1. **What is Interface?**

An interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces. By default, members will be treated as public static final variables, so it expects some value to be initialized. Interfaces does not allow constructors. The variables inside interfaces are static final variables means constants and we cannot create object for interface.

**Example**

interface AInterface{

int A=10;

void show();

}

interface BInterface{

int B=20;

void show();

}

public class InterfaceEx implements AInterface,BInterface {

public static void main(String[] args) {

System.out.println(AInterface.A+" "+ BInterface.B);

InterfaceEx obj = new InterfaceEx();

obj.show();

}

@Override

public void show() {

System.out.println("Show Method");

}

}

1. **What is an Abstract Class?**

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

**abstract void moveTo(double deltaX, double deltaY);**

If a class includes abstract methods, then the class itself must be declared abstract, as in:

**public abstract class GraphicObject {**

**// declare fields**

**// declare nonabstract methods**

**abstract void draw();**

**}**

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

**Which should you use, abstract classes or interfaces?**

* Consider using abstract classes if any of these statements apply to your situation:
  + You want to share code among several closely related classes.
  + You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  + You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
* Consider using interfaces if any of these statements apply to your situation:
  + You expect that unrelated classes would implement your interface. For example, the interfaces [Comparable](https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html) and [Cloneable](https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html) are implemented by many unrelated classes.
  + You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  + You want to take advantage of multiple inheritance of type.

1. **Difference between throw and throws?**

The keyword ‘throw’ is used to throw an exception manually in Java. Using this keyword, it is possible to throw an exception from any method or block. However, it is essential that the exception must be of type java.lang.Throwable class or it belongs to one of the sub classes of java.lang.Throwable class.

**public class TestThrow {**

**public static void main(String[] args) {**

**Scanner s1 = new Scanner(System.in);**

**int age = s1.nextInt();**

**if(age>18){**

**System.out.println("You can Vote");**

**}else{**

**throw new ArithmeticException("You are not eligible.");**

**}}}**

The keyword ‘throws’ is used in the method signature in Java. If the method is capable of throwing exceptions, it is indicated by this method. The mentioned exceptions are handled by their respective caller functions. It is done either by using try and catch blocks or by using the throws keyword.

**public class TestThrows2 {**

**public void voting()throws Exception{**

**Scanner s1 = new Scanner(System.in);**

**int age = s1.nextInt();**

**if(age>18){**

**System.out.println("You can Vote");**

**}else{**

**throw new ArithmeticException("You are not eligible.");**

**}}**

**public static void main(String[] args) throws Exception{**

**TestThrows2 obj = new TestThrows2();**

**obj.voting();**

**}}**

1. **What is Abstraction, Encapsulation and Polymorphism?**

**Abstraction:** It is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

Abstract class (0 to 100%)

Interface (100%)

**Encapsulation :** Encapsulation describes the ability of an object to hide its data and methods from the rest of the world and is one of the fundamental principles of object-oriented programming. In Java, a class encapsulates the fields, which hold the state of an object, and the methods, which define the actions of the object. Encapsulation enables you to write reusable programs. It also enables you to restrict access only to those features of an object that are declared public. All other fields and methods are private and can be used for internal object processing.

**Polymorphism:** Polymorphism means one name many forms. It is concept by which you can perform same action in different ways.It is the ability for different objects to respond differently to the same message. In object-oriented programming languages, you can define one or more methods with the same name. These methods can perform different actions and return different values.

There are two types of polymorphism in java.

Compile time polymorphism

Run time polymorphism

1. **What is Inheritance and its types?**

Inheritance is one important concept of OOPs. In Inheritance, one class can inherit the properties(instance variables) and behavior(methods) of another class.

**Super Class:** This is the class from which the properties(instance variables) and behavior(methods) are inherited. Also called as base class or parent class.

**Sub Class**: This is the class which inherits the properties and behavior from another class. Also called as derived class or child class. The sub class can access all the methods and variables except the private members. It also can have its own instance variables and methods.

extends: Use the keyword extends to extend a class.

Java supports simple, multilevel, and hierarchical inheritance.

Simple: This is one level of inheritance. One class inherits from another class.

Multilevel: This type of inheritance goes down for multiple levels –

Hierarchical: Multiple classes can inherit from a single class or A class can have multiple subclasses.

1. **What is the final keyword and finally keyword?**

**final(lowercase)** is a reserved keyword in java. We can’t use it as an identifier as it is reserved. We can use this keyword with variables, methods and with classes. The final keyword in java has different meaning depending upon it is applied to variable, class, or method.If a class is declared as final, then **by default** all the methods present in that class are automatically final, but **variables are not**.

* **final with Variables :**The value of variable cannot be changed once initialized.
* **final with Class :** The class cannot be subclassed. Whenever we declare any class as final, it means that we can’t extend that class or that class **can’t be extended,** or we can’t make subclass of that class.
* **final with Method :** The method cannot be overridden by a subclass. Whenever we declare any method as final, then it means that we can’t override that method.

**Finally:** Just as final is a reserved keyword, so in same way finally is also a reserved keyword in java i.e., we can’t use it as an identifier. The finally keyword is used in association with a try/catch block and guarantees that a section of code will be executed, even if an exception is thrown. The finally block will be executed after the try and catch blocks, but before control transfers back to its origin.

**Application of finally block:** So basically, the use of finally block is resource deallocation. Means all the resources such as Network Connections, Database Connections, which we opened in try block are needed to be closed so that we won’t lose our resources as opened. So those resources are needed to be closed in finally block.

1. **What is a Collection?**

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

1. **ArrayList vs LinkedList?**

**ArrayList:** ArrayList is a part of the **collection framework.** It is present in the **java.util** package and provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. We can dynamically add and remove items. It automatically resizes itself.

* This class implements a [List interface](https://www.geeksforgeeks.org/list-interface-java-examples/). Therefore, this acts as a list.
* This class works better when the application demands storing the data and accessing it.

**LinkedList** is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays.

* This class implements both the List interface and the Deque interface. Therefore, it can act as a list and a deque.
* This class works better when the application demands manipulation of the stored data.

1. **Equal vs == difference?**

The main difference between the .equals() method and == operator is that one is a method and the other is the operator.

We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.