.
The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.
A constructor which has a specific number of parameters is called a parameterized constructor. The parameterized constructor is used to provide different values to distinct objects. However, we can provide the same values also.

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

ANS: can used only through objects of the class containing the non-static members

19. Explain Destructor.

ANS:
In Java, when we create an object of the class it occupies some space in the memory (heap). If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming. To resolve this problem, we use the destructor. The destructor is the opposite of the constructor. The constructor is used to initialize objects while the destructor is used to delete or destroy the object. there is no concept of destructor in Java. In place of the destructor, Java provides the garbage collector that works the same as the destructor. The garbage collector is a program (thread) that runs on the JVM. It automatically deletes the unused objects (objects that are no longer used) and free-up the memory.

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

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 behavior 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.

22. Explain a friend function.

ANS:

If a function is defined as a friend function then the protected and private data of a class can be accessed using the function. Java does not have the friend keyword like c++, which is used to access the non-public members of a class. However ,we can simulate the friend functionality in java also.

23. Explain function Overloading.

ANS:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. Method overloading increases the readability of the program. There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

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

ANS:

Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

25. Write in brief linking of base class, sub class and base object, sub 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). It needs to be extended and its method implemented. It cannot be instantiated. It can have constructors and static methods also. It can have final methods which will force the subclass not to change the body of the method.

27. Explain operator overloading?

ANS:

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type.

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

ANS:

In programming languages, functions can be invoked in two ways: which is known as Call by Value and Call by Reference. Call by value method copies the value of an argument into the formal parameter of that function. Therefore, changes made to the parameter of the main function do not affect the argument. Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument.

29. Explain the super keyword?

ANS:

The super keyword in Java is a reference variable which is used to refer immediate parent class object. The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name. main usage of super class are:

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- 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. Rules to be followed to carry out method overriding in java are:

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

31. Difference among overloading and overriding?

ANS:

Overriding and overloading are the core concepts in Java programming. They are the ways to implement polymorphism in our Java programs. When the method signature (name and parameters) are the same in the superclass and the child class, it's called overriding. When two or more methods in the same class have the same name but different parameters, it's called overloading.

Method Overloading	Method Overriding
Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
Method overloading is performed within	Method overriding occurs in two classes that have IS-A (inheritance) relationship.

class.	
In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

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

ANS: only through objects.

33.

34.

35. Explain an interface?

ANS:

An interface in Java is a blueprint of a class. It is used to achieve abstraction and multiple inheritance in Java. There can be only abstract methods in the Java interface. It cannot be instantiated just like the abstract class. An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

36. Explain exception handling?

ANS:

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.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.

There are mainly two types of exceptions: checked and unchecked.

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

37. Explain the difference among structure and a class?

ANS:

Structure	Class
A structure is a grouping of variables of various data types referenced by the same name.	In C++, a class is defined as a collection of related variables and functions contained within a single structure.
If no access specifier is specified, all members are set to 'public'	If no access specifier is defined, all members are set to private.
Structure instance is called the structure variables	A class instance is called an object.
It does not support inheritance.	It supports inheritance.
Memory is allocated on the stack.	Memory is allocated on the heap.
Value type.	Reference type.
Grouping of data	Data abstraction and further inheritance
It is used for smaller amounts of data	It is used for a huge amount of data.
No null values	May have null values
May have only parameterized constructor	May have all types of constructors.

38. Explain the default access modifier in a class?

Ans:

The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If we do not specify any access level, it will be the default.

39. Explain a pure virtual function?

Ans:

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

definition of a pure virtual function. 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. An abstract method in java can be considered as a pure virtual function.

40. Explain dynamic or run time polymorphism?

ANS:

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time. 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.

41. Do we require a parameter for constructors?

ANS: No

42. Explain static and dynamic binding?

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

There are two types of binding

1. Static Binding (also known as Early 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.

2. Dynamic Binding (also known as Late Binding).

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

In general we can say that overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding.

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

ANS: Zero

44. Explain the default access specifiers in a class definition?
45.
46. Define the Benefits of Object Oriented Programming?

