1) Explain OOPS ?

- Object oriented programming (OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes.
- Object oriented programming refers to language that that use objects in programming, they use object as a primary source to implement what is to happen in code.
- Object oriented programming is a core of java programming, which is used for designing a program using classes and objects.
- OOPs implement real-world entities like inheritance, polymorphism ,hiding, etc into programming. It also allows binding data and code together.

2) Explain an abstraction? Real life example.

- Data abstraction is very important feature of OOPs that allows to displaying important information & hiding the implementation details.
- Data abstraction is the property by virtue of which only the essential details are displayed to the user. The non-essential units are not displayed to the user.
- E.g. -While riding a bike, we know that if we raise the accelerator, the speed will increase but we don't know how it actually happens.
  This is data abstraction se the implementation details are hidden from rider.

3) Explain encapsulation? Real life example.

- Encapsulation refers to binding data & code together in a single unit.
- In other words, Encapsulation is a programming technique that binds the class members (variable & methods) together and prevents them from being accessed by other classes.
- E.g. – Suppose you have an account in bank. If your balance variable is declared as a public variable in bank software, your account balance will be known as public, In this case anyone can know your account balance.
  So they declare balance variable as private for making your account safe, so that anyone cannot see your account balance.

4) Explain the relationship among abstraction and encapsulation?

- Encapsulation is data hiding & abstraction is implementation hiding.
- Data abstraction can be achieved by using encapsulation.
- Abstraction hides complexity by giving you a more abstract picture, while encapsulation hides internal work so that you can change it later.
- Abstraction is a method of hiding the unwanted information. Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.

5) Explain polymorphism?

- Polymorphism means the ability of object to have many forms is called as polymorphism. Means to exist in multiple forms.
- A concept by which we can perform a single action in different ways.
- It occurs when we have many classes that are related to each other by inheritance.
- Multiple definitions can be given to a single interface.
- For example, If you have a class name vehicle, it can have a method named speed but you cannot define it because different vehicles have different speed. This method will be defined in the subclasses with different definitions for different vehicles.

6) Explain Inheritance?

- Inheritance means one class i.e child/sub is deriving all properties from another class i.e parent/Base class.
- When we inherit from an existing class, we can reuse methods & fields of parent class. And additionally we can add new methods in our child class but they can not access by parent class.
- Means child class has all properties of parent class & also their own properties as well.
- For example, If there is a class such as ‘Vehicle’, other class like ‘Car’, ‘Bike’, etc. can inherit common properties from the Vehicle class.

<u>7) How composition is better than inheritance?</u>

- **<u>Composition -</u>** The composition is a java design way of implementing a **has-a** relationship. The composition is accposition is accomplished by the use of an instance variable that refers to other objects.
- Java does not support multiple inheritance is one reason for composition is better over inheritance in java. Since you can extend one class in java, but if you need multiple features, such as reading and writing character data into a file, you need reader writer functionality. It makes your job simple to have them as private members, and this is called composition.
- Composition offers better test-ability of a class than interface.
- If you use composition, you are flexible enough to replace the better and updated version of the composed class implementation.

<u>8) Which OOPS concept is used as a reuse mechanism?</u>

- Inheritance is a concept used as a reuse mechanism.
- In inheritance we inherit the base class, then we can reuse methods and fields of the parent class in child class.

<u>9) Which OOPS concept exposes only the necessary information to the calling functions?</u>

- Data hiding is a technique used in OOPs to expose only the necessary information to the calling function.
- Data hiding means hiding the internal data within the class to prevent its direct access from outside classes.
- Data hiding is achieved by using the private access specifier.

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

Ans :

Data hiding is the concept of oops which means exposing only necessary information to clients. Data hiding is a technique used in object-oriented programming. It means hiding the internal details.

- Data hiding is a technique used in object-oriented programming.
- It means hiding the internal details.
- Data hiding makes sure that the internal details are restricted to class members.
- Data integrity is maintained in data hiding.
- Data hiding reduces the complexities and increases the robustness.

https://www.javatpoint.com/what-is-data-hiding

Q.10 Explain a class? Create a class.

Ans :

1. Class is a set of object which shares common characteristics/ behavior and common properties/ attributes.

2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.

3. Class does not occupy memory.

4. Class is a group of variables of different data types and group of methods.

A class in java can contain:

• data member

• method

• constructor

• nested class and

• interface

class Student

{

```
        int id;                 //data member (also instance variable)

        String name;                //data member (also instance variable)


        public static void main(String args[])

        {

                Student s1=new Student();           //creating an object of
Student

                System.out.println(s1.id);

                System.out.println(s1.name);

        }

}
```

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

Ans :

They both seem very similar but totally different in concept and implementation.

The **major difference between abstraction and encapsulation** is that abstraction hides the code complexity while encapsulation hides the internal working from the outside world. In this section, we will discuss abstraction and encapsulation and the **differences between abstraction and encapsulation in Java**.

Abstraction

It is a feature of OOPs. It is used to hide the unnecessary information or data from the user but shows the essential data that is useful for the user. It can be achieved by using the interface

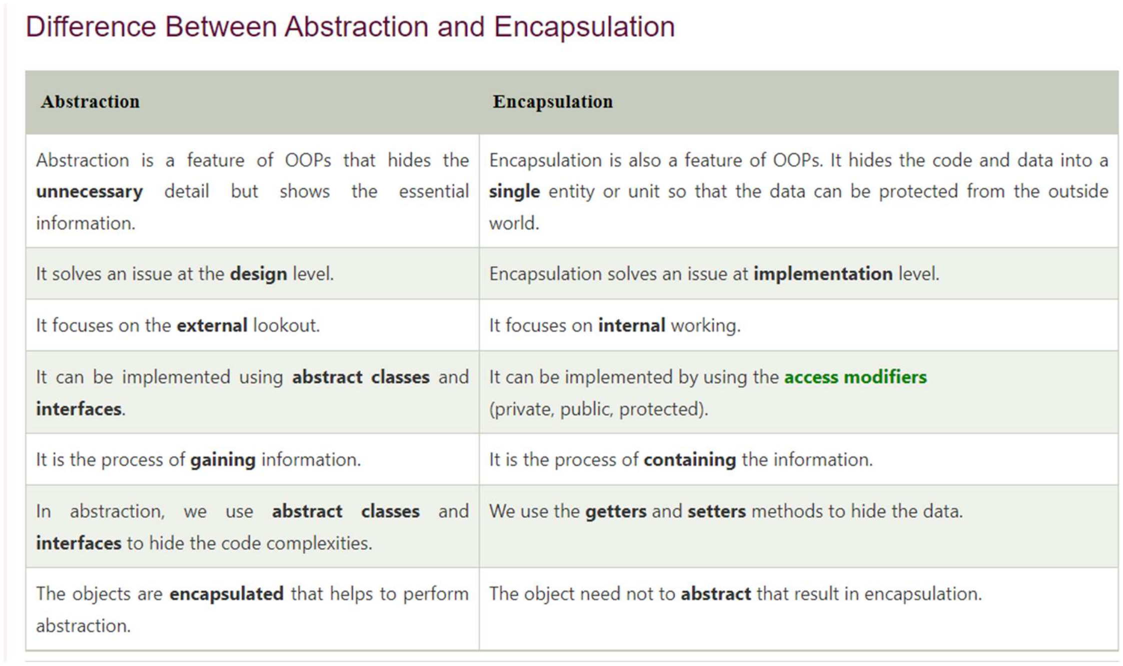
and the abstract class

. In interfaces, only the methods are exposed to the end-user. The best example of abstraction is a **TV remote**. The user only interacts with the outer interface that is nothing but keys. The user only knows which key to press for what function.

