

Java Interview questions for Freshers

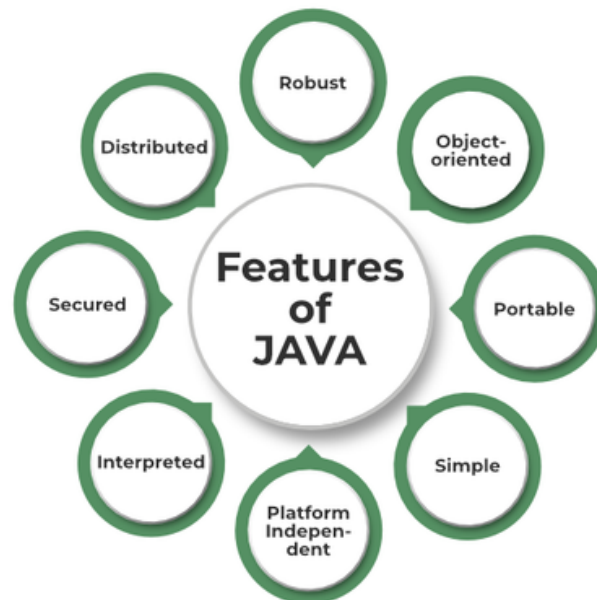
1. Is Java Platform Independent if then how?

Yes, Java is a Platform Independent language. Unlike many programming languages javac compiles the program to form a bytecode or .class file. This file is independent of the software or hardware running but needs a JVM(Java Virtual Machine) file preinstalled in the operating system for further execution of the bytecode.

Although **JVM is platform dependent**, the bytecode can be created on any System and can be executed in any other system despite hardware or software being used which makes Java platform independent.

2. What are the top Java Features?

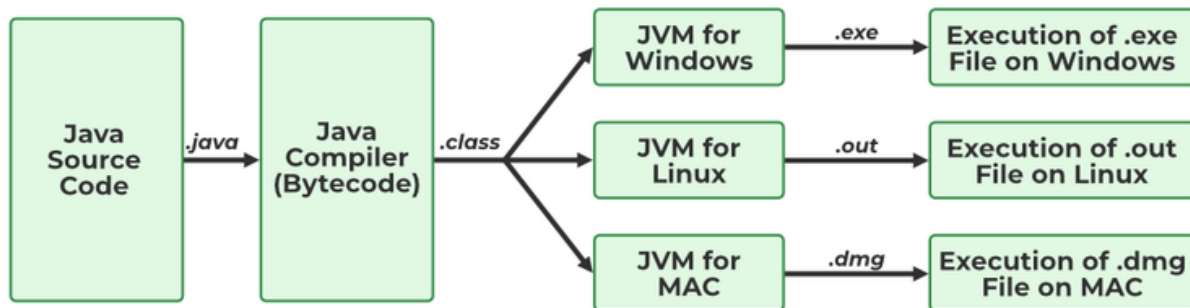
Java is one the most famous and most used language in the real world, there are many features in Java that makes it better than any other language some of them are mentioned below:



- **Simple:** Java is quite simple to understand and the syntax
- **Platform Independent:** Java is platform independent means we can run the same program in any software and hardware and will get the same result.
- **Interpreted:** Java is interpreted as well as a compiler-based language.
- **Robust:** features like Garbage collection, exception handling, etc that make the language robust.
- **Object-Oriented:** Java is an object-oriented language that supports the concepts of class, objects, four pillars of OOPS, etc.
- **Secured:** As we can directly share an application with the user without sharing the actual program makes Java a secure language.
- **High Performance:** faster than other traditional interpreted programming languages.
- **Dynamic:** supports dynamic loading of classes and interfaces.
- **Distributed:** feature of Java makes us able to access files by calling the methods from any machine connected.

- **Multithreaded:** deal with multiple tasks at once by defining multiple threads
- **Architecture Neutral:** it is not dependent on the architecture.