ANS: Benefits of object oriented programming are:

- Modular, scalable, extensible, reusable, and maintainable.
- It models the complex problem in a simple structure.

- Object can be used across the program.
- Code can be reused.
- We can easily modify, append code without affecting the other code blocs.
- Provides security through encapsulation and data hiding features.
- Beneficial to collaborative development in which a large project is divided into groups.
- Debugging is easy.

47. Explain method overloading?

ANS:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. Method overloading increases the readability of the program. There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

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

ANS:

Early Binding	Late Binding
The process of using the class information to resolve method calling that occurs at compile time is called early binding	The process of using the object to resolve method calling that occurs at run time is called late binding
It happens at compile time	It happens at run time
Also known as static binding	Also known as dynamic binding
overloading methods are bonded using early binding	Overridden methods are bonded using late binding
Execution speed is faster	Execution speed is comparatively slower.

49. Explain early binding? Give examples?

ANS: Connection of a method call to the method body ,which we termed as binding, that can be resolved at compile time by the compiler is known as early or static binding. If there is any private, final or static method in a class, there is static binding.

Examples of static binding:

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
```

```
  d1.eat();
 }
}
```

50. Explain loose coupling and tight coupling?

ANS: Tight coupling means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability of the code while Loose coupling means reducing the dependencies of a class that uses the different class directly.

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.

While, 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. It 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: tight coupling example:

```
class A {
  public int a = 0;
  public int getA() {
    System.out.println("getA() method");
    return a;
  }
  public void setA(int aa) {
    if(!(aa > 10))
      a = aa;
  }
}
public class B {
  public static void main(String[] args) {
    A aObject = new A();
     aObject.a = 100; // Not supposed to happen as defined by class A, this causes tight coupling.
    System.out.println("aObject.a value is: " + aObject.a);
```

```
  }
}
```

In the above example, the code that is defined by this kind of implementation uses tight coupling and is error prone since class B knows about the detail of class A, if class A changes the variable 'a' to private then class B breaks, also class A's implementation states that variable 'a' should not be more than 10 but as we can see there is no way to enforce such a rule as we can go directly to the variable and change its state to whatever value we decide.