Encapsulation

It is also a feature of OOP. It is used to bind up the data into a single unit called class. It provides the mechanism which is known as **data hiding**. It is an important feature of OOPs. It prevents to access data members from the outside of the class. It is also necessary from the security point of view.

https://www.javatpoint.com/abstraction-vs-encapsulation-in-java

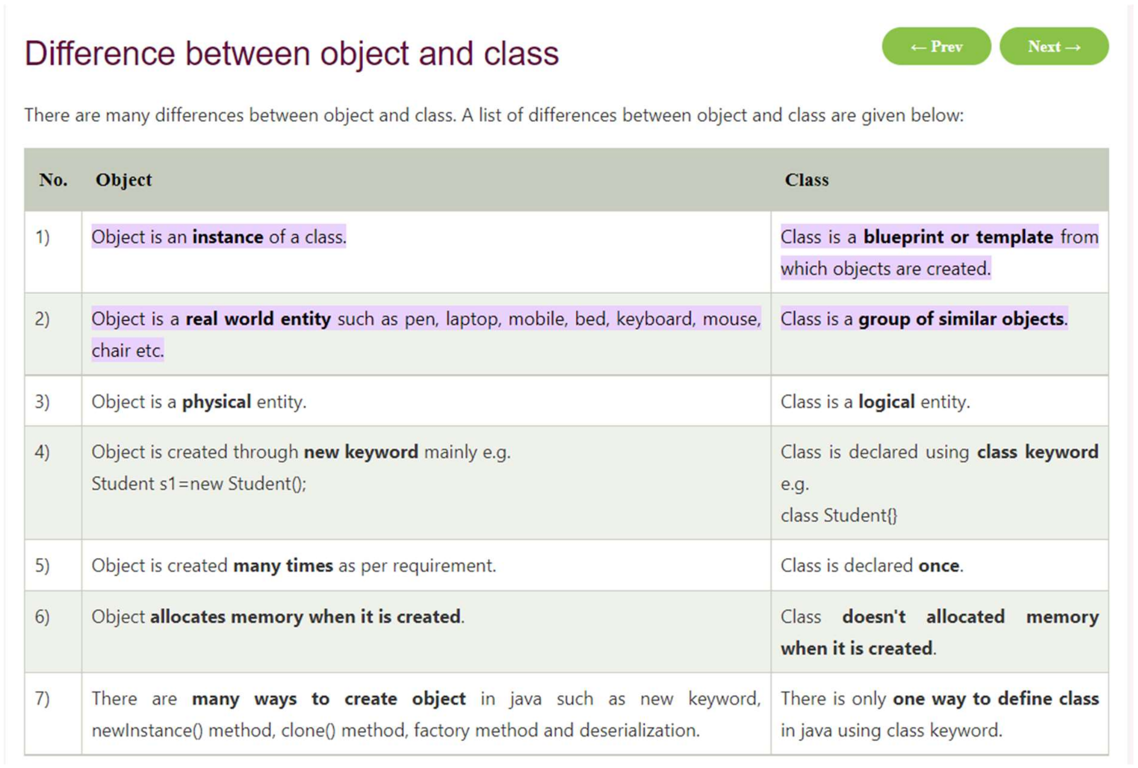


Difference Between Abstraction and Encapsulation

Abstraction	Encapsulation
Abstraction is a feature of OOPs that hides the **unnecessary** detail but shows the essential information.	Encapsulation is also a feature of OOPs. It hides the code and data into a **single** entity or unit so that the data can be protected from the outside world.
It solves an issue at the **design** level.	Encapsulation solves an issue at **implementation** level.
It focuses on the **external** lookout.	It focuses on **internal** working.
It can be implemented using **abstract classes** and **interfaces**.	It can be implemented by using the **access modifiers** (private, public, protected).
It is the process of **gaining** information.	It is the process of **containing** the information.
In abstraction, we use **abstract classes** and **interfaces** to hide the code complexities.	We use the **getters** and **setters** methods to hide the data.
The objects are **encapsulated** that helps to perform abstraction.	The object need not to **abstract** that result in encapsulation.

Q.12 Explain difference among class and object?

Ans :



Difference between object and class

There are many differences between object and class. A list of differences between object and class are given below:

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

Let's see some real life example of class and object in java to understand the difference well:

Class: Human **Object:** Man, Woman

Class: Fruit **Object:** Apple, Banana, Mango, Guava wtc.

Class: Mobile phone **Object:** iPhone, Samsung, Moto

Class: Food **Object:** Pizza, Burger, Samosa

Q.13 Define access modifiers ?

Ans :

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

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.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

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

Ans :

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

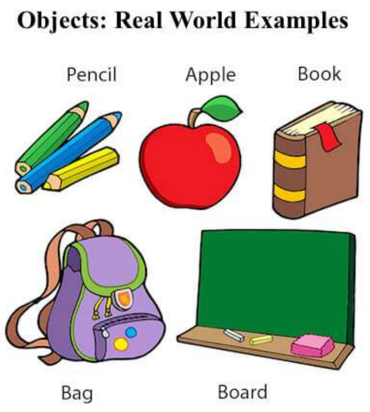
- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
    //defining fields
    int id;//field or data member or instance variable
    String name;
    //creating main method inside the Student class
```