3. What is JVM?

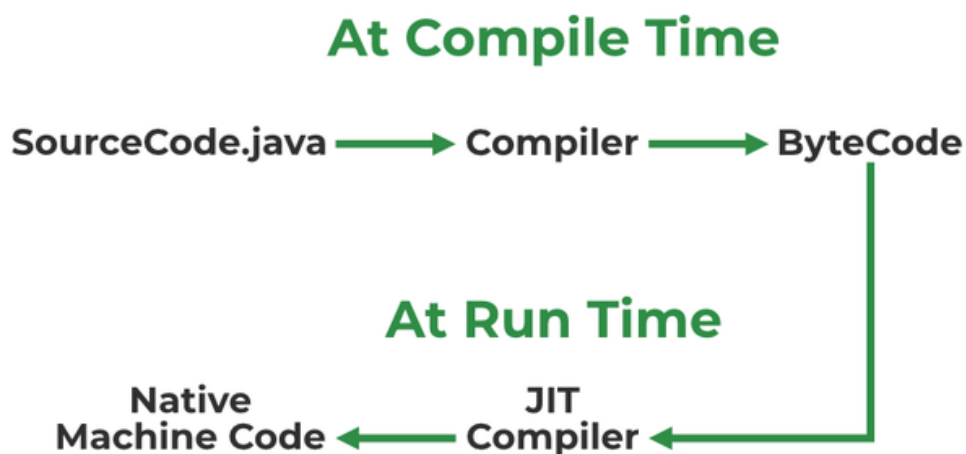


JVM stands for Java Virtual Machine it is a Java interpreter. It is responsible for loading, verifying, and executing the bytecode created in Java.

Although it is platform dependent which means the software of JVM is different for different Operating Systems it plays a vital role in making Java platform Independent.

To know more about the topic refer to [JVM in Java](#).

4. What is JIT?

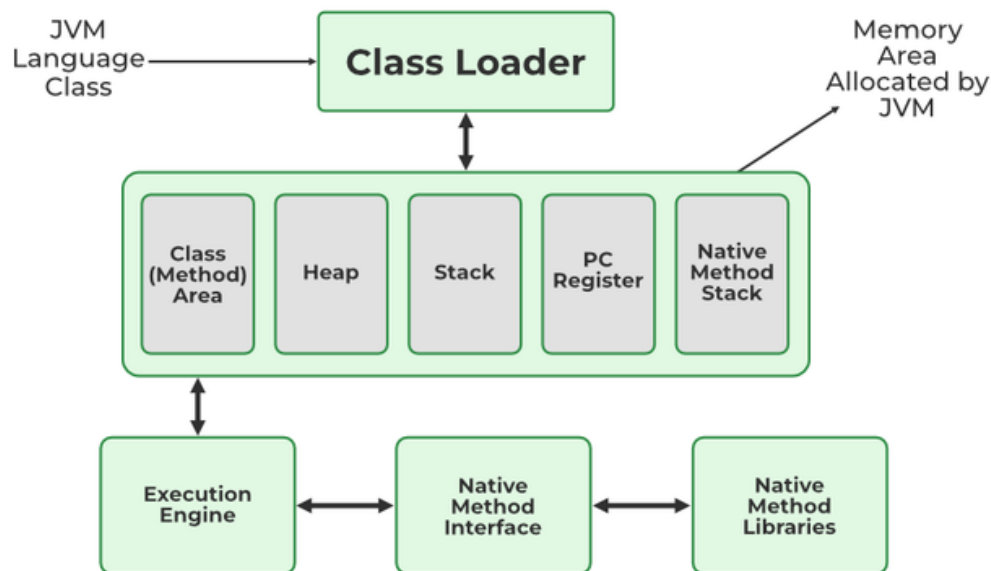


JIT stands for (Just-in-Time) compiler is a part of JRE(Java Runtime Environment), it is used for better performance of the Java applications during run-time. The use of JIT is mentioned in step by step process mentioned below:

1. Source code is compiled with **javac** to form bytecode
2. Bytecode is further passed on to JVM
3. JIT is a part of JVM, JIT is responsible for compiling bytecode into native machine code at run time.
4. The JIT compiler is enabled throughout, while it gets activated when a method is invoked. For a compiled method, the JVM directly calls the compiled code, instead of interpreting it.
5. As JVM calls the compiled code that increases the performance and speed of the execution.

To know more about the topic refer to [JIT in Java](#).

5. What are Memory storages available with JVM?



JVM consists of a few memory storages as mentioned below:

1. Class(Method) Area: stores class-level data of every class such as the runtime constant pool, field, and method data, and the code for methods.
2. Heap: Objects are created or objects are stored. It is used to allocate memory to objects during run time.
3. Stack: stores data and partial results which will be needed while returning value for method and performing dynamic linking
4. Program Counter Register: stores the address of the Java virtual machine instruction currently being executed.
5. Native Method Stack: stores all the native methods used in the application.

To know more about the topic refer to [JVM Memory Storages](#).