```
class A {
  private int a = 0;
  public int getA() {
    System.out.println("getA() method");
    return a;
  }
  public void setA(int aa) {
    if(!(aa > 10))
      a = aa;
  }
}
public class B {
  public static void main(String[] args) {
    A aObject = new A();
    aObject.setA(100); // No way to set 'a' to such value as this method call will
                       // fail due to its enforced rule.
    System.out.println("aObject value is: " + aObject.getA());
  }
}
```

In the above example, the code that is defined by this kind of implementation uses loose coupling and is recommended since class B has to go through class A to get its state where rules are enforced. If class A is changed internally, class B will not break as it uses only class A as a way of communication.

52. Write in brief abstract class.

ANS:

53. Define the Benefits of oops over pop?

ANS:

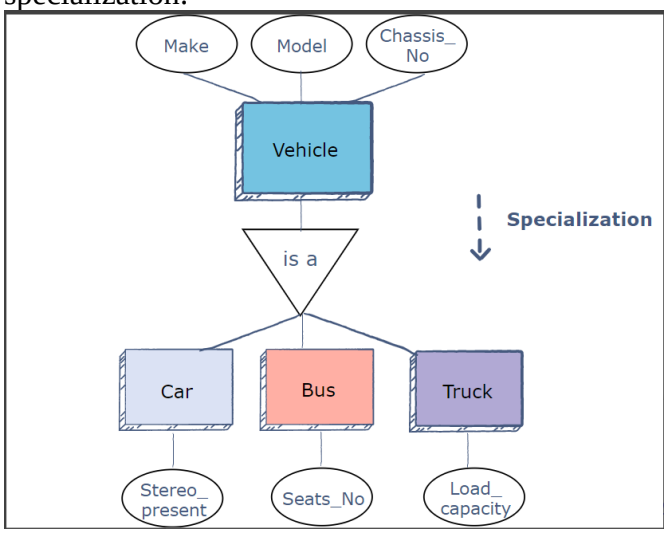
On the basis of	Pop	oops
Security	Comparatively less secure	More secure because data hiding is possible due to abastraction
Access modifier	No access modifier	There are access modifier
Inheritance	No inheritance	There is

Code reusability	No	Yes
Overloading	Not possible	Yes
Virtual class	No	Yes
Complex problem	Not appropriate for complex problems	Yes
Code management	Difficult to manage if code grows as project size grows	Can manage easily

54. Explain Generalization and Specialization?

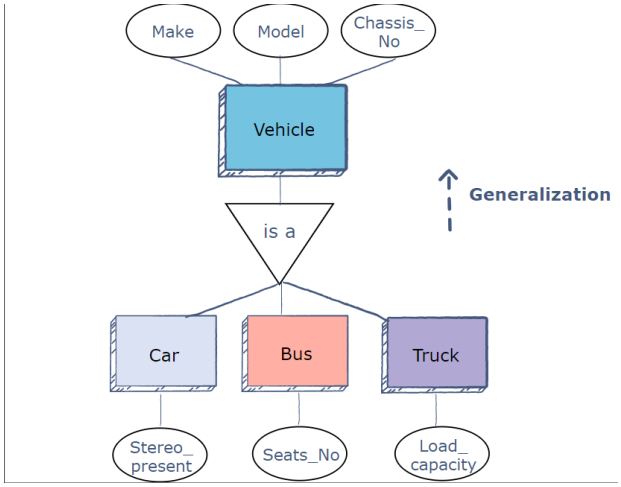
ANS:

Specialization and generalization are fundamental concepts in database modeling that are useful for establishing superclass-subclass relationships. Specialization is a top-down approach in which a higher-level entity is divided into multiple specialized lower-level entities. In addition to sharing the attributes of the higher-level entity, these lower-level entities have specific attributes of their own. Specialization is usually used to find subsets of an entity that has a few different or additional attributes. The following enhanced entity relationship diagram expresses the entities in a hierarchical database to demonstrate specialization:



Generalization is a bottom-up approach in which multiple lower-level entities are combined to form a single higher-level entity. Generalization is usually used to find common attributes among entities to form a generalized entity. It can also be thought of as the opposite of specialization.

The following enhanced entity relationship diagram expresses entities in a hierarchical database to demonstrate generalization:



55. Write in brief Association, Aggregation and Composition?

ANS:

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.
The composition and aggregation are two subsets of association. In both of the cases, the object of one class is owned by the object of another class; the only difference is that in composition, the child does not exist independently of its parent, whereas in aggregation, the child is not dependent on its parent i.e., standalone. An aggregation is a special form of association, and composition is the special form of 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. 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.

56. Write in brief Object Composition vs. Inheritance
ANS:

Basis of comparison	Composition	Inheritance
Functionality	We can evaluate the functioning of the classes that we are using without having to be concerned with determining whether or not they are the parent or the child classes.	It is not possible to test a child class without first testing its parent class.
Code reusability	Because of composition, it is possible to reuse the code even in the final classes.	Inheritance cannot be used to extend the functionality of the final class.
Significance	Simply declaring a type that	In inheritance, we are

	we want to use in composition allows it to store several implementations, each of which can have unique behaviors depending on the context of when they are called.	responsible for defining the class that will become our "superclass," and this definition is immutable once it has been used.
Class Combining	Composition gives users the flexibility to mix features and capabilities from multiple classes into a single entity.	Java does not support multiple inheritances, which means that several classes cannot be extended.
Relationship	It is a "has-a" kind of circumstance.	It's a case of "is-a."

57. Explain cohesion.

ANS:

Cohesion is the principle of Object-Oriented programming that defines how the classes are designed. Cohesion is closely related to ensuring that the purpose for which a class is getting created in oops is well-focused and single. In other words, the more closely related stuff is grouped in a class, the higher will be the cohesiveness. We need cohesiveness because it is easy to work in a highly cohesive structure. The update and modification in the code become easy.

For example, in a library, books of biology are placed together, and books of mathematics are placed together. In a library, we do not shuffle books of mathematics and biology. Because it becomes difficult to find a mathematics or biology book. Similarly, we do the categorization of stuff in the programming world also. Shuffling of unrelated stuff is avoided in the code. In fact, it is impossible to work on a code where cohesiveness within the class is ignored.

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

ANS:

Object composition and inheritance are two techniques for reusing functionality in object-oriented systems. Class inheritance, or subclassing, allows a subclass' implementation to be defined in terms of the parent class' implementation. This type of reuse is often called white-box reuse. This term refers to the fact that with inheritance, the parent class implementation is often visible to the subclasses. Object composition is a different method of reusing functionality. Objects are composed to achieve more complex functionality. This approach requires that the objects have well-defined interfaces since the internals of the objects are unknown. Because objects are treated only as "black boxes," this type of reuse is often called black-box reuse.

Each of these two methods have advantages and disadvantages. The advantage of class inheritance is that it is done statically at compile-time and is easy to use. The disadvantage of

class inheritance is that the subclass becomes dependent on the parent class implementation. This makes it harder to reuse the subclass, especially if part of the inherited implementation is no longer desirable. One way around this problem is to only inherit from abstract classes. Another problem with class inheritance is that the implementation inherited from a parent class cannot be changed at run-time. n object composition, functionality is acquired dynamically at run-time by objects collecting references to other objects. The advantage of this approach is that implementations can be replaced at run-time. This is possible because objects are accessed only through their interfaces, so one object can be replaced with another just as long as they have the same type. In addition, since each object is defined in terms of object interfaces, there are less implementation dependencies. The disadvantage of object composition is that the behavior of the system may be harder to understand just by looking at the source code. A system using object composition may be very dynamic in nature so it may require running the system to get a deeper understanding of how the different objects cooperate.

In general, object composition should be favored over inheritance. It promotes smaller, more focused classes and smaller inheritance hierarchies. Most designers overuse inheritance, resulting in large inheritance hierarchies that can become hard to deal with. A design based on object composition will have less classes, but more objects.

59. Explain “this”.

ANS:

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object. Six usage of this keyword in java are:

- this can be used to refer current class instance variable.

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

- this can be used to invoke current class method (implicitly)

We may invoke the method of the current class by using the this keyword. If we don't use the this keyword, compiler automatically adds this keyword while invoking the method.

- this() can be used to invoke current class constructor.

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

- this can be passed as an argument in the method call.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

- this can be passed as argument in the constructor call.

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

- this can be used to return the current class instance from the method.

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

60. Write in brief static member and member functions.

ANS:

Variables and methods declared using keyword static are called static members of a class. 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 variables are also called class variables because they can be accessed using class name. Static variables occupy single location in the memory. While declaration, there is no need to initialize the static variables explicitly because they take default values like other non-static variables.

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. Generally, static methods are used to perform some common tasks for all objects of a class. For example, a method returns a random number. It does not matters which instance use this method, it will always behave exactly in this way. In other words, static methods are used when the behavior of a method has no dependency on the values of instance variables.

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

ANS:

62. Explain the Diamond problem?

ANS:

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. 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. Java does not support the multiple inheritance because of the diamond problem. Multiple inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one

parent class. The feature creates a problem when there exist methods with the same name and signature in both the super-class and sub-class. When we call the method, the compiler gets confused and cannot determine which class method to be called and even on calling which class method gets the priority. This is known as diamond problem. The solution to the diamond problem is default methods and interfaces.

63. Explain the solution for diamond problem?

ANS:

The solution to the diamond problem is default methods and interfaces. 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. we

can have same default methods (same name and signature) in two different interfaces and, from a class you can implement these two 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. Therefore, we can achieve multiple inheritance by using the interface. It is also the solution to the diamond problem.

64. Explain the need of abstract class?

ANS: one of the main aim of oops, Abstraction ,is a process of hiding the implementation details and showing only functionality to the user. Abstraction can be achieved with either abstract classes or interfaces. 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. Abstract classes also have the advantage of allowing better forward compatibility. Once clients use an interface, we cannot change it; if they use an abstract class, we can still add behavior without breaking the existing code.