```
        public static void main(String args[]){
                //Creating an object or instance
                Student s1=new Student();//creating an object of Student
                //Printing values of the object
                System.out.println(s1.id);//accessing member through refer
               ence variable
                System.out.println(s1.name);
        }
    }
```

Q.15 Give real life examples of object.

Ans :
The 10 real life examples of class and object are as follows:



- 1. Class - Human being
- Man - an object of Human being
- Woman - an object of Human being

- 2. Class - colour
- Red - an object of colour
- Blue - an object of colour

- 3. Class - pets
- Dog - an object of pets
- Cat - an object of pets

- 4. Class - Wild animals
- Tiger - an object of Wild animals
- Lion - an object of Wild animals

- 5. Class - drinks
- Hot drink - an object of drinks
- Cold drink - an object of drinks

- 6. Class - pen
- Ballpoint pen - an object of pen
- Ink pen - an object of pen

- 7. Class - water
- Freshwater - an object of water
- Rainwater - an object of water

- 8. Class - tea
- Green - an object of tea
- Black - an object of tea

- 9. Class - food
- Lunch - an object of food
- Dinner - an object of food

- 10. Class - coffee
- Filter coffee - an object of coffee
- Instant coffee - an object of coffee

Q.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. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

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

Q.17 Define the various types of constructors ?

Ans :

Types of Java constructors

There are two types of constructors in Java:

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

Java Default Constructor

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

Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized construc
tor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
```

```
        }
        //method to display the values
        void display(){System.out.println(id+" "+name);}

        public static void main(String args[]){
        //creating objects and passing values
        Student4 s1 = new Student4(111,"Karan");
        Student4 s2 = new Student4(222,"Aryan");
        //calling method to display the values of object
        s1.display();
        s2.display();
       }
    }
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

Q.18 Whether static method can use nonstatic members ?

Ans :

#1: Accessing members and methods

A static method can only access static data members and static methods of another class or same class but cannot access non-static methods and variables. Also, a static method can rewrite the values of any static data member.

A non-static method can access static data members and static methods as well as non-static members and methods of another class or same class, also can change the values of any static data member

#2: Calling process

The memory of a static method is fixed in the ram, for this reason, we don’t need the object of a class in which the static method is defined to call the static method. To call the method we need to write the class name followed by the name of the method

The memory of the non-static method is not fixed in the ram, so we need a class object to call a non-static method. To call the method we need to write the name of the method followed by the class object name

#3: Binding process

In the static method, the method use compile-time or early binding. For this reason, we can access the static method without creating an instance. In a non-static method, the method use runtime or dynamic binding. So that we cannot access a non-static method without creating an instance.

#4: Overriding

In the static method, we cannot override a static method, because of early binding.

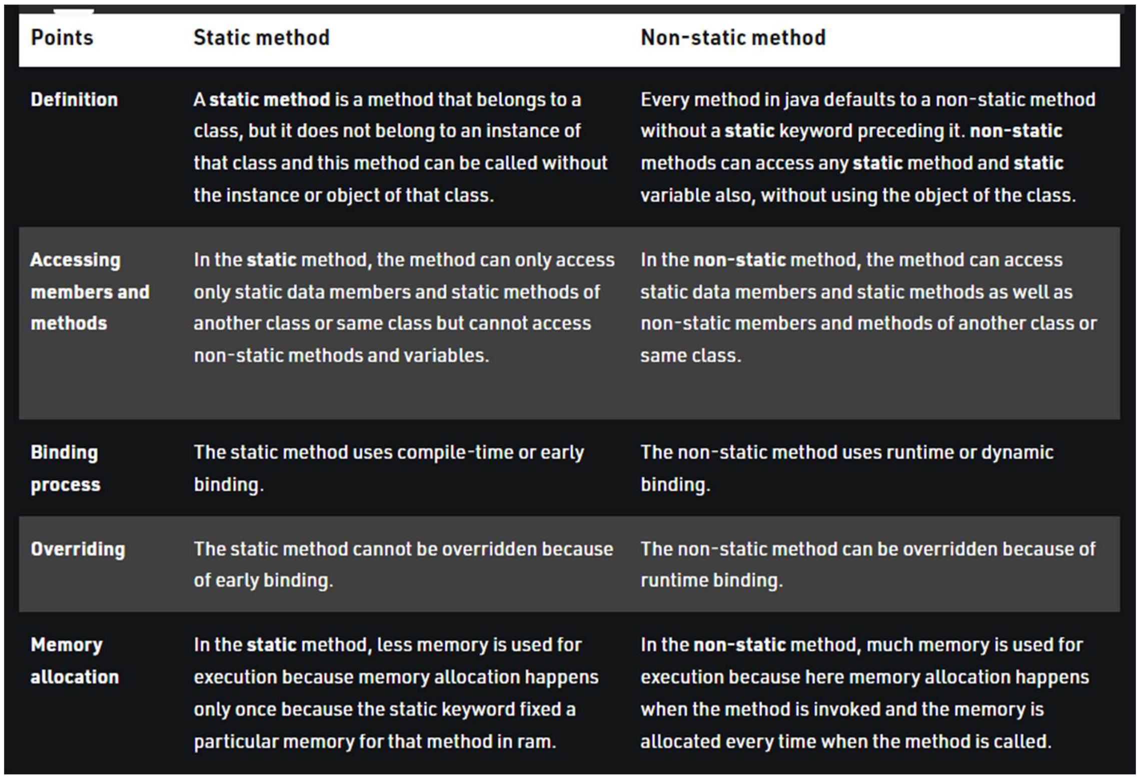
In the non-static method, we can override a non-static method. Because for override we need runtime polymorphism, which happens only in runtime binding.

#5: Memory allocation

In the static method, memory allocation happens only once, because the static keyword fixed a particular memory for that method in ram. So when the method is called every time in a program, each time that particular memory is used. For that reason, less memory is allocated.

In the non-static method, here memory allocation happens when the method is invoked and the memory is allocated every time when the method is called. So much memory is used here.

https://www.geeksforgeeks.org/difference-between-static-and-non-static-method-in-java/#:~:text=A%20static%20method%20can%20only,of%20any%20static%20data%20member.



Points	Static method	Non-static method
Definition	A **static method** is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class.	Every method in java defaults to a non-static method without a **static** keyword preceding it. **non-static** methods can access any **static** method and **static** variable also, without using the object of the class.
Accessing members and methods	In the **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.	In the **non-static** method, the method can access static data members and static methods as well as non-static members and methods of another class or same class.
Binding process	The static method uses compile-time or early binding.	The non-static method uses runtime or dynamic binding.
Overriding	The static method cannot be overridden because of early binding.	The non-static method can be overridden because of runtime binding.
Memory allocation	In the **static** method, less memory is used for execution because memory allocation happens only once because the static keyword fixed a particular memory for that method in ram.	In the **non-static** method, much memory is used for execution because here memory allocation happens when the method is invoked and the memory is allocated every time when the method is called.

Q19. What is Destructor ?

Ans ➔ A destructor is a member Function that is invoked automatically

When the object goes out of Scope . destructor is the last function that is going to be called before an object Is Destroyed.

➔Destructor is also special member function like Constructor.

➔Destructor has the same name as their class name preceded by a tilde (~) symbol.

➔It is not possible to define more than one destructor.

➔Destructor neither requires any argument nor returns any value.

➔Destructor release memory space occupied by the objects created by constructor.

Syntax:
Syntax for defining the destructor within the class

```
~ <class-name>()
{
}
```

Q20.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.

➔ If make a function as inline, then the compiler replaces the function calling location with the definition of the inline function at compile time.

Syntax for an inline function:

```
inline return_type function_name(parameters)
{
  // function code
}
```

Q21.Explain a Virtual Function?

Ans ➔ A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

➔ Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

➔ They are mainly used to achieve Runtime polymorphism.

➔ Functions are declared with a **virtual** keyword in base class.

➔ Virtual functions cannot be static.

➔ A class may have virtual destructor but it cannot have a virtual constructor.

➔ Virtual functions must be members of some class.

Q22.Explain a Friend Function?

Ans➔ A friend function in C++ is defined as a function that can access **private, protected** and **public** members of a class.

➔The friend function is declared using the **friend keyword** inside the body of the class.

➔ By using the keyword **friend** compiler knows the given function is a friend function.

➔ **Declaration of a friend function**

```
class <class_name>
{
   friend  <return_type>  <function_name>(argument/s);
 };
```

Q23.Explain function Overloading?

Ans➔ If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

➔ If we have to perform only one operation, having same name of the methods increases the readability of the program.

➔ Advantage of method overloading

Method overloading increases the readability of the program.

➔ **Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type

For example:

```
void func()
{
 }
void func(int a)
 {
 }
```

Q24.Explain base class , sub class , super class?

Ans➔ Super Class/Parent Class/Base Class: A base class is a class in [Object-Oriented Programming language](), from which other classes are derived.

➔ It is also known as a superclass or parent class.

➔It cannot inherit properties and methods of Derived Class.

➔ **Syntax:**