6. What is a classloader?

ClassLoader is the part of JRE(Java Runtime Environment), during the execution of the bytecode or created .class file classloader is responsible for dynamically loading the java classes and interfaces to JVM(Java Virtual Machine). Because of classloaders Java run time system does not need to know about files and file systems.

To know more about the topic refer to [ClassLoader in Java](#).

7. Difference between JVM, JRE, and JDK.

JVM: JVM also known as Java Virtual Machine is a part of JRE. JVM is a type of interpreter responsible for converting bytecode into machine-readable code. JVM itself is platform dependent but it interprets the bytecode which is the platform-independent reason why Java is platform-independent.

JRE: JRE stands for Java Runtime Environment, it is an installation package that provides an environment to run the Java program or application on any machine.

JDK: JDK stands for Java Development Kit which provides the environment to develop and execute Java programs. JDK is a package that includes two things Development Tools to provide an environment to develop your Java programs and, JRE to execute Java programs or applications.

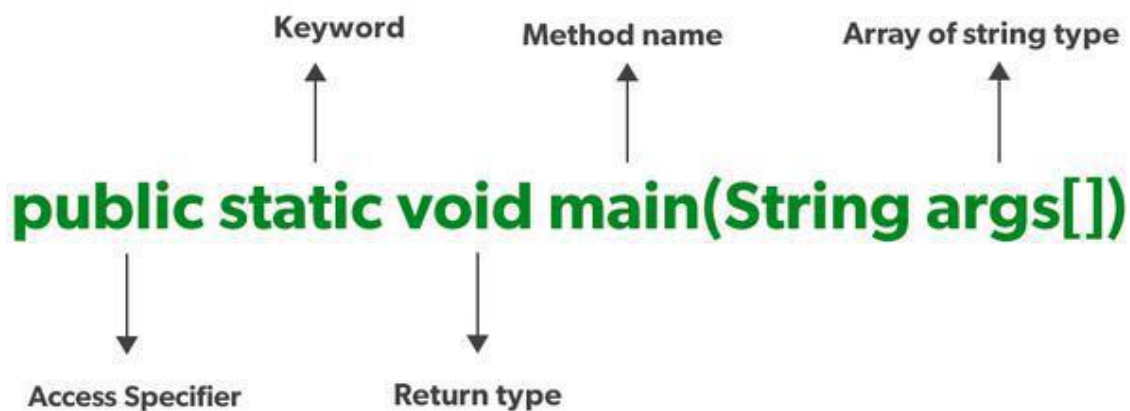
To know more about the topic refer to the [Differences between JVM, JRE, and JDK](#).

8. What are the differences between Java and C++?

Basis	C++	Java
Platform	C++ is Platform Dependent	Java is Platform Independent
Application	C++ is mainly used for System Programming	Java is Mainly used for Application Programming
Hardware	C++ is nearer to hardware	Java is not so interactive with hardware
Global Scope	C++ supports global and namespace scope.	Java doesn't support global scope.
Not Supporting	Functionality supported in Java but not in C++ are: <ul style="list-style-type: none">• thread support• documentation comment• unsigned right shift(>>>)	Functionality supported in C++ but not in Java are: <ul style="list-style-type: none">• goto• Pointers• Call by reference• Structures and Unions• Multiple Inheritance• Virtual Functions

Basis	C++	Java
OOPS	C++ is an object-oriented language. It is not a single root hierarchy .	Java is also an object-oriented language. It is a single root hierarchy as everything gets derived from a single class (java.lang.Object).
Inheritance Tree	C++ always creates a new inheritance tree.	Java uses a Single inheritance tree as classes in Java are the child of object classes in Java.