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

66. Can abstract class have constructors?

ANS: yes

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

ANS: zero.

68. Which keyword can be used for overloading?

ANS:

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

Ans:

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.

70. Define all the operators that cannot be overloaded?

ANS: The list of operators which cannot be overloaded is as follows:

- Conditional or Ternary Operator (?:) cannot be overloaded.
- Size of Operator (sizeof) cannot be overloaded.
- Scope Resolution Operator (::) cannot be overloaded.
- Class member selector Operator (.) cannot be overloaded.
- Member pointer selector Operator (.*) cannot be overloaded.
- Object type Operator (typeid) cannot be overloaded.

71. Explain the difference among structure and a class?

Ans: 37.

72. Explain the default access modifier in a class?

ANS:

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

ANS: 17

74. Explain a ternary operator?

ANS: 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 operator decides which value will be assigned to the variable. It is the only conditional operator that accepts three operands. It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

The ternary operator ?: in Java is the only operator that accepts three operands:

booleanExpression ? expression1 : expression2

The very first operand must be a boolean expression, and the second and third operands can be any expression that returns some value. The ternary construct returns expression1 as an output if the first operand evaluates to true, expression2 otherwise.

75. Do We Require Parameter For Constructors?

ANS: No

76. Explain Sealed Modifiers?

ANS:

77. Explain The Difference Between New And Override.

ANS:

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

ANS: if we were to call the base method without creating an instance, we can do it by making the method static .Another way is to inherit from that class containing the method we want to access without creating an instance.

79. Define The Various Types Of Constructors?

ANS:

80. Define Manipulators?

ANS:

81. Can you give some examples of tokens?

ANS: a token is a single element of a programming language. There are five categories of tokens: 1) constants, 2) identifiers, 3) operators, 4) separators, and 5) reserved words.

In Java, the program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation. These statements and expressions are made up of tokens. In other words, we can say that the expression and

statement is a set of tokens. The tokens are the small building blocks of a Java program that are meaningful to the Java compiler.

82. Explain structured programming and its disadvantage?

ANS: In structured programming design, programs are broken into different functions these functions are also known as modules, subprogram, subroutines and procedures. Each function is design to do a specific task with its own data and logic. Information can be passed from one function to another function through parameters. A function can have local data that cannot be accessed outside the function's scope. The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written.

The following are the disadvantages of structured programming:

- It is machine-independent so converting it to machine code takes time.
- The program is dependent on variables such as data types. As a result, it must be kept up to date with the changing needs.
- The converted machine code differs from the assembly language code.
- This approach is language-dependent, so the development usually takes longer. In the case of assembly language, development takes less time because it is pre-programmed for the machine.

83. When to use interface over abstract class.

ANS: Interface is preferred over abstract class for the following situations:

- If the functionality we are creating will be useful across a wide range of differente objects, an interface implementation is better. 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 also a good choice when we want to have something similar to multiple inheritances since we can implement multiple interfaces.
- If we are going to design the small, concise bits of functionality, then we must use interfaces. But if we are designing the large functional units, then we must use an abstract class.

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

85. Can you override private virtual methods

Ans:

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

ANS: yes .

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

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

89. Define different ways a method can be overloaded?

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

ANS: yes

91. Explain the default access modifier of a class?

92. . Can function overriding be explained in same class?

ANS: No, there must be an IS-A relationship.

93. Does function overloading depends on Return Type?

ANS: No

94. . Can abstract class have a constructor?

ANS: Yes

95. Define rules of Function overloading and function overriding?

ANS: 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).

Rules of overloading a method in Java

- Method to be overloaded and the overloading methods must have same names.

- we must set different method signature for the methods. the method signature is made of a number of arguments, types of arguments, and order of arguments if they are of different types. we can change any of these or combinations of them to overload a method in Java.

- We need to remember in java, method overloading is not possible by changing the return type of the method only.