```
 Class base_classname
{
 }
```

➔**Sub Class/Child Class/Derived Class:** A class that is created from an existing class. The derived class inherits all members and member functions of a base class.

➔ The derived class can have more functionality with respect to the Base class and can easily access the Base class. A Derived class is also called a **child class** or **subclass**.

➔**Syntax :**

```
 Class derived_classname : access_mode base_class_name{
}
```

Q25.Write in brief linking of base class, sub class and base object, sub object.

Ans➔ BaseClass: A reference variable of a **Baseclass** can be used to a refer any subclass object derived from that superclass. If the methods are present in **BaseClass**, but overridden by **SubClass**, it will be the overridden method that will be executed.

➔ A subclass reference can be used to refer its object.

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

➔ So difference between referencing using superclass reference and referencing using subclass reference is use superclass referencing can holds object of subclass and could only access the methods which are defined/overridden by subclass while use subclass referencing can not hold object of superclass and could access the methods of both superclass and subclass.

Q26.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).

➔An abstract class have Zero or More Abstract method.

➔If there is any abstract method inside a class ,then that class is also considered as abstract class & in that case it become mandatory to declare that class Abstract.

➔ 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.

Syntax➔

abstract class Demo{

//Code

}

Q27.Explain Operator Overloding?

Ans➔ Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

➔ In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

➔ For operator overloading to work, at least one of the operands must be a user-defined class object.

➔ Operators that can be overloaded
We can overload

- Unary operators
- Binary operators
- Special operators ([ ], () etc)

➔ But, among them, there are some operators that cannot be overloaded. They are

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

➔**Java doesn't supports operator overloading** because it's just a choice made by its creators who wanted to keep the language more simple.

28. define different types of arguments? (call by value/call by reference)

- **what is call by value..?**

in this particular parameter passing method, the values of the actual parameters copy into the function's formal parameters. it stores both types of parameters in different memory locations. thus, if one makes any changes inside the function- it does not show on the caller's actual parameters.

this method passes a copy of an actual argument to the formal argument of any called function. in the case of a call by value method, any changes or alteration made to the formal arguments in a called function does not affect the overall values of an actual argument. thus, all the actual arguments stay safe, and no accidental modification occurs to them.

- **what is call by reference ?**

in this case, both the formal and actual parameters refer to a similar location. it means that if one makes any changes inside the function, it gets reflected in the caller's actual parameters.

this method passes the address or location of the actual arguments to the formal arguments of any called function. it means that by accessing the actual argument's addresses, one can easily alter them from within the called function. thus, in call by reference, it is possible to make alterations to the actual arguments. thus, the code needs to handle the arguments very carefully. or else, there might be unexpected results and accidental errors.

29. explain the super keyword?

the super keyword **refers to superclass (parent) objects**. it is used to call superclass methods, and to access the superclass constructor. the most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name

30. explain method overriding?

if subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

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

usage of java method overriding

- method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- method overriding is used for runtime polymorphism

rules for java method overriding

1. the method must have the same name as in the parent class
2. the method must have the same parameter as in the parent class.
3. there must be an is-a relationship (inheritance).

31. difference among overloading and overriding?

the differences between method overloading and method overriding in java are as follows:

method overloading	method overriding
method overloading is a compile-time polymorphism.	method overriding is a run-time polymorphism.
it helps to increase the readability of the program.	it is used to grant the specific implementation of the method

method overloading	method overriding
	which is already provided by its parent class or superclass.
it occurs within the class.	it is performed in two classes with inheritance relationships.
method overloading may or may not require inheritance.	method overriding always needs inheritance.
in method overloading, methods must have the same name and different signatures.	in method overriding, methods must have the same name and same signature.
in method overloading, the return type can or can not be the same, but we just have to change the parameter.	in method overriding, the return type must be the same or co-variant.
static binding is being used for overloaded methods.	dynamic binding is being used for overriding methods.
poor performance due to compile time polymorphism.	it gives better performance. the reason behind this is that the binding of overridden methods is being done at runtime.
private and final methods can be overloaded.	private and final methods can’t be overridden.
argument list should be different while doing method overloading.	argument list should be same in method overriding.

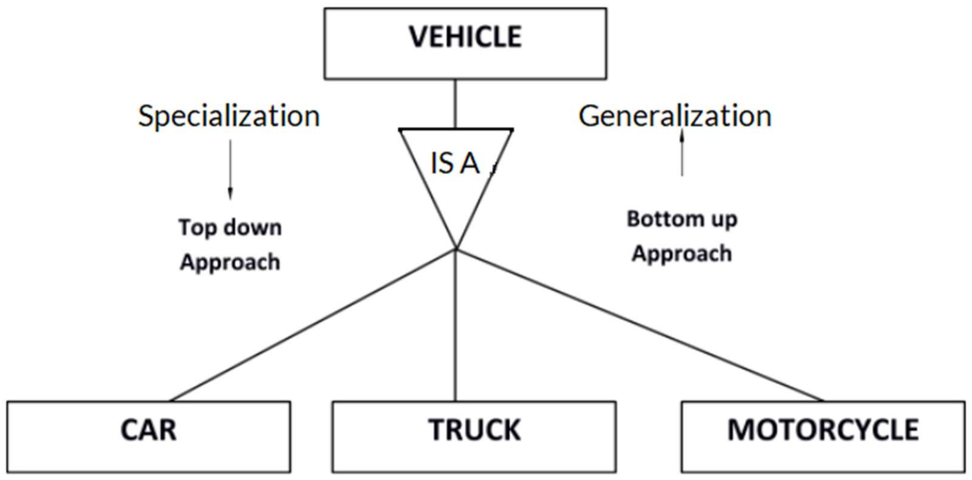
32. whether static method can use non-static members?

a static method can only access static data members and static methods of another class or same class but **cannot access non-static methods and variables**

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

subclasses

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



car, truck and motorcycle are all subclasses of the superclass vehicle. they all inherit common attributes from vehicle such as speed, colour etc. while they have different attributes also i.e number of wheels in car is 4 while in motorcycle is 2.

superclasses

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

in the above example, vehicle is the superclass and its subclasses are car, truck and motorcycle.

inheritance

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

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

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

there are two approaches to refer a subclass object. both have some advantages/disadvantages over the other. the declaration affect is seen on methods that are visible at compile-time.

1. **first approach (referencing using superclass reference):** a reference variable of a superclass can be used to a refer any subclass object derived from that superclass. if the methods are present in superclass, but overridden by subclass, it will be the overridden method that will be executed.
2. **second approach (referencing using subclass reference) :** a subclass reference can be used to refer its object.

35. explain an interface?

in general, an interface is **a device or a system that unrelated entities use to interact**.

implementing an interface allows a class to become more formal about the behavior it promises to provide. interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. if your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

36. explain exception handling?

in computing and computer programming, exception handling is **the process of responding to the occurrence of exceptions – anomalous or exceptional conditions requiring special processing – during the execution of a program**.

exception handling is **the process of responding to unwanted or unexpected events when a computer program runs**. exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

37. Explain the difference among structure and a class

Classes DEFAULT to having private members. Structures DEFAULT to having public members.

Structures are values type. Classes are reference type.

Structure stores in memory via stack. Classes stored in memory via heap.

Structure doesn't support inheritance. Classes support inheritance.

Constructor works in different way.

'new' operator works in different way.

Allocating memory for structure is very fast because this takes place inline or on the stack.

38. Explain the default access modifier in a class?

There are two types of modifiers in Java: access modifiers and non-access modifiers.

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:

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

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

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

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

Similarly a default method or variable is also accessible inside the package in which they are defined and not outside the package.

39. Explain a pure virtual function?

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 as these methods can be used to achieve polymorphism.

A virtual function for which we are not required implementation is considered as pure virtual function. For example, Abstract method in Java is a pure virtual function.

The Virtual Function is an ordinary function in Java.

We are not required to declare any explicit description to define the virtual function.

In Java, no virtual keyword is used to define the virtual function.

The Parent class pointer is used to refer to the object of the child class

the virtual function should be defined in the child class with the same name in the parent class.

All the instance methods in Java are considered the Virtual function except final, static, and private methods.