9. Explain public static void main(String args[]) in Java.



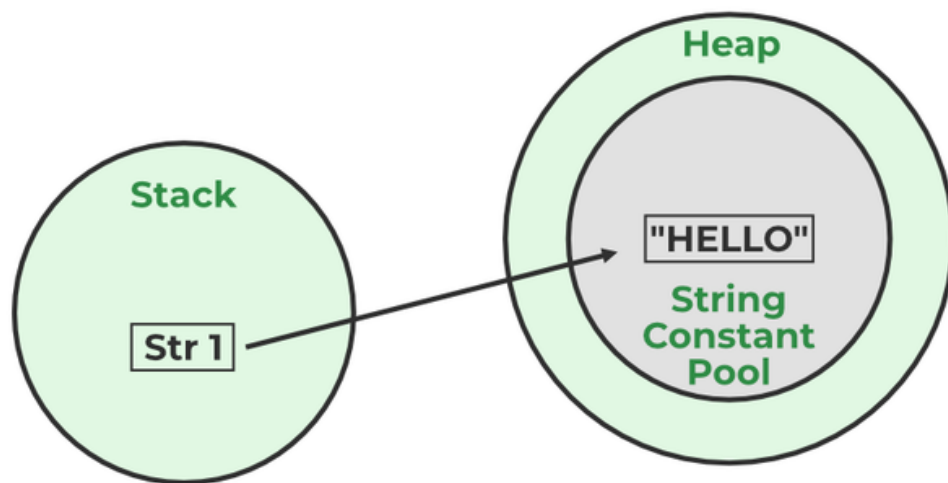
Unlike any other programming language like C, C++, etc. In Java, we declared the main function as a public static void main (String args[]). The meanings of the terms are mentioned below:

1. **public**: the public is the access modifier responsible for mentioning who can access the element or the method and what is the limit. It is responsible for making the main function globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.
2. **static**: static is a keyword used so that we can use the element without initiating the class so to avoid the unnecessary allocation of the memory.
3. **void**: void is a keyword and is used to specify that a method doesn't return anything. As the main function doesn't return anything we use void.

4. **main:** main represents that the function declared is the main function. It helps JVM to identify that the declared function is the main function.
5. **String args[]:** It stores Java command-line arguments and is an array of type `java.lang.String` class.

10. What is Java String Pool?

A Java String Pool is a place in heap memory where all the strings defined in the program are stored. A separate place in a stack is there where the variable storing the string is stored. Whenever we create a new string object, JVM checks for the presence of the object in the String pool, If String is available in the pool, the same object reference is shared with the variable, else a new object is created.



Example:

```
String str1="Hello";  
// "Hello" will be stored in String Pool  
// str1 will be stored in stack memory
```

11. What will happen if we declare don't declare the main as static?

We can declare the main method without using static and without getting any errors. But, the main method will not be treated as the entry point to the application or the program.

12. What are Packages in Java?

Packages in Java can be defined as the grouping of related types of classes, interfaces, etc providing access to protection and namespace management.

13. Why Packages are used?

Packages are used in Java in order to prevent naming conflicts, control access, and make searching/locating and usage of classes, interfaces, etc easier.

14. What are the advantages of Packages in Java?

There are various advantages of defining packages in Java.

- Packages avoid name clashes.
- The Package provides easier access control.

- We can also have the hidden classes that are not visible outside and are used by the package.
- It is easier to locate the related classes.

15. How many types of packages are there in Java?

There are two types of packages in Java

- User-defined packages
- Build In packages

16. Explain different data types in Java.

There are 2 types of data types in Java as mentioned below:

1. Primitive Data Type
2. Non-Primitive Data Type or Object Data type

Primitive Data Type: Primitive data are single values with no special capabilities. There are 8 primitive data types:

- **boolean:** stores value true or false
- **byte:** stores an 8-bit signed two's complement integer
- **char:** stores a single 16-bit Unicode character
- **short:** stores a 16-bit signed two's complement integer
- **int:** stores a 32-bit signed two's complement integer
- **long:** stores a 64-bit two's complement integer
- **float:** stores a single-precision 32-bit IEEE 754 floating-point
- **double:** stores a double-precision 64-bit IEEE 754 floating-point

Non-Primitive Data Type: Reference Data types will contain a memory address of the variable's values because it is not able to directly store the values in the memory. Types of Non-Primitive are mentioned below:

- Strings
- Array
- Class
- Object
- Interface

17. When a byte datatype is used?

A byte is an 8-bit signed two-complement integer. The minimum value supported by bytes is -128 and 127 is the maximum value. It is used in conditions where we need to save memory and the limit of numbers needed is between -128 to 127.

18. Can we declare Pointer in Java?

No, Java doesn't provide the support of Pointer. As Java needed to be more secure because which feature of the pointer is not provided in Java.

19. What is the default value of byte datatype in Java?

The default value of the byte datatype in Java is 0.

20. What is the default value of float and double datatype in Java?

The default value of the float is 0.0f and of double is 0.0d in Java.

21. What is the Wrapper class in Java?

Wrapper, in general, is referred to a larger entity that encapsulates a smaller entity. Here in Java, the wrapper class is an object class that encapsulates the primitive data types.

The primitive data types are the ones from which further data types could be created. For example, integers can further lead to the construction of long, byte, short, etc. On the other hand, the string cannot, hence it is not primitive.

