Java Interview Questions & Answers

1.What is java ?

- Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.
- It is a general-purpose programming language intended to let programmers write once, run anywhere, meaning that compiled Java code can run on all platforms that support Java without the need to recompile.
- it is used to develop a wide variety of applications, including:
- Web applications, Mobile applications, Desktop applications, Enterprise software, Scientific applications, and Embedded systems.

2.Explain the differ b/w JDK, JRE, JVM.

JDK :

Java Development Kit is a software development environment that includes JRE and development tools. It's used to create **Java applications and applets**. JDK includes tools like a **compiler, debugger, and documentation generator.**

JRE :

Java Runtime Environment is a set of software tools that provides a runtime environment for running other software. **It's used to run Java applications**. JRE contains **class libraries, supporting files, and the JVM.**

JVM :

Java Virtual Machine is the foundation of Java programming language and ensures the program's Java source code will be platform-agnostic. **It's used to run Java bytecode**. JVM is included in **both JDK and JRE, and Java programs won't run without it.**

3. what are the main features of java ?

1. Simple and Easy to Learn. Java is easy to learn and simple to use as a programming language.
2. Object-Oriented Programming.
3. Platform Independence.
4. Automatic Memory Management.
5. Security.
6. Rich API.
7. Multithreading.
8. High Performance
9. Scalability

4. what is differ b/w java and javascript ?

- Java is an OOP **programming language** while Java Script is an **OOP scripting language**.
- Java creates **applications that run in a virtual machine or browser** while JavaScript code is **run on a browser only.**
- Java code **needs to be compiled** while JavaScript code are **all in text.**

5. what is an object-oriented programming language?

Object-oriented programming (OOP) is a programming language that uses objects to bind data and the functions that operate on it. The goal of OOP is to implement real-world entities like inheritance, hiding, and polymorphism. OOP concepts include:

- Class
- Object
- Inheritance
- polymorphism
- encapsulation

- ❖ abstraction

6. what are the pillars of oops ?

The four pillars of OOPS (object-oriented programming) are **Inheritance, Polymorphism, Encapsulation and Data Abstraction**.

7. what is an class in java ?

- ➢ A class in Java is a set of objects which shares common characteristics/ behavior and common properties/ attributes.
- ➢ It is a user-defined blueprint or prototype from which objects are created.
- ➢ For example, Student is a class while a particular student named Ravi is an object

8. what is an object in java ?

- ➢ An object in Java is a basic unit of Object-Oriented Programming (OOP) and represents real-life entities.
- ➢ Objects are the instances of a class that are created to use the attributes and methods of a class.
- ➢ Java objects are very similar to the objects we can observe in the real world. A cat, a lighter, a pen, or a car are all objects.

9. How do you create an object in java ?

To create an object in Java, you use the **new** keyword, followed by the class name and parentheses. For example, to create an object of the class MyClass, you would write:

MyClass myObject = new MyClass();

Different ways to create objects in Java

- ❖ Using new keyword.
- ❖ Using new instance.
- ❖ Using clone() method.
- ❖ Using deserialization.
- ❖ Using newInstance() method of Constructor class.

10. what is differ b/w class and object in java ?

Class

- Class is a user-defined datatype that has its own data members and member functions.
- Class is a blueprint or prototype from which objects are created.
- It is a logical entity.
- It does not occupy any memory space.
- Class is declared using the class keyword.

Object

- Object is an instance of a class.
- Object is a real-world entity such as book, car, etc.
- It is a physical entity.
- It occupies memory space.
- Object is created using the new keyword.

11. Explain the concept of inheritance ?

- ➢ Inheritance is a mechanism in object-oriented programming that allows a class to inherit the properties and behaviors of another class.

- In Java, inheritance is implemented using the extends keyword. When a class inherits from another class, it is called a subclass or child class, and the class it inherits from is called a superclass or parent class. The subclass inherits all of the public and protected members of the superclass, including its fields, methods, and constructors.

Eg :

```
class Animal {
  String name;
  int age;

  void eat() {
    System.out.println("Animal is eating");
  }
}

class Dog extends Animal {
  String breed;

  void bark() {
    System.out.println("Dog is barking");
  }
}

public class Main {
  public static void main(String[] args) {
    Dog dog = new Dog();
dog.name = "Fido";
    dog.age = 5;
    dog.breed = "Labrador";
    dog.eat(); // Prints "Animal is eating"
    dog.bark(); // Prints "Dog is barking"
  }
}
```

12. what is the purpose of super keyword ?

The super keyword can be used to call the superclass constructor from the subclass constructor. This is done by using the super() keyword followed by the arguments to the superclass constructor.

For example,

```
class Superclass {
  public Superclass(int x) {
    System.out.println("Superclass constructor called with x = " + x);
  }
}

class Subclass extends Superclass {
  public Subclass(int x, int y) {
    super(x); // Call the superclass constructor with the value of x
    System.out.println("Subclass constructor called with y = " + y);
```

```
  }
}

public class Main {
  public static void main(String[] args) {
    Subclass subclass = new Subclass(10, 20);
  }
}
```

OUTPUT :

Superclass constructor called with x = 10

Subclass constructor called with y = 20

13. What is Polymorphism in java ?

- Polymorphism is derived from two Greek words, “poly” and “morph”, which mean “many” and “forms”, respectively.
- Hence, polymorphism meaning in Java refers to the ability of objects to take on many forms.
- In other words, it allows **different objects to respond to the same message or method call in multiple ways.**

For Example:

```
public class Animal {
  public void makeSound() {
    System.out.println("Animal sound");
  }
}

public class Dog extends Animal {
  @Override
  public void makeSound() {
    System.out.println("Woof!");
  }
}

public class Cat extends Animal {
  @Override
  public void makeSound() {
    System.out.println("Meow!");
  }
}

public class Main {
  public static void main(String[] args) {
    Animal animal = new Animal();
    animal.makeSound(); // Prints "Animal sound"

    Dog dog = new Dog();
    dog.makeSound(); // Prints "Woof!"

    Cat cat = new Cat();
```

```
    cat.makeSound(); // Prints "Meow!"
  }
}
```

14. Explain the difer b/w method overloading & method overriding in java ?

Method overloading is when a class has **two or more methods with the same name but different parameters**. This allows us to have multiple methods with the same name that perform different tasks, depending on the arguments passed to them.

For example, we could have a calculateArea() method that takes a single argument (the radius of a circle) and returns the area of the circle, and we could also have a calculateArea() method that takes two arguments (the length and width of a rectangle) and returns the area of the rectangle.

Method overriding is when a **subclass provides its own implementation of a method that is already defined in the superclass.** This allows us to customize the behavior of a method in a subclass, without having to break the code that relies on the superclass method.

For example, we could have a draw() method in a Shape class that simply prints the message "Drawing a shape". We could then override the draw() method in a Circle subclass to print the message "Drawing a circle".

Feature	Method overloading	Method overriding
Number of methods	Two or more methods with the same name	One method in the subclass and one method in the superclass
Parameters	Different parameters	Same parameters
Return type	Same or different return type	Same return type
Inheritance	Not required	Required
Purpose	To provide multiple methods with the same name that perform different tasks	To customize the behavior of a method in a subclass

15. what is Encapsulation in java ?

Encapsulation in Java is the process by which **data (variables) and the code that acts upon them (methods) are integrated as a single unit.** By encapsulating a class's variables, other classes cannot access them, and only the methods of the class can access them.

```
public class Person {
    private String name;
    private int age;

    // getters and setters
    public String getName() {
```

```
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }
}
```

In this example, the data members name and age are declared as private, and public getter and setter methods are provided to access and modify them. This way, the internal implementation of the Person class is hidden from the outside world, and the data members can only be accessed or modified through the public methods.

16. What are the purpose of access modifiers in java ?

- Access modifiers in Java are used to control the **access level of classes, methods, variables, and constructors**.
- They are used to restrict access to certain parts of a program, which can help to improve security and make the code more maintainable.

17. differ b/w public, private, protected & default in java ?

There are four access modifiers in Java:

Public:

Public members are accessible from anywhere in the program.

Private:

Private members are only accessible from within the class in which they are declared.

Protected:

Protected members are accessible from within the class in which they are declared, as well as from any subclasses of that class.

Default:

Default members are accessible from within the package in which they are declared.