40. Explain dynamic or run time polymorphism?

Purpose of Polymorphism in OOPs

The primary purpose of polymorphism is to perform a single action in multiple ways. In other words, polymorphism provides one interface with many implementations. Poly means many and morphs means forms. True to its name, polymorphism offers different forms to a single function.

Runtime Polymorphism in Java

Runtime polymorphism, also known as the Dynamic Method Dispatch, is a process that resolves a call to an overridden method at runtime. The process involves the use of the reference variable of a superclass to call for an overridden method.

The method to be called will be determined based on the object being referred to by the reference variable.

Benefits of Runtime Polymorphism in Java

The primary advantage of runtime polymorphism is enabling the class to offer its specification to another inherited method. This implementation transfer from one method to another can be done without altering or changing the codes of the parent class object. Also, the call to an overridden method can be resolved dynamically during runtime.

41. Do we require a parameter for constructors?

Not necessarily, we can have a constructor without parameter or with multiple parameters.

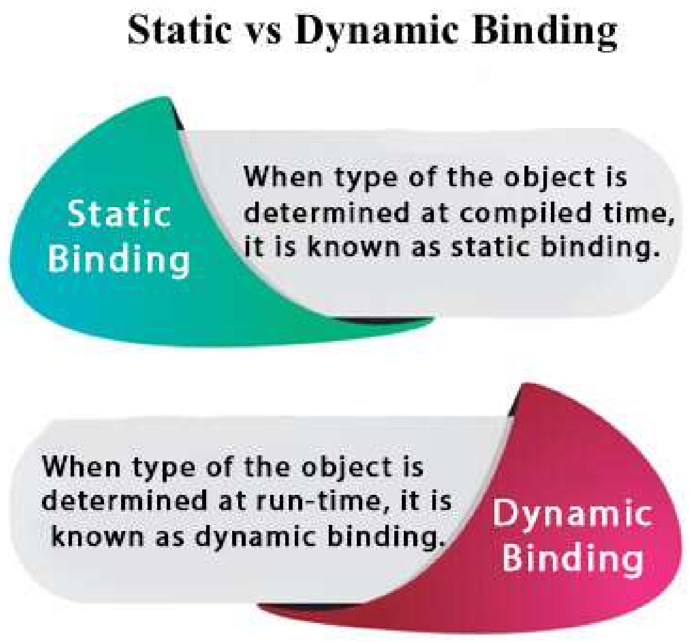
42. Explain static and dynamic binding?

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

There are two types of binding

Static Binding (also known as Early Binding).

Dynamic Binding (also known as Late Binding).


Static vs Dynamic Binding
Static Binding
When type of the object is determined at compiled time, it is known as static binding.
When type of the object is determined at run-time, it is known as dynamic binding.
Dynamic Binding

Static binding is better performance-wise because java compiler knows that all such methods cannot be overridden and will always be accessed by object reference variable.

Hence, the compiler doesn’t have any difficulty in binding between a method call and method definition. That’s why binding for such methods is always static.

The binding which occurs during runtime is called dynamic binding in java. This binding is resolved based on the type of object at runtime. In dynamic binding, the actual object is used for binding at runtime.

Dynamic binding is also called late binding or runtime binding because binding occurs during runtime. An example of dynamic binding is *method overriding.*

Difference between Static and Dynamic Binding in Java

There are the following differences between static and dynamic binding. They are as follows:

1. Static binding occurs at compile-time while dynamic binding occurs at runtime.

2. In static binding, actual object is not used whereas, actual object is used in the dynamic binding.

3. Static binding is resolved at compile time whereas, dynamic binding is resolved at runtime.

4. Static binding is also called early binding because it happens during compilation whereas, dynamic binding is called late binding because it happens during runtime.

5. An example of static binding is method overloading whereas, the example of dynamic binding is method overriding.

6. Private, static, and final methods show static binding because they cannot be overridden whereas, except private, static, and final methods, other methods show dynamic binding because they can be overridden.

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

zero

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

<u>44. Explain the default access specifiers in a class definition?</u>

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

I can't really think of any better real life example than FACEBOOK post.

When you update your Facebook Status, it gives you 4 options

1. If you make this status visible for Public, anyone can see this status on Internet (Anyone On or Off Facebook). [Public Access Specifiers]

If you make this status visible "only for me". No one can see this status except you. [Private Access Specifiers]

If you make this status visible for Friends or Friends of friends, then your status will be only visible for your friends and your friends friends. Not everyone presents on Facebook or Internet. [Protected Access Specifiers]

If you make this status visible for "Friends", then your status will be only visible for your Friends. Not your friends friends or everyone presents on Facebook. [Default Access Specifiers]

In Programming Terms:

There are four Access Specifiers in Java.

1. Public
2. Private
3. Default.
4. Protected

45. Which OOPS concept is used as reuse mechanism

Inheritance is the feature that provides a reuse mechanism.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

Terms used in Inheritance

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

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

When we define class, data members, and member function without specifying any access modifier, it has a default access modifier by default. The class, data members, and member functions that we defined without using access modifiers can be accessed only within the same package. Outside of the current package, we cannot use the classes and methods.

The meaning of both the access specifiers and the access modifiers is the same. There are no differences between the specifiers and modifiers, and the use of both is the same. The access modifier is an official term and the new term that we use instead of modifier is specifier. So, default, public, protected, and private access modifiers can also be referred to as default, public, protected, and private access specifiers.

45. Which OOPS concept is used as reuse mechanism?

Inheritance is the object-oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called the superclass and the object that inherits the superclass is called a subclass.

46. Define the Benefits of Object-Oriented Programming?

OOP has a lot of benefits over procedural programming.

Here are just a few: OOP allows you to better represent the real world in your programs. Just about everything in the world can be described as a set of properties and actions, which is exactly how objects are organized. By better representing the world, OOP programs can better solve real problems. OOP programs are easier to read and understand. Because you don't have to write out the properties and actions for each character or sprite, your programs will be much shorter. This makes them easier to read and understand.

It can be faster to program in OOP. OOP makes it easy to reuse objects in other programs. Instead of creating code from scratch, you can use an existing object or method, and change it to fit your program. It's easier to create large programs. Because OOP helps you eliminate unnecessary code, it's easier to create larger, more complex programs with OOP.

OOP programs are easier to modify and maintain. With OOP, it's easy to make changes to existing objects. You can also use existing objects to create new objects with minor modifications.

Because objects are self-contained, OOP programs are easier to debug.

OOP makes your programs easier to write, maintain, and reuse because of things like objects, classes, properties, and methods.

47. Explain method overloading?

Method overloading is a concept that allows to declare multiple methods with same name but different parameters in the same class.

Java supports method overloading and always occur in the same class (unlike method overriding). Method overloading is one of the ways through which java supports polymorphism. Polymorphism is a concept of object-oriented programming that deal with multiple forms. We will cover polymorphism in separate topics later on.

Method overloading can be done by changing number of arguments or by changing the data type of arguments.

If two or more method have same name and same parameter list but differs in return type cannot be overloaded.

Note: Overloaded method can have different access modifiers and it does not have any significance in method overloading.

There are two different ways of method overloading.

- Different datatype of arguments
- Different number of arguments

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

Early Binding

It is a compile-time process

The method definition and method call are linked during the compile time.

Actual object is not used for binding.

For example: Method overloading Program execution is faster

Late Binding

It is a run-time process

The method definition and method call are linked during the run time.

Actual object is used for binding

For example: Method overriding Program execution is slower

49. Explain early binding? Give examples?

Binding generally refers to a mapping of one thing to another. In the context of compiled programming languages, binding is the association of a method call with the method definition. In simpler terms, when a method is called in Java, the program control binds to the memory address where that method is defined.

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

• The normal method calls and overloaded method calls are examples of early binding.

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

• Since all information needed to call a method is available before run time, early binding results in faster execution of a program

50.Explain loose coupling and tight coupling?

Coupling is nothing but the dependency of one class on the other. If one object in a code uses the other object in the program, it is called loose coupling in Java. In coupling, two classes or objects collaborate and work with each other to complete a pre-defined task. It simply means that one element requires another element to complete a function.

1) Loose Coupling in Java:

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. Here, the parent object is rarely using the object, and the object can be easily changed from external sources. The loose coupling in Java has the edge over tight coupling as it reduces code maintenance and efforts. A change in one class does not require changes in the other class, and two classes can work independently.

2) 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.

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