Getting back to the wrapper class, Java contains 8 wrapper classes. They are Boolean, Byte, Short, Integer, Character, Long, Float, and Double. Further, custom wrapper classes can also be created in Java which is similar to the concept of Structure in the C programming language. We create our own wrapper class with the required data types.

22. Why do we need wrapper classes?

The wrapper class is an object class that encapsulates the primitive data types, and we need them for the following reasons:

1. Wrapper classes are final and immutable
2. Provides methods like `valueOf()`, `parseInt()`, etc.
3. It provides the feature of autoboxing and unboxing.

23. Differentiate between instance and local variables.

Instance Variable	Local Variable
Declared outside the method, directly invoked by the method.	Declared within the method.
Has a default value.	No default value
It can be used throughout the class.	The scope is limited to the method.

24. What are the default values assigned to variables and instances in Java?

In Java When we haven't initialized the instance variables then the compiler initializes them with default values. The default values for instances and variables depend on their data types. Some common types of default data types are:

- The default value for numeric types (byte, short, int, long, float, and double) is 0.
- The default value for the boolean type is false.
- The default value for object types (classes, interfaces, and arrays) is null.
- The null character, "u0000," is the default value for the char type.

Example:

Java

// Java Program to demonstrate use of default values

```
import java.io.*;
class GFG {
    // static values
    static byte b;
    static int i;
    static long l;
    static short s;
    static boolean bool;
```



```

static char c;
static String str;
static Object object;
static float f;
static double d;
static int[] Arr;
public static void main(String[] args)
{
    // byte value
    System.out.println("byte value" + b);
    // short value
    System.out.println("short value" + s);
    // int value
    System.out.println("int value" + i);
    // long value
    System.out.println("long value" + l);
    System.out.println("boolean value" + bool);
    System.out.println("char value" + c);
    System.out.println("float value" + f);
    System.out.println("double value" + d);
    System.out.println("string value" + str);
    System.out.println("object value" + object);
    System.out.println("Array value" + Arr);
}
}

```

Output:

```

byte value0
short value0
int value0
long value0
boolean valuefalse
char value
float value0.0
double value0.0
string valuenull
object valuenull
Array valuenull

```

25. What is a Class Variable?

In Java, a class variable (also known as a static variable) is a variable that is declared within a class but outside of any method, constructor, or block. Class variables are declared with the static keyword, and they are shared by all instances (objects) of the class as well as by the class itself. No matter how many objects are derived from a class, each class variable would only exist once.

Example:

Java

// Java program to demonstrate use of Class Variable

```
class GFG {  
    public static int ctr = 0;  
    public GFG() { ctr++; }  
    public static void main(String[] args)  
    {  
        GFG obj1 = new GFG();  
        GFG obj2 = new GFG();  
        GFG obj3 = new GFG();  
        System.out.println("Number of objects created are "  
            + GFG.ctr);  
    }  
}
```

Output

Number of objects created are 3

26. What is the default value stored in Local Variables?

There is no default value stored with local variables. Also, primitive variables and objects don't have any default values.

27. Explain the difference between instance variable and a class variable.

Instance Variable: A class variable without a static modifier known as an instance variable is typically shared by all instances of the class. These variables can have distinct values among several objects. The contents of an instance variable are completely independent of one object instance from another because they are related to a specific object instance of the class.

Example:

Java

// Java Program to demonstrate Instance Variable

```
import java.io.*;  
class GFG {  
    private String name;  
    public void setName(String name) { this.name = name; }  
    public String getName() { return name; }  
    public static void main(String[] args)  
    {  
        GFG obj = new GFG();  
        obj.setName("John");  
        System.out.println("Name " + obj.getName());  
    }  
}
```

Output

Name John

Class Variable: Class Variable variable can be declared anywhere at the class level using the keyword static. These variables can only have one value when applied to various objects. These variables can be shared by all class members since they are not connected to any specific object of the class.

Example:

Java

// Java Program to demonstrate Class Variable

```
import java.io.*;
class GFG {
    // class variable
    private static final double PI = 3.14159;
    private double radius;
    public GFG(double radius) { this.radius = radius; }
    public double getArea() { return PI * radius * radius; }
    public static void main(String[] args)
    {
        GFG obj = new GFG(5.0);
        System.out.println("Area of circle: "
            + obj.getArea());
    }
}
```

Output

Area of circle: 78.53975

28. What is a static variable?

The static keyword is used to share the same variable or method of a given class. Static variables are the variables that once declared then a single copy of the variable is created and shared among all objects at the class level.