18. What is abstract class in java ?

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

```
abstract class Shape {
  // abstract method
  abstract void draw();
}
```

```
// subclass of Shape
class Circle extends Shape {
  // overriding the draw() method
  @Override
  void draw() {
    System.out.println("Drawing a circle");
  }
}

// subclass of Shape
class Rectangle extends Shape {
  // overriding the draw() method
  @Override
  void draw() {
    System.out.println("Drawing a rectangle");
  }
}

// main class
public class Main {
  public static void main(String[] args) {
    // creating an object of Circle class
    Circle circle = new Circle();
    // calling the draw() method on circle object
    circle.draw();

    // creating an object of Rectangle class
    Rectangle rectangle = new Rectangle();
    // calling the draw() method on rectangle object
    rectangle.draw();
  }
}
```

Output:

Drawing a circle

Drawing a rectangle

19. How do you achieve abstraction in java ?

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces. Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

// Java Program to implement

// Java Abstraction

// Abstract Class declared

```
abstract class Animal {
        private String name;

        public Animal(String name) { this.name = name; }

        public abstract void makeSound();

        public String getName() { return name; }
}

// Abstracted class
class Dog extends Animal {
        public Dog(String name) { super(name); }

        public void makeSound()
        {
                System.out.println(getName() + " barks");
        }
}

// Abstracted class
class Cat extends Animal {
        public Cat(String name) { super(name); }

        public void makeSound()
        {
                System.out.println(getName() + " meows");
        }
}

// Driver Class
public class AbstractionExample {
        // Main Function
        public static void main(String[] args)
        {
                Animal myDog = new Dog("Buddy");
                Animal myCat = new Cat("Fluffy");

                myDog.makeSound();
                myCat.makeSound();
        }
}
```

OUTPUT :

Buddy barks

Fluffy meows

20. What is constructor in java ?

A constructor in Java is a special method that is used to initialize objects. It is called when an object of a class is created. It can be used to set initial values for object attributes.

Constructors are similar to methods, but they have some important differences:

- Constructors have the same name as the class they are in.
- Constructors do not have a return type.
- Constructors are called automatically when an object is created.

Here is an example of a constructor:

```
public class Person {
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

This constructor takes two parameters, a name and an age, and uses them to initialize the corresponding object attributes.

To create a new Person object, you would use the following code:

```
Person person = new Person("John Doe", 30);
```

21. What is differ b/w default & parameterized constructor in java ?

Default Constructor:

- A default constructor is a constructor that has no parameters.
- It is implicitly provided by the compiler if no constructor is explicitly defined in a class.
- A default constructor initializes all instance variables to their default values.

```
class Student {
  int id;
  String name;

  // Default constructor
  Student() {
    id = 0;
    name = "null";
  }
}
```

To create an instance of the Student class using the default constructor, you would simply write:

```
Student student = new Student();
```

Parameterized Constructor :

- A parameterized constructor is a constructor that has one or more parameters.
- It is explicitly defined by the programmer.

- A parameterized constructor allows you to initialize instance variables to specific values at the time of object creation.

```
class Student {
  int id;
  String name;

  // Parameterized constructor
  Student(int id, String name) {
    this.id = id;
    this.name = name;
  }
}
```

To create an instance of the Student class using the parameterized constructor, you would write:

Student student = new Student(1, "John Doe");

22. What is the purpose of the static keyword in java ?

- The **static keyword** in Java is mainly used for **memory management**. The static keyword in Java is used to share the same variable or method of a given class.
- The users can apply static keywords with **variables, methods, blocks, and nested classes**. The static keyword belongs to the class than an instance of the class.

23. What is static method in java ?

A static method in Java is a **method that belongs to a class rather than an instance of a class**. Static methods are used to access and change static variables and other non-object-based static methods.

Here are some features of static methods:

- Static methods are called using the class name, not the instance name.
- Static methods can only access static variables and other static methods.
- Static methods cannot access non-static variables or non-static methods.
- Static methods are typically used for utility functions, such as mathematical functions or string manipulation functions.

24. **What is the purpose of the final keyword in java ?**

- The final keyword is a **non-access modifier used for classes, attributes and methods, which makes them non-changeable or unmodifiable**. (impossible to inherit or override).
- The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...). The final keyword is called a "modifier".

25. Explain differ b/w final, finally, and finalize in java ?

1. **final** is a keyword used in Java to **restrict the modification of a variable, method, or class.**
2. **finally** is a block used in Java to **ensure that a section of code is always executed, even if an exception is thrown.**
3. **finalize** is a method in Java used to **perform cleanup processing on an object before it is garbage collected.**

26. What is method overriding in java ?

- In Java, method overriding occurs when a subclass (child class) has the same method as the parent class. In other words, method overriding occurs when a subclass provides a particular implementation of a method declared by one of its parent classes.
- To override a method, the subclass must have the same method name, return type, and parameter list as the method in the parent class. The overriding method can also return a subtype of the type returned by the overridden method.

Here is an example of method overriding:

```
class Animal {
  public void move() {
    System.out.println("Animals can move");
  }
}

class Dog extends Animal {
  @Override
  public void move() {
    System.out.println("Dogs can walk and run");
  }
}

public class TestDog {
  public static void main(String[] args) {
    Animal a = new Animal();
    Animal b = new Dog();
    a.move(); // prints "Animals can move"
    b.move(); // prints "Dogs can walk and run"
  }
}
```

In this example, the Dog class overrides the move() method from the Animal class. The Dog class's move() method prints a different message than the Animal class's move() method.

When you call the move() method on a Dog object, the Dog class's move() method is called, even though the Dog object is an instance of the Animal class. This is because the Dog class overrides the move() method.

Method overriding is a powerful feature that allows you to customize the behavior of classes in Java. It is one of the key features that makes Java an object-oriented programming language.

27. What is method overloading in java ?

Method overloading in Java means having two or more methods (or functions) in a class with the same name and different arguments (or parameters). It can be with a different number of arguments or different data types of arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
```

```
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

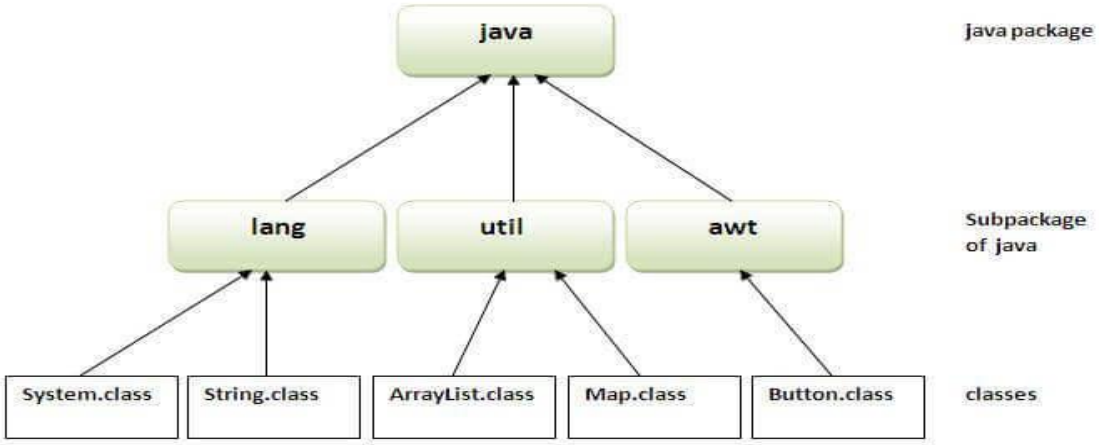
22

24.9

28. What is a package in java ?

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



29. How do you import packages in java ?

Packages are used to avoid naming conflicts, and to control access to classes. For example, the java.lang package contains classes such as String, Object, and Math, which are fundamental to the Java language**.**

Packages are declared using the package keyword.

```
Package mypackage;
```

the following code imports the String class from the java.lang package:

```
import java.lang.String;
```

the following code declares a nested package named mypackage.subpackage:

```
package mypackage.subpackage;
```

30. What is the purpose of import statement in java ?

An import statement tells Java which class you mean when you use a short name (like List). It tells Java where to find the definition of that class. You can import just the classes you need from a package as shown below. Just provide an import statement for each class that you want to use.

Import statement in Java is helpful to take a class or all classes visible for a program specified under a package, with the help of a single statement.