```
 // tight coupling concept
 class Volume
 {
 public static void main(String args[])
 {
 Box b = new Box(5,5,5);
 System.out.println(b.volume);
 }
}
 class Box {
 public int volume;
 Box(int length, int width, int height) {
 this.volume = length * width * height;
 }
 }
```

Output: 125

Explanation: In the above example, there is a strong inter-dependency between both the classes. If there is any change in Box class then they reflect in the result of Class Volume.

```
// loose coupling concept
class Volume {
 public static void main(String args[]) {
```

```
 Box b = new Box(5,5,5);
 System.out.println(b.getVolume());
 }
 }
 final class Box {
private int volume;
 Box(int length, int width, int height) {
 this.volume = length * width * height;
 }
 public int getVolume() {
 return volume;
 }
 }
```

Output: 125

Explanation: In the above program, there is no dependency between both the classes. If we change anything in the Box classes then we don’t have to change anything in Volume class

52. Write in brief abstract class.

Hiding internal details and showing only the functionality is known as abstraction. Abstraction is one of the key concepts in Object Oriented Programming Paradigm. In order to implement Abstraction, we use Abstract Classes or Interfaces. Abstract class in Java is a collection of abstract and non-abstract methods. Abstract methods do not have a body or implementation. The implementation for the abstract methods is provided by the child classes which extend the abstract class.

Basically, an Abstract Class is nothing but a blueprint for the child class. Abstract classes justify the layout of your idea and does not really implement them.

Syntax:
```
abstract class ClassName {
 // Abstract and Non-abstract methods
 }
```

53. Define the Benefits of oops over pop?

• OOP makes development and maintenance easier.

• OOP provides data hiding.

• OOP provide ability to simulate real-world event much more effectively.

• OOP is faster and easier to execute

• OOP provides a clear structure for the programs

• OOP helps to keep the code DRY "Don't Repeat Yourself".

• OOP makes it possible to create full reusable applications with less code and shorter development time.

• Modularity for easier troubleshooting.

• OOP gives flexibility through polymorphism.

• OOP gives better productivity.

Q.54 Explain Generalization and Speacialization ?

Ans➔ Generalization

Converting a subclass type into a superclass type is called 'Generalization'

because we are making the subclass to become more general and its scope is widening. This is also called widening or up casting. Widening is safe because the classes will become more general.

For example-: if we say Car is a Vehicle, there will be no objection.

Thus Java compiler will not ask for cast operator in generalization.

Specialization

Converting a super class type into a sub class type is called 'Specialization'.

Here, we are coming down from more general form to a specific form and hence the scope is narrowed. Hence, this is called narrowing or down-casting.

Narrowing is not safe because the classes will become more and more specific thus giving rise to more and more doubts.

For example:

if we say Vehicle is a Car we need a proof. Thus, In this case, Java compiler specifically asks for the casting. This is called explicit casting.

Q.55) Write **in brief Association , Aggregation and Composition?**

---> Relationship between two classes can be of following type:

1) Inheritance
2) Association
   --> Aggregation
   --> Composition

Inheritance --> Where we extend one class with another class to use its properties.

--> Tightly coupled. (If changes are made in properties of any class both the classes will be affected as all properties inherits).

```
class Vehicle
{ }
class Car extends
{  }
```

(IS-A relationship → Car **IS-A** Vehicle)

Association --> The relation between two objects is known as association.

Using reference variable or new keyword.

Encapsulation and data hiding can be achieved.

```
class Engine{ }
class Car
{
        Engine e = new Engine ();
}
```

(HAS-A relationship → Car **HAS-A** Engine)

--> No tightly coupled.

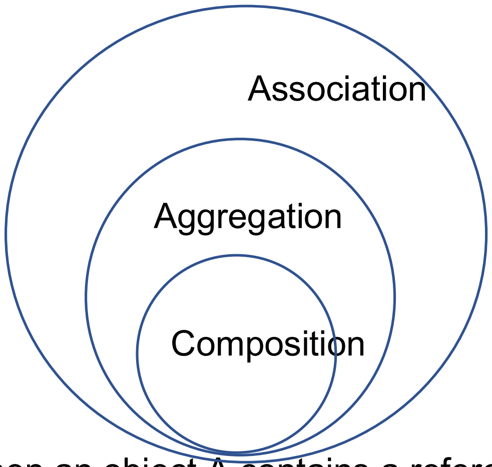
Advantages --> 1) Code reusability.

2) Cost cutting.

3) Reduce redundancy. (removing unnecessary codes)

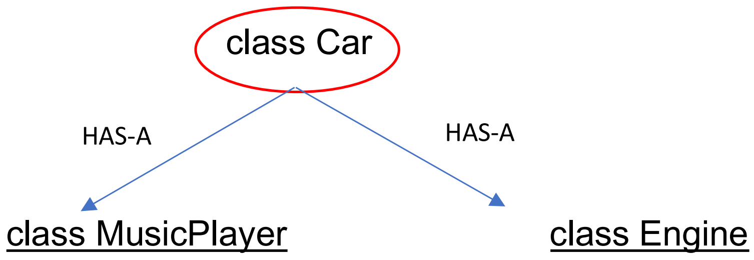
Relation between Association , Aggregation , Composition.

(Association has two types based upon their existence or bonding.)



Aggregation --> When an object A contains a reference to another object B and their classes can exists independently then it is termed as Aggregation.

Composition --> When an object A contains a reference to another object B and their classes cannot exists independently then it is termed as Composition.



- Here , class Car and class MusicPlayer can exists independently. i.e Car can exists without a MusicPlayer and a MusicPlayer can exists without a Car. Hence it is a weak bonding. This is called as **Aggregation.**

- Now class Car and class Engine cannot exists independently. i.e Car must have a engine and Engine is of no use if its not used in car. Hence it is a strong bonding. This is called as **Composition.**

Q.56) Write in brief Object Composition vs Inheritance.

-->

Inheritance	Composition
> In inheritance a class is **derived** from another class.	> In composition , a class is **sum** of its parts from other classes.
> **extends** keyword is used here. class B extends class A then B will have access to all data from class A.	> reference variable or **new** keyword is used to access data from other classes using objects.
> **IS-A** relationship Car is a vehicle.	> **HAS-A** relationship Car has a engine.
> **tight coupling** changing base class can cause unwanted side effects on its subclasses or even all over the codebase .	> **loose coupling** it brings more flexibility and and allow us to change runtime behavior.
> **no access control** in inheritance	> **access can be restricted** in composition.
> class inheritance is defined at **compile time.**	> object composition is defined dynamically at **run time.**
> **exposes** both public and protected members of the base class.	> the internal details are **not exposed** to each other and they interact through their public interfaces.
> it often **breaks encapsulation** as the parents details are exposed to the subclasses.	> objects are accessed solely through interfaces hence not all details are exposed,so it **won't break encapsulation.**

Q.57) Explain Cohesion ?

-->

Cohesion in Java is the object oriented principle most closely associated with making sure that a class is designed with a single , well-focused purpose.

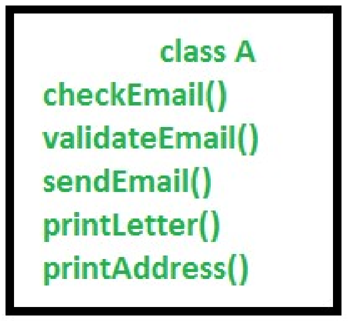


Fig: Low cohesion

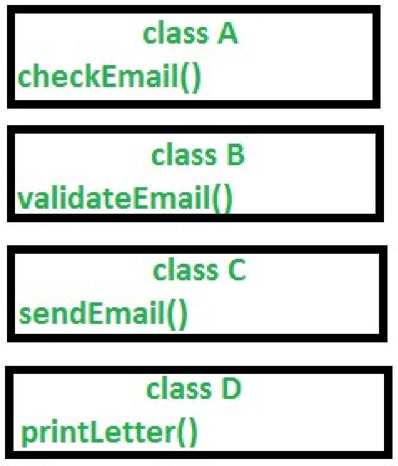


Fig: High cohesion

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

Two types of Cohesion:

1) Low cohesion :- When class is designed to perform many different tasks instead of focusing on any specific task then that class is called "Low Cohesive class". This kind of approach is the bad programming design approach. It required a lot of modifications for small change.

2) High cohesion :- When class is designed to perform any specific task then that class is called as "High Cohesive class". This kind of approach is good programming design approach. It can easily maintain and less modifiable.

Advantages : 1) much easier to maintain.

2) more reusable than other classes.

Q58)Explain “black-box-reuse” and “white box reuse” ?

-->

Black-Box reuse means that you use component without knowing it's internals. All you have is a component interface.

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

Black box reuse is using a class/function/code unmodified in a different project

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

White-box:

pros:

- simple (very natural concept)
- you have more control over things

cons:

- requires intrinsic knowledge on component internals
- can be difficult to implement (OO inheritance constraints) sometimes it leads to broken\incorrect inheritance chains

Black-box:

pros:

- low coupling (gives late binding and other goodies)

cons:

- not obvious (code is much harder to understand)
- interfaces are more fragile than classes (i.e. interfaces vs inheritance)

Q59)Explain “this”.
-->

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 parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

this can also be used to:

- used to refer current class instance variable
- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

Q60)Write in brief static member and member functions.
-->
In Java, static members are those which belongs to the class and you can access these members without instantiating the class.

The static keyword can be used with methods, fields, classes (inner/nested), blocks.

Static Methods − You can create a static method by using the keyword *static*. Static methods can access only static fields, methods. To access static methods there is no need to instantiate the class, you can do it just using the class name.

Static Fields − You can create a static field by using the keyword static. The static fields have the same value in all the instances of the class. These are created and initialized when the class is loaded for the first time. Just like static methods you can access static fields using the class name (without instantiation).

Static Blocks − These are a block of codes with a static keyword. In general, these are used to initialize the static members. JVM executes static blocks before the main method at the time of class loading.

Important points about Static :

* A static member is shared by all objects of the class , all static data is initialized to zero when the first object is created , if no other initialization is present.
* A static member function can only access static data member , other static member functions and any other functions from outside the class.
* By declaring a function member as static , we make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name.

Q61)How will you relate unrelated classes or how will you achieve polymorphism without the base classes?

-->

relating unrelated classes?

-->I thinks answer lies in the question in which we had to make one(BOOKS) class as member of other class (Library)- also creating object in that.

How we achieve polymorphism is by function Overriding? Right

But for function overriding u need a parent class/base class which is being inherited by child class/derived class.

So question is how u will achieve it without base class?

Answer is within a class without any inheritance or function over ridding we can achieve it by function overloading.

Same function names with different parameters and doing separate jobs. One form can do may types of work.

#Using interfaces

--> Achieving polymorphism without using base classes.