```
import java.util.List;
import java.util.*;
import static java.lang.Math.PI;
```

31. What is java interface ?

An interface is an abstract type that is used to declare a behavior that classes must implement. They are similar to protocols in other languages.

Here are some of the key features of interfaces in Java:

- Interfaces can contain only abstract methods.
- Interfaces cannot contain concrete methods.
- Interfaces cannot contain instance variables.
- Interfaces can be used to achieve multiple inheritance.
- Interfaces can be extended by other interfaces.
- Interfaces can be implemented by classes.

```
class Dog implements Animal {
  @Override
  public void animalSound() {
    // The body of animalSound() is provided here
  }

  @Override
  public void run() {
    // The body of run() is provided here
  }
}
```

32. Explain the differ b/w abstract classes and interfaces in java ?

Feature	Abstract Class	Interface
Definition	A class that is declared abstract cannot be instantiated, but it can be subclassed.	An interface is a reference type that is used to declare a set of abstract methods.
Methods	Abstract classes can have both abstract and non-abstract methods.	Interfaces can only have abstract methods.
Variables	Abstract classes can have variables.	Interfaces cannot have variables.
Inheritance	Abstract classes can only be extended by one class.	Interfaces can be implemented by multiple classes.
Implementation	Abstract classes can provide default implementations for abstract methods.	Interfaces cannot provide implementations for abstract methods.

33. Can you implement multiple interfaces in java ?

our class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class. By convention, the implements clause follows the extends clause, if there is one**.**

34 . What is a java Abstract class ?

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). Abstract method: can only be used in an abstract class, and it does not have a body.

35. What is the purpose of the instance of the operator in java ?

The instanceof operator in Java is used to check whether an object is an instance of a particular class or not. objectName instanceOf className; Here, if objectName is an instance of className , the operator returns true . Otherwise, it returns false .

For Eg :

```
if (myObject instanceof Dog) {
  // myObject is a Dog object
} else if (myObject instanceof Animal) {
  // myObject is an Animal object, but not a Dog object
} else {
  // myObject is not an Animal object
}
```

36. What are the different types of variables in java ?

There are three different types of variables in Java:

Local variables

the method, constructor, or block is entered and destroyed when it exits. Declare **inside a method, constructor, or block**. They are only accessible within the scope in which they are declared. Local variables are created when

Instance variables

declared **outside of any method, constructor, or block, but within a class**. They are accessible to all methods within the class. Instance variables are created when an object is created and destroyed when the object is destroyed**.**

Static variables

declared with the static keyword. They are **shared among all instances of a class. Static variables are created when the class is loaded and destroyed when the class is unloaded.**

Here is an example of each type of variable:

```
public class MyClass {
  // Local variable
  private int myLocalVariable;

  // Instance variable
  private int myInstanceVariable;

  // Static variable
  private static int myStaticVariable;

  public void myMethod() {
    // Local variable
    int myLocalVariable = 10;

    // Access instance variable
    myInstanceVariable = 20;
 / Access static variable
    myStaticVariable = 30;
  }
}
```

37. How do you declare and initialize variable in java ?

```
// Declare an integer variable
int myNum = 5;

// Declare a float variable
float myFloatNum = 5.99f;
```

```
// Declare a character variable
char myLetter = 'D';

// Declare a boolean variable
boolean myBool = true;

// Declare a String variable
String myText = "Hello";
```

38. What is the purpose of volatile keyword in java ?

The volatile keyword in **Java is used to indicate that a variable's value can be modified by different threads.** Used with the syntax, volatile dataType variableName = x; It ensures that changes made to a volatile variable by one thread are immediately visible to other threads.

```
public class VolatileExample extends Thread {
   private volatile boolean running = true;

   public void run() {
      while (running) {
         System.out.println("Running");
      }
   }

   public void stopRunning() {
      running = false;
   }
}

public class Main {
   public static void main(String[] args) throws
InterruptedException {
      VolatileExample example = new
VolatileExample();
      example.start();
 // Sleep for 1 second
      Thread.sleep(1000);

      example.stopRunning();
   }
}
```

Output:

Running...

(After 1 second)

(No more output, thread stops)

39 . What is the purpose of transient keyword in java ?

The transient keyword is a variable modifier in Java used in the context of serialization. When applied to a variable, it instructs the Java Virtual Machine (JVM) to exclude that variable from the serialization process. Transient variables are not saved in the serialized form of the object.

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
import java.util.Date;

public class Student implements Serializable {

 String firstName;
 String secondName;
 transient String fullName;
 String email;
 String password;
 Date dob;

 /*
  Constructor
  */
 Student(
  String firstName,
  String secondName,
  String email,
  String password,
  Date dob
 ) {
  this.firstName = firstName;
  this.secondName = secondName;
  this.email = email;
  this.password = password;
  this.dob = dob;
  this.fullName = firstName + " " + secondName;
 }

 public static void main(String[] args) {
  Student student = new Student(
   "Salman",
   "Khan",
   "salman.khan@gmail.com",
   "raindeer@123",
   new Date()
  );

  serialize(student);
  deserialize();
 }
```

```
 /*
   Method to serialize object
    */
 private static void serialize(Student student) {
   try {
     System.out.println("Student serializing: " + student.toString());
     FileOutputStream fileOut = new FileOutputStream("student.ser");
     ObjectOutputStream out = new ObjectOutputStream(fileOut);
     out.writeObject(student);
     out.close();
     fileOut.close();
   } catch (IOException i) {
     i.printStackTrace();
   }
 }
 /*
   Method to deserialize object
    */
 private static void deserialize() {
   try {
     FileInputStream fileIn = new FileInputStream("student.ser");
     ObjectInputStream in = new ObjectInputStream(fileIn);
     Student student = (Student) in.readObject();
     in.close();
     fileIn.close();
     System.out.println("Student deserialized: " + student.toString());
   } catch (IOException | ClassNotFoundException i) {
     i.printStackTrace();
   }
 }
/*
   Method to get String value of object
*/
 @Override
 public String toString() {
   return (
     "Student{" +
     "firstName='" +
     firstName +
     '\'' +
     ", secondName='" +
     secondName +
     '\'' +
     ", fullName='" +
     fullName +
     '\'' +
     ", email='" +
     email +
     '\'' +
     ", password='" +
     password +
```

```
    '\'' +
    ", dob=" +
    dob +
    '}'
  );
 }
}
```

RESULT:

Student serializing : Student{firstName='Salman', secondName='Khan', fullName='Salman Khan', email='salman.khan@gmail.com', password='raindeer@123', dob=Sat Jul 17 21:42:29 IST 2021}

Student deserialized : Student{firstName='Salman', secondName='Khan', fullName='null', email='salman.khan@gmail.com', password='raindeer@123', dob=Sat Jul 17 21:42:29 IST 2021}

40. What is a java array ?

In Java, an array is a **data structure that can store a fixed-size sequence of elements of the same data type**. An array is an object in Java, which means it can be assigned to a variable, passed as a parameter to a method, and returned as a value from a method.

41. How do you declare and initialize array in java ?

For eg;

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
int[] myNum = {10, 20, 30, 40};
you can use and iterate over the elements in array
for (int i = 0; i < myArray.length; i++) {
  System.out.println(myArray[i]);
}
```

42. What is the length of an array in java ?

The theoretical maximum Java array size is **2,147,483,647** elements. To find the size of a Java array, query an array's length property. The Java array size is set permanently when the array is initialized. The size or length count of an array in Java includes both **null and non-null characters.**

The following code prints length of the array:

```
int[] arr = new int[5];
System.out.println(arr.length); // Prints 5
```

43. What is diff b/w Array and ArrayList in java ?

Array

- An array is a data structure that stores a collection of items. The size of an array is fixed, meaning that it cannot be changed once it is created.
- Arrays can store both primitive and object types.
- Arrays are accessed using indexes. The index of an element is its position in the array, starting from 0.
- Elements can be inserted into and deleted from arrays, but this can be slow and inefficient.
- Arrays can be sorted using the Arrays.sort() method.
- Arrays can be multidimensional, meaning that they can store arrays of arrays.
- Arrays cannot store null values.
- Arrays are generally faster than ArrayLists, but they are also less flexible.

ArrayList

- An ArrayList is a resizable array. This means that its size can be changed as needed.
- ArrayLists can only store object types.
- ArrayLists are accessed using methods, such as get(), add(), and remove().
- Elements can be inserted into and deleted from ArrayLists efficiently.
- ArrayLists can be sorted using the Collections.sort() method.
- ArrayLists can only be single-dimensional.
- ArrayLists can store null values.
- ArrayLists are generally slower than arrays, but they are also more flexible.

44. What is the purpose of the clone() method in java ?

clone() is a method in the Java programming language for **object duplication**. In Java, objects are manipulated through reference variables, and there is no operator for copying an object—the assignment operator duplicates the reference, not the object. The clone() method provides this missing functionality.

```
public class MyClass {
  private int myField;

  public MyClass(int myField) {
    this.myField = myField;
  }

  public MyClass clone() {
    MyClass newMyClass = new
MyClass(myField);
    return newMyClass;
  }
```

```
        }

        public class Main {
          public static void main(String[] args) {
            MyClass myClass = new MyClass(10);
            MyClass clonedMyClass = myClass.clone();

            System.out.println(myClass.myField); // 10

System.out.println(clonedMyClass.myField); // 10
          }
        }
```

45. What is the purpose of the toString() method in java ?

The toString() method in Java is a pre-existing method found in the Object class. It serves the purpose **of returning a string representation of an object**. By default, it produces a string comprising the object's class name, followed by an "@" symbol and hash code**.**

```
public class Person {
   private String name;
   private int age;

   public Person(String name, int age) {
      this.name = name;
      this.age = age;
   }

   @Override
   public String toString() {
      return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
   }
}
```

In this example, the toString() method is overridden to return a string that contains the person's name and age. This string can then be printed to the console using the System.out.println() method:

```
Person person = new Person("John Doe", 30);
System.out.println(person);
```

This will print the following output to the console:

Person{name='John Doe', age=30}

46. What is the purpose of the equals() methods in java ?

equals() method is primarily used to compare the **'value'** of two objects. It's an instance method that's part of the Object class, which is the parent class of all classes in Java. This means you can use . equals() to compare any two objects in Java.

For Eg:

```
String str1 = "Hello, World!";
String str2 = "Hello, World!";
// Compare the two strings for equality
boolean areEqual = str1.equals(str2);
// Print the result
System.out.println(areEqual);
```

47 What is the purpose of the hashcode() method in java ?

- The **hashCode()** method in Java is a built-in function used to return an integer hash code representing the value of the object, used with the syntax,

  int hash = targetString. hashCode();

- It plays a crucial role in data retrieval, especially when dealing with Java collections like HashMap and HashSet.

48. What is the purpose of the getClass() method in java ?

getClass() in Java is a method of the Object class present in java. lang package. **getClass() returns the runtime class of the object "this".** This returned class object is locked by static synchronized method of the represented class.

```
Object obj = new String("Hello, world!");
Class c = obj.getClass();
System.out.println(c.getName());
```

49. What is Exception handling in java ?

The Java programming language uses exceptions to handle errors and other exceptional events. the process of responding to unwanted or unexpected events when a computer program runs. There are 5 keywords.

- Try
- Catch
- Throw
- Throws
- Finally

50. Explain try, catch and finally blocks in java ?

1. try Block

Enclose the code that might throw an exception within a **try** block. If an exception occurs within the try block, that exception is handled by an exception handler associated with it. The try block contains at least one **catch** block or **finally** block.

The syntax of the try-catch block:

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

The syntax of a try-finally block:

```
try{
//code that may throw exception
}finally{}
```

2. catch Block

Java catch block is used to handle the Exception. It must be used after the *try* block only. You can use multiple catch blocks with a single try.

Syntax:

```
try
{
      //code that cause exception;
}
catch(Exception_type  e)
{
      //exception handling code
}
```

3 finally Block

- Java *finally* block is a block that is used to execute important code such as closing connection, stream, etc.
- Java *finally* block is always executed whether an exception is handled or not.
- Java *finally* block follows try or catch block.
- For each try block, there can be zero or more catch blocks, but only one *finally* block.

- The *finally* block will not be executed if the program exits(either by calling *System.exit()* or by causing a fatal error that causes the process to abort).

Syntax:

```
try {
    // Code that might throw an exception
} catch (ExceptionType1 e1) {
    // Code to handle ExceptionType1
} catch (ExceptionType2 e2) {
    // Code to handle ExceptionType2
}
// ... more catch blocks if necessary ...
finally {
    // Code to be executed always, whether an exception occurred or not
}
```

Example 1:

In this example, we have used **FileInputStream** to read the *simple.txt* file. After reading a file the resource **FileInputStream** should be closed by using *finally* block.

```
public class FileInputStreamExample {
    public static void main(String[] args) {

        FileInputStream fis = null;
        try {
            File file = new File("sample.txt");
            fis = new FileInputStream(file);
            int content;
            while ((content = fis.read()) != -1) {
                // convert to char and display it
                System.out.print((char) content);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
```

51. What is the purpose of throw and throws keyword in java ?

throw Keyword

The *throw* keyword is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exceptions using the *throw* keyword. The throw keyword is followed by an instance of the exception.

Syntax:

```
throw exception_instance;
```

```
public class ThrowExample {

    private int age;

    public void setAge(int age) {
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative!");
        }
        this.age = age;
    }

    public static void main(String[] args) {
        ThrowExample person = new ThrowExample();

        try {
            person.setAge(-5);   // This will cause an exception
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
Error: Age cannot be negative!
```

In the *setAge* method, if the age provided is negative, we throwan **IllegalArgumentException** with a relevant message.

Throws keyword

The *throws* keyword is used to declare exceptions. It doesn't throw an exception but specifies that a method might throw exceptions. It's typically used to inform callers of the exceptions they might encounter.

Syntax:

```
return_type method_name() throws exception_class_name{
//method code
}
```

Example:

```
public class ExceptionHandlingWorks {

    public static void main(String[] args) {
        exceptionHandler();
    }

    private static void exceptionWithoutHandler() throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(new
File("/invalid/file/location")))) {
            int c;
            // Read and display the file.
            while ((c = reader.read()) != -1) {
                System.out.println((char) c);
            }
        }
    }

    private static void exceptionWithoutHandler1() throws IOException {
        exceptionWithoutHandler();
    }

    private static void exceptionWithoutHandler2() throws IOException {
        exceptionWithoutHandler1();
    }

    private static void exceptionHandler() {
        try {
            exceptionWithoutHandler2();
        } catch (IOException e) {
            System.out.println("IOException caught!");
        }
    }
}
```

52. What is difference between checked and un checked Exceptions in java ?

Checked Exceptions

- They occur at compile time.
- The compiler checks for a checked exception.
- These exceptions can be handled at the compilation time.
- It is a sub-class of the exception class.
- The JVM requires that the exception be caught and handled.
- Example of Checked exception- 'File Not Found Exception'

Unchecked Exceptions

- These exceptions occur at runtime.
- The compiler doesn't check for these kinds of exceptions.
- These kinds of exceptions can't be caught or handled during compilation time.

- This is because the exceptions are generated due to the mistakes in the program.
- These are not a part of the 'Exception' class since they are runtime exceptions.
- The JVM doesn't require the exception to be caught and handled.
- Example of Unchecked Exceptions- 'No Such Element Exception'

53. What is the purpose of the try-with-resources statement in java ?

In Java, the try-with-resources statement is a try statement that **declares one or more resources**. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

```
        try (BufferedReader br = new BufferedReader(new FileReader("myfile.txt")))
{
          String line;
          while ((line = br.readLine()) != null) {
            System.out.println(line);
          }
        }
```

54. What is java string ?

A Java string is a sequence of characters that exists as an object of the class java. lang. Java strings are created and manipulated through the string class. Once created, a string is immutable -- its value cannot be changed.