```
public interface Foo {
  public int a();
}

public class A implements Foo {
  public int a() {
    return 5;
  }
}



public class B implements Foo {
  public int a() {
    return 6;
  }
}
```

Q62)Explain the diamond problem?

-->

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. But remember that **Java does not support the multiple inheritance** because of the diamond problem.

What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**.

It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPP's languages. It is sometimes referred to as the **deadly diamond of death**.

Q63)Explain the solution for the diamond problem?

-->

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

The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.

The advantage of interfaces is that it can have the same default methods with the same name and signature in two different interfaces. It allows us to implement these two interfaces, from a class. We must override the default methods explicitly with its interface name.

64.**Explain the need of abstract class?**

Ans:

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.

65. Why can't we instantiate abstract class?

Ans:

An abstract class is a class which doesnt have an implementation for one or more methods. It means that the jvm doesnt have any direction of what to do in case if someone calls the method which is abstract. So if you 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 you 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, an abstract class can have a constructor in Java. You can either explicitly provide a constructor to the abstract class or if you don't, the compiler will add a default constructor of no argument in the abstract class. This is true for all classes and it also applies to an abstract class. For those who want to recall what is an abstract class in Java, it's a class that can not be instantiated with the new() operator or any other way. In order to use an abstract class in Java, You need to extend it and provide a concrete class. An abstract class is commonly used to define a base class for a type hierarchy with default implementation, which is applicable to all child classes.

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

Ans:

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

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

Ans:

Using Super Keyword:

- If both parent & child classes have the same method, then the child class would override the method available in its parent class. By using the super keyword we can take advantage of both classes (child and parent) to achieve this. We create an object of child class as it can inherit the parent class methods.
- In the child class method, we call the method available in its parent class by using super().

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

Ans :

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

70. Define all the operators that cannot be overloaded?

Ans :

Java doesn't support operator overloading. Java doesn't provide freedom to programmers, to overload the standard arithmetic operators e.g. +, -, * and / etc.

71. Explain the difference among structure and a class?

Ans:

the structure is the same as the class with some differences. Security is the most important thing for both structure and class. A structure is not safe because it could not hide its implementation details from the end-user, whereas a class is secure as it may hide its programming and design details

72. Explain the default access modifier in a class?

Ans:

1. 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.

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

Ans:

There are two types of constructors in Java:

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

73. Explain a ternary operator?

Ans:

The conditional operator or ternary operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

S.No	Operator & Description
1	? : (Conditional) If Condition is true? Then value X: Otherwise value Y

74. Do We Require Parameter For Constructors?

Ans:

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

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

Ans:

1. Default constructor (no-arg constructor): -

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

2. Parameterized constructor: -

   A constructor which has a specific number of parameters is called a parameterized constructor.

74. Explain a ternary operator?

Ans:

1. In Java, the **ternary operator** is a type of Java conditional operator.
2. It is used to evaluate Boolean expressions.
3. The operator decides which value will be assigned to the variable.
4. It is the only conditional operator that accepts three operands.
5. It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.

75. Do We Require Parameter for Constructors?

Ans:

It is not necessary to have a constructor block in your class definition. If you don't explicitly write a constructor, the compiler automatically inserts one for you.

76. Explain Sealed Modifiers?

Ans:

We can say that the class that cannot be inherited but can be instantiated is known as the sealed class. It allows classes and interfaces to have more control over their permitted subtypes. It is useful both for general domain modelling and for building a more secure platform for libraries.

77. Explain the Difference Between New And Override.

Ans:

Override**:** Override implements Polymorphism. Which says I want to use the same method but in various classes with the same signature of methods.

New: The new keyword hides the definition of the base class method and gives a new definition in the derived class.

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

Ans:

Yes, it is possible,

1. If it is a static method.
2. By inheriting from that class.
3. From derived classes using base keyword.

79. Define the Various Types of Constructors?

Ans:

1. Default constructor (no-arg constructor): -

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

2. Parameterized constructor: -

   A constructor which has a specific number of parameters is called a parameterized constructor.

80. Define Manipulators?

Ans:

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

81. Can you give some examples of tokens?

Ans:

Java token includes the following:

1. **Keywords**:

   Examples: abstract, Boolean, byte, break, class, catch , char, this, try, etc.

2. **Identifiers:**

Examples: Identifiers are used to name a variable, constant, function, class, and array

3. **Literals:**

   It can be categorized as an integer literal, string literal, Boolean literal, etc.

4. **Operators:**

   Arithmetic Operators, Assignment Operators, Relational Operators, etc.

5. **Separators:**

   [], {}, (), = , ; , . ,

6. **Comments:** //, /*

82. Explain structured programming and its disadvantage?

Ans - Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:
C, C++, Java etc.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, so it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore, it needs to be updated with the need on the go.
4. Usually, the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly

language, the development takes lesser time as it is fixed for the machine.

83. When to use interface over abstract class?
Ans-

- 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.

84. Explain a private constructor? Where will you use it
Ans- Java allows us to declare a constructor as private. We can declare a constructor private by using the private access specifier. Note that if a constructor is declared private, we are not able to create an object of the class.

- It can be used with static members-only classes.
- It can be used with static utility or constant classes.
- It can be used to assign a name, for instance, creation by utilizing factory methods.
- It is also used to avoid sub-classing.

85. Can you override private virtual methods?

Ans- A private virtual method cannot be overridden as it can’t be accessed outside the class.

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

Ans- We can use the sealed modifier on a method or property that overrides a virtual method or property in a base class. This enables you to allow classes to derive from your class and prevent them from overriding specific virtual methods or properties.

About Sealed Keyword: A method modified by the "sealed" keyword is known as a sealed method. A sealed keyword is used for a method to prevent it from being overridden in the derived class, i.e to prevent the runtime polymorphic feature of OOPs.

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

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

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

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

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

Ans-You can't access non-static data from a static function. This is because the static function can be called irrespective of whether there are any instantiated objects of the class. The non-static data, however, is dependent on a specific object (instantiation) of the class. Since you can't be sure that there *are* any objects instantiated when calling a static function, it is illogical (and therefore not allowed) to access non-static data from it.

89. Define different ways a method can be overloaded?
Ans-

- By changing the number of parameters in a method
- By changing the order of parameters in a method
- By using different data types for parameters

90. Can we have an abstract class without having any abstract method?
Ans-Yes you can. The abstract class used in java signifies that you can't create an object of the class. And an abstract method the subclasses have to provide an implementation for that method so you can easily define an abstract class without any abstract method.

91. Explain the default access modifier of a class ?

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of Java access modifiers:

1) Private 2)Default 3)Protected 4)Public

Default

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

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
```

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

//save by B.java
package mypack;
import pack.*;
class B{
public static void main(String args[]){
A obj = new A();//Compile Time Error
obj.msg();//Compile Time Error
 }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

92. Can Function overriding be explained in same class ?

No, you cannot override the same method in one class.

Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

93. Does function overloading depends on return type ?

The return type of a function has no effect on function overloading, therefore the same function signature with different return type will not be overloaded.

Function overloading is possible in Java but only if the functions differ from each other by the types and the number of arguments in the argument list. However, functions can not be overloaded if they differ only in the return type.

94. Can Abstract class have a constructor ?
Constructor is always called by its class name in a class itself. A constructor is used to initialize an object not to create the object.

Abstract classes also have a constructor. So if we do not define any constructor inside the abstract class then JVM (Java Virtual Machine) will give a default constructor to the abstract class.

95. Define rules of function overloading and function overriding ?

Rules of Overloading in Java

1] Method Signature

The first and foremost rule to overload a method in Java is to change the method signature. the method signature is made of a number of arguments, types of arguments, and order of arguments if they are of different types. You can change any of these or combinations of them to overload a method in Java.

For example, the println() method of PrintStream class has several overloaded versions to accept different data types like String, int, long, float, double, boolean, etc.

2] Method Return Type

The return type of method is not part of the method signature, so just changing the return type will not overload a method in Java. In fact, just changing the return type will result in a compile-time error as "duplicate method

Rules of Overriding in Java

1] Location

A method can only be overridden in sub-class, not in the same class. If you try to create two methods with the same signature in one class compiler will complain about it saying *"duplicate method in type Class"*

2] Exception

For terminology, the original method is known as overridden method and the new method is known as the overriding method.

Overriding method cannot throw checked Exception which is higher in the hierarchy, than checked Exception thrown by the overridden method. For example, if an overridden method throws IOException or ClassNotfoundException, which are checked Exception then the overriding method can not throw java.lang.Exception because it comes higher in type hierarchy (it's the superclass of IOException and ClassNotFoundExcepiton).

3] Visibility

The overriding method can not reduce access of overridden method. It means if the overridden method is defined as public then the overriding method can not be protected or package-private. Similarly, if the original method is protected then the overriding method cannot be package-private.

4] Accessibility

Overriding method can increase access of overridden method. This is the opposite of the earlier rule, according to this if the overridden method is declared as protected then the overriding method can be protected or public.

5] Types of Methods

The private, static, and final methods can not be overridden in Java. See other articles in this blog to learn why you cannot override private, static, or final methods in Java. By the way, you can hide private and static methods but trying to override the final method will result in compile-time error "Cannot override the final method from a class"

6] Return Type

The return type of overriding method must be the same as overridden method. Trying to change the return type of method in the child class will throw compile-time error "return type is incompatible with parent class method"