```
        String str1 = "Hello";
        String str2 = "World";
        // Concatenate two strings
        String str3 = str1 + str2;
        // Get the length of a string
        int length = str3.length();
        // Find the index of a character in a string
        int index = str3.indexOf('o');
        // Substring a string
        String substring = str3.substring(0, 5);
        // Compare two strings
        boolean isEqual = str1.equals(str2);
```

55. How do you create and manipulate strings in java ?

Strings in Java are immutable, which means that they cannot be changed once they are created. To create a string, you can use the new keyword and the String constructor. For example, the following code creates a string object named str and assigns it the value "Hello World":

```
String str = new String("Hello World");
```

You can also create strings using string literals. String literals are sequences of characters enclosed in double quotes. For example, the following code creates a string object named str and assigns it the value "Hello World":

```
String str = "Hello World";
```

Once you have created a string object, you can manipulate it using the methods provided by the String class. Some of the most common string manipulation methods include:

length() : Returns the length of the string.

charAt() : Returns the character at a specified index in the string.

substring() : Returns a new string that is a substring of the original string.

indexOf() : Returns the index of the first occurrence of a specified character or substring in the string.

lastIndexOf() : Returns the index of the last occurrence of a specified character or substring in the string.

replace() : Returns a new string that is a copy of the original string with all occurrences of a specified character or substring replaced with another character or substring.

toUpperCase(): Returns a new string that is a copy of the original string with all characters converted to uppercase.

toLowerCase(): Returns a new string that is a copy of the original string with all characters converted to lowercase.

For example, the following code uses the **length()** method to print the length of the string str:

```
System.out.println(str.length());
```

The following code uses the **charAt()** method to print the character at index 0 in the string str:

```
System.out.println(str.charAt(0));
```

The following code uses the substring() method to print the substring of the string str that starts at index 0 and ends at index 4:

```
System.out.println(str.substring(0, 4));
```

The following code uses the **indexOf()** method to print the index of the first occurrence of the character 'o' in the string str:

```
System.out.println(str.indexOf('o'));
```

The following code uses the **lastIndexOf()** method to print the index of the last occurrence of the character 'o' in the string str:

```
System.out.println(str.lastIndexOf('o'));
```

The following code uses the **replace()** method to print a new string that is a copy of the string str with all occurrences of the character 'o' replaced with the character 'a':

```
System.out.println(str.replace('o', 'a'));
```

The following code uses the **toUpperCase()** method to print a new string that is a copy of the string str with all characters converted to uppercase:

System.out.println(str.toUpperCase());

The following code uses the **toLowerCase()** method to print a new string that is a copy of the string str with all characters converted to lowercase:

System.out.println(str.toLowerCase());

These are just a few of the many string manipulation methods that are available in Java. For more information, please see the Java documentation for the String class.

56. What is difference b/w String, StringBuilder, and StringBuffer in java ?

String :

- String is **immutable**, meaning that it **cannot be changed** after it is created. This is because String objects are stored in a **String pool**, which is a shared area of memory that is used to store **all String objects** in a Java program.

StringBuilder :

- StringBuilder is a **mutable class**, meaning that it **can be changed** after it is created. This is because StringBuilder objects are **not stored in the String pool**.
- StringBuilder is also **thread-safe**, meaning that it can be safely used by multiple threads at the same time. This is because StringBuilder methods are **synchronized.**

StringBuffer :

- StringBuffer is also a **mutable class**, meaning that it **can be changed** after it is created. This is because StringBuffer objects are **not stored in the String pool.**
- StringBuffer is also **thread-safe**, meaning that it can be safely used by multiple threads at the same time. This is because StringBuffer methods are **synchronized.**

57. What is a java Arraylist ?

An ArrayList class is a **resizable array**, which is present in the **"java. util package"**. While built-in arrays have a fixed size, ArrayLists can **change their size dynamically**. Elements can be added and removed from an ArrayList whenever there is a need, helping the user with memory management.

58. How do you create and manipulate ArrayLists in java ?

Here are the steps on how to create and manipulate ArrayLists in Java:

To create an ArrayList, you can:

Import the java.util.ArrayList package.
Create an ArrayList. For example, to create an ArrayList of strings, you can use:
ArrayList<String> languages = new ArrayList<>();

Here are some other methods of the ArrayList class:

- **get()** : Returns an element from the ArrayList**.**
- **set()** : Changes an element of the ArrayList**.**
- **remove()** : Removes an element from the ArrayList**.**
- **size()** : Returns the number of elements in the ArrayList.
- **isEmpty()** : Checks if the ArrayList is empty.
- **contains()** : Checks if the ArrayList contains a specific element.
- **clear()** : Removes all elements from the ArrayList.

59. What is the purpose of iterator interface in java ?

Java Iterator Interface of java collections **allows us to access elements of the collection and is used to iterate over the elements in the collection(Map, List or Set).** It helps to easily retrieve the elements of a collection and perform operations on each element.

The Iterator interface has three methods:

- **hasNext()** - Returns true if the iterator has more elements.
- **next()** - Returns the next element in the iteration.
- **remove()** - Removes the last element returned by the next() method from the collection.

Eg :

```
import java.util.ArrayList;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> names = new ArrayList<>();
    names.add("John");
    names.add("Doe");
    names.add("Jane");

    Iterator<String> iterator = names.iterator();
    while (iterator.hasNext()) {
      String name = iterator.next();
      System.out.println(name);
    }
  }
}
```

Output :

John

Doe

Jane

60. What is java HashMap ?

HashMap in Java stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer). One object is used as a key (index) to another object (value). If you try to insert the duplicate key in HashMap, it will replace the element of the corresponding key.

Here are some of the key features of HashMaps:

- HashMaps store elements in key-value pairs.
- HashMaps are unsorted, which means that the order in which elements are added to the map is not preserved.
- HashMaps allow for one null key and multiple null values.
- HashMaps provide efficient access and manipulation of data based on unique keys.
- HashMaps are widely used in Java applications.

Here is an example of how to use a HashMap in Java:

```
import java.util.HashMap;

public class HashMapExample {

   public static void main(String[] args) {

      HashMap<String, Integer> hashMap = new HashMap<>();

      // Add elements to the HashMap
      hashMap.put("one", 1);
      hashMap.put("two", 2);
      hashMap.put("three", 3);

      // Get the value for a key
      Integer value = hashMap.get("two");

      // Remove an element from the HashMap
      hashMap.remove("three");

      // Check if the HashMap contains a key
      boolean containsKey = hashMap.containsKey("one");

      // Print the contents of the HashMap
      System.out.println(hashMap);
   }
}
```

Output:

{one=1, two=2}

61. How do you create and manipulate Hashmaps in java ?

Here's how to create and manipulate a HashMap in Java:

Create a HashMap instance using the syntax **HashMap<KeyType, ValueType>.** KeyType specifies the type of keys, and ValueType specifies the type of values the map will hold. For example, to create a HashMap called numberMapping that stores key-value pairs of strings and integers, you can use the following code:

```
Map<String, Integer> numberMapping = new HashMap<>();
```

Add key-value pairs to the HashMap using the **put()** function. For example, to add the key-value pairs "One" to 1, "Two" to 2, and "Three" to 3, you can use the following code:

```
numberMapping.put("One", 1); numberMapping.put("Two", 2); numberMapping.put("Three", 3);
```

Access elements in the HashMap using the **get()** function. For example, to print the value associated with the key "John", you can use the following code:

```
System.out.println(hashMap.get("John"));
```

Remove an element from the HashMap using the **remove()** function. For example, to remove the element associated with the key "Jim", you can use the following code:

```
hashMap.remove("Jim");
```

Check if an element is present in the HashMap using the **containsKey()** function. For example, to check if the element associated with the key "Jim" is present, you can use the following code:

```
System.out.println(hashMap.containsKey("Jim"));
```

62. What is the purpose of entrySet() method in java Hashmap ?

The Java HashMap. **entrySet()** method is used **to convert the elements within a HashMap into a Set.** This provides a convenient way to access and manipulate the elements stored in the HashMap.

The **entrySet()** method is useful for iterating over the entries in a map. For example, the following code iterates over the entries in a map and prints the keys and values:

```
HashMap<String, Integer> map = new HashMap<>();
map.put("one", 1);
map.put("two", 2);
for (Map.Entry<String, Integer> entry : map.entrySet()) {
  System.out.println(entry.getKey() + " : " + entry.getValue());
}
```

The **entrySet()** method can also be used to remove entries from a map. For example, the following code removes the entry with the key "one" from the map:

```
map.entrySet().removeIf(entry -> entry.getKey().equals("one"));
```

The **entrySet()** method is a powerful tool for working with maps in Java. It can be used to iterate over the entries in a map, remove entries from a map, and more.

63. What is the purpose of the put() and get() methods in java HashMap ?

- The **put()**and **get()** methods are two of the most important methods in the Java HashMap class.
- The **put()** method is used **to add a new key-value pair to the map,**
- the **get()** method is used **to retrieve the value associated with a given key.**
- The **put()** method returns the previous value associated with the key, or null if there was **no previous value**.
- The **get()** method returns the value associated with the key, or null if there is **no value associated with the key.**

Here is an example of how to use the **put()** and **get()** methods:

```
HashMap<String, Integer> map = new HashMap<>();
// Add a new key-value pair to the map
map.put("name", "John Doe");
// Get the value associated with the key "name"
String name = map.get("name");
// Print the value
System.out.println(name);
```

OUTPUT:

John Doe

64. what is the purpose of the remove() methods in java HashMap ?

The **remove()** method **removes the mapping and returns: the previous value associated with the specified key.** true if the mapping is removed.

The **remove()** method can be used to delete an item from a HashMap. This can be useful for a variety of reasons, such as:

- To remove an item that is no longer needed.
- To remove an item that is causing problems.
- To remove an item that is outdated.

Here is an example of how to use the **remove()** method:

SOURCE CODE:

```
HashMap<String, Integer> hashMap = new HashMap<>();
hashMap.put("one", 1);
hashMap.put("two", 2);
hashMap.put("three", 3);
// Remove the item with the key "two"
Integer removedValue = hashMap.remove("two");
// Print the removed value
System.out.println(removedValue); // 2
```

65. What is a java HashSet ?

- HashSet in Java is a class from the **Collections Framework.**
- It allows you to store multiple values in a collection using a hash table.
- The hash table stores the values in an unordered method with the help of hashing mechanism**.**

Here are some of the methods of HashSet in Java:

- **add(E e)** : Adds the specified element to the HashSet.
- **contains(Object o)** : Returns true if the specified element is present in the HashSet.
- **remove(Object o)** : Removes the specified element from the HashSet.
- **size()** : Returns the number of elements in the HashSet.
- **isEmpty()** : Returns true if the HashSet is empty.
- **clear()** : Removes all elements from the HashSet.

66. How do you create and manipulate HashSets in java ?

We **add elements to the HashSet using the add() method, and then we print the HashSet using the println() method.** We demonstrate checking if an element exists in the HashSet using the **contains()** method and removing an element using the **remove()** method.

SOURCE CODE:

```
import java.util.HashSet;
public class HashSetExample {
   public static void main(String[] args) {
         / Create a HashSet
      HashSet<String> names = new HashSet<>();
         // Add elements to the HashSet
      names.add("John");
      names.add("Alice");
      names.add("Bob");
          // Print the HashSet
      System.out.println("HashSet: " + names);
          // Check if an element exists in the HashSet
      boolean containsAlice = names.contains("Alice");
      System.out.println("Contains 'Alice': " + containsAlice);
          // Remove an element from the HashSet
      names.remove("Bob");
      System.out.println("After removal: " + names);
   }
}
```

OUTPUT :

HashSet: [Alice, Bob, John]

Contains 'Alice': true

After removal: [Alice, John]

67. What is the purpose of the add () method in java Hashset ?

add() method in Java HashSet is used **to add a specific element into a HashSet.** This method will add the element only if the specified element is not present in the HashSet else the function will return False if the element is already present in the HashSet.

```
HashSet<String> hashSet = new HashSet<>();
// Add a new element to the HashSet
hashSet.add("Element 1");
// Check if an element is present in the HashSet
boolean isPresent = hashSet.contains("Element 1");
// Print the result
System.out.println(isPresent); // true
```

In this example, we create a new HashSet and add the element **"Element 1"** to it. We then check if the element **"Element 1"** is present in the HashSet. The **contains()** method returns true, which means that the element is present in the HashSet.

The **add()** method is a very useful method for adding new elements to a HashSet and for checking if an element is already present in a HashSet.

68. What is the purpose of the remove () method in java HashSet ?

remove() method is present in the HashSet class inside the java. util package. The HashSet **remove()** method is used **to remove only the specified element from the HashSet** .

SOURCE CODE:

```
import java.util.HashSet;
public class Main {
   public static void main(String[] args) {
      HashSet<String> names = new HashSet<>();
      names.add("John");
      names.add("Mary");
      names.add("Bob");
  // Remove the element "Mary" from the HashSet
      names.remove("Mary");
          // Print the remaining elements in the HashSet
      for (String name : names) {
         System.out.println(name);
      }
   }
}
```

OUTPUT :

John

Bob

69. What is a java LinkedList ?

A linked list in Java is **a dynamic data structure whose size increases as you add the elements and decreases as you remove the elements from the list**. The elements in the linked list are stored in containers. The list holds the link to the first container.

70. How do you create and manipulate LinkedList in java ?

Here are the steps on how to create and manipulate a Java LinkedList:

Import the LinkedList class :

The LinkedList class is part of the **java.util package**, so you need to import it before you can use it.

Create a new LinkedList object :

You can do this by using the new keyword, followed by the LinkedList class name.

Add elements to the LinkedList :

You can add elements to the LinkedList using the **add() method**. The **add()** method takes an element as an argument and adds it to the end of the LinkedList.

Remove elements from the LinkedList :

You can remove elements from the LinkedList using the **remove()** method. The **remove()** method takes an element as an argument and removes it from the LinkedList.

Get the size of the LinkedList :

You can get the size of the LinkedList using the **size() method**. The **size()** method returns the number of elements in the LinkedList.

Check if the LinkedList is empty :

You can check if the LinkedList is empty using the **isEmpty() method**. The **isEmpty()** method returns true if the LinkedList is empty, and false otherwise.

Iterate over the LinkedList:

You can iterate over the LinkedList using a **for-each loop**. The for-each loop will iterate over each element in the LinkedList and print it to the console.

Here is an example of how to create and manipulate a Java LinkedList:

```
import java.util.LinkedList;
public class Main {
   public static void main(String[] args) {
      // Create a new LinkedList object
      LinkedList<String> names = new LinkedList<>();
          // Add elements to the LinkedList
      names.add("John");
      names.add("Mary");
      names.add("Bob");
          // Remove an element from the LinkedList
      names.remove("Bob");
```

```
        // Get the size of the LinkedList
    int size = names.size();
        // Check if the LinkedList is empty
    boolean isEmpty = names.isEmpty();
         // Iterate over the LinkedList
    for (String name : names) {
       System.out.println(name);
    }
  }
}
```

OUTPUT:

John

Mary

71. What is the purpose of the add () and remove ()methods in java LinkedList ?

The **add()** and **remove()** methods in Java LinkedList are used to add and remove elements from a linked list. The **add()** method takes an element as a parameter and adds it to the end of the list. The **remove()** method takes an element as a parameter and removes the first occurrence of that element from the list.

Here is an example of how to use the add() and remove() methods:

```
import java.util.LinkedList;
public class Main {
  public static void main(String[] args) {
    LinkedList<String> list = new LinkedList<>();
        // Add elements to the list
    list.add("Hello");
    list.add("World");
        // Print the list
    System.out.println(list);
        // Remove an element from the list
    list.remove("Hello");
         // Print the list again
    System.out.println(list);
  }
}
```

OUTPUT

[Hello, World]

[World]

72. What is java TreeSet?

Published in the Java Collections group. Java provides a vast set of data structures for efficiently working with element collections. One such data structure is TreeSet, **an implementation of a red-black tree in Java.** TreeSet maintains a sorted order for storing unique elements.

Source Code :

```
import java.util.*;
public class TreeSetExample {

   public static void main(String[] args) {
         // Create a TreeSet
      TreeSet<String> names = new TreeSet<>();

      // Add some elements to the TreeSet
      names.add("Alice");
      names.add("Bob");
      names.add("Carol");
      names.add("Dave");

      // Print the elements of the TreeSet
      for (String name : names) {
         System.out.println(name);
      }
         // Check if the TreeSet contains an element
      System.out.println(names.contains("Alice"));      //
true

      // Remove an element from the TreeSet
      names.remove("Bob");

      // Print the elements of the TreeSet
      for (String name : names) {
         System.out.println(name);
      }
   }
}
```

OUTPUT :

Alice

Carol

Dave

true

Alice

Carol

Dave

73. What is the purpose of add() and remove() methods in java TreeSets ?

The **add() method** in a Java TreeSet is used **to add a new element to the s**et. If the element is already present in the set, the add() method will do nothing and return false. Otherwise, the element will be added to the set and the method will return true.

The **remove() method** in a Java TreeSet is used **to remove an element from the set**. If the element is not present in the set, the remove() method will do nothing and return false. Otherwise, the element will be removed from the set and the method will return true**.**

Here is an example of how to use the add() and remove() methods in a Java TreeSet:

```
import java.util.*;
public class TreeSetExample {
public static void main(String[] args) {

      TreeSet<Integer> treeSet = new TreeSet<>();
           // Add elements to the TreeSet
      treeSet.add(1);
      treeSet.add(2);
      treeSet.add(3);
      treeSet.add(4);
      treeSet.add(5);
            // Remove an element from the TreeSet
      treeSet.remove(3);
           // Print the elements in the TreeSet
      for (Integer element : treeSet) {
         System.out.println(element);
      }
   }
}
```

OUTPUT:

1

2

4

5

74. What is the purpose of the contains() method in java Collections ?

The **contains()** method of Java AbstractCollection is used **to check whether an element is present in a Collection or not**. It takes the element as a parameter and returns True if the element is present in the collection.

SOURCE CODE:

```
import java.util.ArrayList;
public class Main {
   public static void main(String[] args) {
      ArrayList<String> names = new ArrayList<>();
```

```
        names.add("John");
        names.add("Mary");
        names.add("Bob");
            // Check if the collection contains the element "John"
        boolean containsJohn = names.contains("John");
            // Print the result
        System.out.println(containsJohn);
    }
}
```

OUTPUT :

True

75. **What is the purpose of the isEmpty() method in java collections ?**

The **isEmpty()** method is a convenient way **to check if a collection is empty**. It is available in all collection interfaces, so you can use it with any type of collection.

Source code :

```
import java.util.*;
public class Example {

    public static void main(String[] args) {
            // Create a new list
        List<String> names = new ArrayList<>();
            // Check if the list is empty
        if (names.isEmpty()) {
            System.out.println("The list is empty.");
        } else {
            System.out.println("The list is not empty.");
        }
            // Add an element to the list
        names.add("John Doe");
            // Check if the list is empty
        if (names.isEmpty()) {
            System.out.println("The list is empty.");
        } else {
            System.out.println("The list is not empty.");
        }
    }
}
```

Output :

The list is empty.

The list is not empty.

76. What is the purpose of size() method in java collections ?

The **size()** method simply retrieves the **value of the internal variable that tracks the number of elements in the ArrayList.** This variable is updated every time an element is added or removed from the ArrayList.

The **size()** method in Java collections is used to get the number of elements in a collection. It is a very useful method for determining the length of a collection and for iterating over its elements. The **size()** method is available in all collection interfaces, including List, Set, and Map.

Here is an example of how to use the size() method:

```
import java.util.*;
public class Example {

   public static void main(String[] args) {
         // Create a list of elements
      List<String> names = new ArrayList<>();
      names.add("John");
      names.add("Mary");
      names.add("Bob");
         // Get the size of the list
      int size = names.size();
         // Print the size of the list
      System.out.println("The size of the list is: " + size);
   }
}
```

OUTPUT:

```
The size of the list is: 3
```

77. What is the purpose of the clear() method in java collections ?

The **clear()** method of Java Collection Interface removes all of the elements from this collection. It returns a Boolean value 'true', if it successfully empties the collection**.**

SOURCE CODE :

```
import java.util.ArrayList;
public class Main {
   public static void main(String[] args) {
      ArrayList<String> list = new ArrayList<>();
      list.add("Hello");
      list.add("World");
         // Print the contents of the list
      System.out.println(list);
         // Clear the list
      list.clear();
         // Print the contents of the list again
      System.out.println(list);
   }
}
```

OUTPUT

[Hello, World]

[]

78. What is the purpose of the ToArray() method in java collections ?

The **toArray()** method of the **ArrayList is used to convert an ArrayList to an array in Java.** This function either takes in no parameter or takes in an array of Type T(T[] a) in which the element of the list will be stored. The toArray() function returns an Object array if no argument is passed.

SOURCE CODE

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      List<String> list = new ArrayList<>();
      list.add("Hello");
      list.add("World");
         // Convert the list to an array
      String[] array = list.toArray(new String[0]);
         // Print the array
      for (String s : array) {
         System.out.println(s);
      }
   }
}
```

Output:

Hello

World

79. What is the purpose of the iterator () method in java collections ?

Java Iterator Interface of java collections allows us to **access elements of the collection and is used to iterate over the elements in the collection(Map, List or Set).** It helps to easily retrieve the elements of a collection and perform operations on each element.

The Iterator interface provides three methods: **hasNext(), next(), and remove().**

1. The **hasNext()** method returns true if there are more elements in the collection, and false otherwise.
2. The **next()** method returns the next element in the collection
3. The **remove()** method removes the current element from the collection.

The iterator() method is a very useful method for iterating over collections in Java. It allows you to easily access and manipulate the elements of a collection.

SOURCE CODE:

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
   public static void main(String[] args) {
      ArrayList<String> names = new ArrayList<>();
      names.add("Alice");
      names.add("Bob");
      names.add("Charlie");

      Iterator<String> iterator = names.iterator();

      while (iterator.hasNext()) {
         String name = iterator.next();
         System.out.println(name);
      }
   }
}
```

OUTPUT :

Alice

Bob

Charlie

80. What is the java Comparator ?

A comparator interface is used **to order the objects of user-defined classes.** A comparator object is capable of comparing two objects of the same class.

The Comparator interface in Java is used to order the objects of a user-defined class. It provides a single method, compare(), which takes two objects as input and returns an integer value indicating whether the first object is less than, equal to, or greater than the second object. The compare() method has the following signature:

int compare(T o1, T o2);

where T is the type of object being compared.

The Comparator interface can be used to sort collections of objects using the Collections.sort() or Arrays.sort() methods. It can also be used to create sorted sets and maps.

Here is an example of how to use the Comparator interface to sort a collection of objects:

SOURCE CODE :

```
import java.util.*;
public class ComparatorExample {

   public static void main(String[] args) {
         // Create a list of objects
      List<Integer> numbers = new ArrayList<>();
      numbers.add(10);
```

```
        numbers.add(5);
        numbers.add(20);
            // Create a comparator to compare the objects by their values
        Comparator<Integer> comparator = new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o1.compareTo(o2);
            }
        };
            // Sort the list using the comparator
        Collections.sort(numbers, comparator);
            // Print the sorted list
        for (Integer number : numbers) {
            System.out.println(number);
        }
    }
}
```

OUTPUT

5

10

20

81. How do you implement custom sorting using a comparator in java ?

To sort a collection of objects by multiple criteria, first define the class representing your objects. Then**, create a custom comparator class that implements the Comparator interface. Override the compare method to define how objects should be compared based on different criteria**

SOURCE CODE:

```
import java.util.Comparator;
public class StudentComparator implements Comparator<Student> {
@Override
    public int compare(Student student1, Student student2) {
        int nameComparison = student1.getName().compareTo(student2.getName());
        if (nameComparison == 0) {
            return student1.getAge() - student2.getAge();
        } else {
            return nameComparison;
        }
    }
}
```

To use this comparator, you would simply pass it to the sort() method of a list of students:

SOURCE CODE:

```
List<Student> students = new ArrayList<>();
students.add(new Student("Alice", 12));
students.add(new Student("Bob", 10));
students.add(new Student("Carol", 11));

students.sort(new StudentComparator());
```

After sorting, the students list will be in the following order:

[Bob, Carol, Alice]

You can use custom comparators to sort any collection of objects, regardless of their type. This can be useful for sorting objects based on multiple criteria or for sorting objects in a specific order that is not defined by the object's class.

82. What is the purpose of the compareTo() method in java ?

The **compareTo()** method **returns an integer value that represents the comparison result**. If the result is less than 0, str1 comes before str2 in lexicographical order. If the result is greater than 0, str1 comes after str2. If the result is 0, it means that both strings are equal.

SOURCE CODE:

```
String str1 = "Hello";
String str2 = "World";
int result = str1.compareTo(str2);
if (result > 0) {
  System.out.println("str1 is greater than str2");
} else if (result < 0) {
  System.out.println("str1 is less than str2");
} else {
  System.out.println("str1 is equal to str2");
}
```

OUTPUT :

"str1 is less than str2"

83. What is the purpose of the equals() method in java ?

equals() method is primarily used **to compare the 'value' of two objects**. It's an instance method that's part of the Object class, which is the parent class of all classes in Java. This means you can use . equals() to compare any two objects in Java.

Source code :

```
String str1 = "Hello, World!";
String str2 = "Hello, World!";
```

```
// Compare the two strings for equality.
boolean isEqual = str1.equals(str2);
// Print the result.
System.out.println(isEqual);
```

OUTPUT:

True //Because two strings are equal

84. What is the purpose of the HashCode() method in java ?

The **hashCode()** method in Java is a built-in function used **to return an integer hash code representing the value of the object,** used with the syntax, **int hash = targetString. hashCode();** . It plays a crucial role in data retrieval, especially when dealing with Java collections like HashMap and HashSet.

SOURCE CODE :

```
String str = "Hello, world!";
int hashCode = str.hashCode();

System.out.println(hashCode);
```

OUTPUT

"Hello, world!"

85. what is java thread ?

A thread in Java is **the direction or path that is taken while a program is being executed.** Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution.

86. How do you create and start a thread in java ?

There are two ways to create threads in Java:

1. By extending the Thread class and implementing the **run()** method.
2. By implementing the Runnable interface and passing an instance of the class to the Thread constructor.

Once a thread is created, it can be started by calling the **start()** method. The thread will then run concurrently with the main thread until it finishes executing its **run() method**.

To create a thread by extending the Thread class, you need to create a subclass of Thread and override the **run()** method. The **run()** method contains the code that will be executed by the thread.

To start a thread, you need to call the **start()** method on the thread object. The **start()** method causes the thread to begin executing the **run()** method.

Here is an example of how to create and start a thread by extending the Thread class:

SOURCE CODE :

```
public class MyThread extends Thread {
   @Override
   public void run() {
      System.out.println("Hello from MyThread!");
   }
}

public class Main {
   public static void main(String[] args) {
      MyThread thread = new MyThread();
      thread.start();
   }
}
```

OUTPUT

Hello from MyThread!

create a thread by implementing the Runnable interface, you need to create a class that implements the Runnable interface. The Runnable interface has a single method, run(), which contains the code that will be executed by the thread.

To start a thread, you need to create a Thread object and pass the Runnable object to the constructor. Then, you need to call the start() method on the Thread object.

Here is an example of how to create and start a thread by implementing the Runnable interface:

SOURCE CODE :

```
public class MyRunnable implements Runnable {
   @Override
   public void run() {
      System.out.println("Hello from MyRunnable!");
   }
}

public class Main {
   public static void main(String[] args) {
      MyRunnable runnable = new MyRunnable();
      Thread thread = new Thread(runnable);
      thread.start();
   }
}
```

OUTPUT :

Hello from MyRunnable!

87. What is difference between a thread and a process in java ?

Process :

- ✓ A process is an independent unit of execution in Java.
- ✓ It has its own memory space, which contains its code, data, and stack.
- ✓ A process can have multiple threads.
- ✓ Processes are created and destroyed by the operating system.
- ✓ Processes communicate with each other through inter-process communication (IPC) mechanisms, such as pipes, sockets, and shared memory.

Thread :

- ✓ A thread is a lightweight unit of execution within a process.
- ✓ It shares the process's memory space, but it has its own stack.
- ✓ A thread can be created and destroyed by the Java Virtual Machine (JVM).
- ✓ Threads communicate with each other through shared memory**.**

Feature	Thread	Process
Memory space	Shares the process's memory space	Has its own memory space
Stack	Has its own stack	Has its own stack
Creation	Created and destroyed by the JVM	Created and destroyed by the operating system
Communication	Communicates with other threads through shared memory	Communicates with other processes through IPC mechanisms

88. What is synchronization in java ?

Synchronization in java is **the capability to control the access of multiple threads to any shared resource**. In the Multithreading concept, multiple threads try to access the shared resources at a time to produce inconsistent results. The synchronization is necessary for reliable communication between threads.

89. How do you achieve synchronization in java ?

In Java, synchronization can be achieved **using the synchronized keyword and the volatile modifier.** The synchronized Keyword: The synchronized keyword is used to define critical sections of code that should be accessed by only one thread at a time. It can be applied to methods or code blocks.

Eg :

```
// Synchronize a method
public synchronized void increment() {
  // ...
}

// Synchronize a code block
synchronized (obj) {
  // ...
}
```

90. What is the purpose of the synchronized keyword in java ?

The synchronized keyword in Java is a powerful tool for **achieving thread safety and synchronization in multithreaded applications.** By using synchronized blocks or synchronized methods, you can control concurrent access to shared resources, prevent data inconsistencies, and ensure proper thread synchronization.

They can be used in two ways:

To synchronize a method.

```
public synchronized void methodName() {
  // code to be synchronized
}
```

To be synchronized code

```
synchronized (object) {
  // code to be synchronized
}
```

Here are some examples of how the synchronized keyword can be used:

// Synchronize a method

```
public synchronized void withdrawMoney(int amount) {
  // code to withdraw money from the account
}

// Synchronize a code block
synchronized (account) {
  // code to access and modify the account
}
```

91. What are the states of a thread in java ?

A thread in Java can exist in one of the following **six states** at any given time:

1. **New:** When a thread object is created, it is in the new state. It is not yet runnable.
2. **Runnable:** A thread that is ready to run is in the runnable state. It is waiting to be scheduled by the operating system.
3. **Running**: A thread that is currently executing is in the running state.

4. **Blocked:** A thread that is waiting for an event to occur, such as a lock to become available, is in the blocked state.
5. **Waiting:** A thread that is waiting for another thread to finish executing is in the waiting state.
6. **Terminated:** A thread that has finished executing is in the terminated state

Example :

```
Thread thread = new Thread();
Thread.State state = thread.getState();
```

92. what is deadlock in java ?

Deadlock in Java is **a condition where two or more threads are blocked forever, waiting for each other.** This usually happens when multiple threads need the same locks but obtain them in different orders. Multithreaded Programming in Java suffers from the deadlock situation because of the synchronized keyword.

93. How do you prevent deadlock in java ?

Here are some ways to prevent deadlock in Java:

Avoid unnecessary locking :

Only lock resources when absolutely necessary, and release them as soon as possible.

Acquire locks in a consistent order :

This means that all threads should acquire locks in the same order, to avoid situations where two threads are waiting for each other to release locks.

Avoid nested locks :

This means that a thread should not acquire a second lock while it is already holding another lock.

Use timeouts when acquiring locks.

This will help to prevent situations where a thread is waiting indefinitely for a lock to be released.

Use lock-free data structures :

These data structures are designed to be used by multiple threads without the need for locks.

Use a deadlock detection and avoidance algorithm :

This can be a complex solution, but it can be effective in preventing deadlocks in situations where other methods are not feasible.

Here are some additional tips for preventing deadlocks in Java:

Use proper synchronization techniques :

When multiple threads are accessing the same data, it is important to use synchronization techniques to ensure that the data is not corrupted.

Be aware of the potential for deadlocks :

When designing your code, be aware of the potential for deadlocks and take steps to avoid them.

Test your code thoroughly :

Test your code thoroughly to ensure that it is free of deadlocks.

These tips, you can help to prevent deadlocks in your Java code.

94. What is the purpose of the Wait(), notify(), notifyAll() methods in java ?

The wait(), notify(), and notifyAll() methods in Java are used to coordinate actions of multiple threads. They are part of the Object class and are used to implement inter-thread communication.

- **wait():** Causes the current thread to wait indefinitely until another thread invokes notify() or notifyAll() on the same object.
- **notify():** Wakes up a single thread that is waiting on that object's monitor.
- **notifyAll():** Wakes up all threads that are waiting on that object's monitor.

These methods are typically used in conjunction with synchronized blocks to ensure that only one thread can access a shared resource at a time. For example, a producer-consumer problem could be solved using wait() and notify() as follows:

SOURCE CODE :

```
class Producer {
   private Object lock;

   public Producer(Object lock) {
      this.lock = lock;
   }

   public void produce() {
      synchronized (lock) {
         // Produce an item
         // Notify the consumer that an item is available
         lock.notify();
      }
   }
}

class Consumer {
   private Object lock;

   public Consumer(Object lock) {
      this.lock = lock;
   }

   public void consume() {
      synchronized (lock) {
         // Wait for an item to be produced
```

```
            lock.wait();
            // Consume the item
        }
    }
}
```

The wait(), notify(), and notifyAll() methods are powerful tools for coordinating the actions of multiple threads. However, they must be used carefully to avoid race conditions and deadlocks**.**