29. What is the difference between System.out, System.err, and System.in?

System.out – It is a PrintStream that is used for writing characters or can be said it can output the data we want to write on the Command Line Interface console/terminal.

Example:

Java

// Java Program to implement

// System.out

```
import java.io.*;
```

// Driver Class

```
class GFG {
    // Main Function
    public static void main(String[] args)
    {
        // Use of System.out
        System.out.println("");
    }
}
```

```
}
```

System.err – It is used to display error messages.

Example:

Java

// Java program to demonstrate

// System.err

import java.io.*;

// Driver Class

class GFG {

// Main function

public static void main(String[] args)

{

// Printing error

System.err.println(

"This is how we throw error with System.err");

}

}

Output:

This is how we throw error with System.err

Although, System.err have many similarities both of them have quite a lot of difference also, let us check them.

System.out	System.err
It will print to the standard out of the system.	It will print to the standard error.
It is mostly used to display results on the console.	It is mostly used to output error texts.
It gives output on the console with the default(black) color.	It also gives output on the console but most of the IDEs give it a red color to differentiate.

System.in – It is an InputStream used to read input from the terminal Window. We can't use the System.in directly so we use Scanner class for taking input with the system.in.

Example:

Java

// Java Program to demonstrate

// System.in

import java.util.*;

// Driver Class

```

class Main {
    // Main Function
    public static void main(String[] args)
    {
        // Scanner class with System.in
        Scanner sc = new Scanner(System.in);
        // Taking input from the user
        int x = sc.nextInt();
        int y = sc.nextInt();
        // Printing the output
        System.out.printf("Addition: %d", x + y);
    }
}

```

Output:

3

4

Addition: 7

30. What do you understand by an IO stream?



Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all types of objects, data types, characters, files, etc to fully execute the I/O operations.

31. What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The key difference between them is that byte stream data is read and written by input/output stream classes. Characters are handled by the Reader and Writer classes. In contrast to Reader/Writer classes, which accept character arrays as parameters, input/output stream class methods accept byte arrays. In comparison to input/output streams, the Reader/Writer classes are more efficient, handle all Unicode characters, and are useful for internationalization. Use Reader/Writer classes instead of binary data, such as pictures, unless you do so.

Example:

Java

// Java Program to demonstrate Reading Writing Binary Data

// with InputStream/OutputStream

```
import java.io.*;
```

```

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

```

```

try {
    // Writing binary data to a file using OutputStream
    byte[] data = {(byte) 0xe0, 0x4f, (byte) 0xd0, 0x20, (byte) 0xea};
    OutputStream os = new FileOutputStream("data.bin");
    os.write(data);
    os.close();

    // Reading binary data from a file using InputStream
    InputStream is = new FileInputStream("data.bin");
    byte[] newData = new byte[5];

    is.read(newData);
    is.close();

    // Printing the read data
    for (byte b : newData) {
        System.out.print(b+" ");
    }
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

Output

-32 79 -48 32 -22

32. What are the super most classes for all the streams?

All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

33. What are the FileInputStream and FileOutputStream?

To read and write data, Java offers I/O Streams. A Stream represents an input source or an output destination, which could be a file, an i/o device, another program, etc. [FileInputStream](#) in Java is used to read data from a file as a stream of bytes. It is mostly used for reading binary data such as images, audio files, or serialized objects.

Example:

```

File file = new File("path_of_the_file");
FileInputStream inputStream = new FileInputStream(file);

```

In Java, the [FileOutputStream](#) function is used to write data byte by byte into a given file or file descriptor. Usually, raw byte data, such as pictures, is written into a file using `FileOutputStream`.

Example:

```
File file = new File("path_of_the_file");
FileOutputStream outputStream = new FileOutputStream(file);
```

34. What is the purpose of using `BufferedInputStream` and `BufferedOutputStream` classes?

When we are working with the files or stream then to increase the Input/Output performance of the program we need to use the `BufferedInputStream` and `BufferedOutputStream` classes. These both classes provide the capability of buffering which means that the data will be stored in a buffer before writing to a file or reading it from a stream. It also reduces the number of times our OS needs to interact with the network or the disk. Buffering allows programs to write a big amount of data instead of writing it in small chunks. This also reduces the overhead of accessing the network or the disk.

```
BufferedInputStream(InputStream inp);
// used to create the bufferedinput stream and save the arguments.
BufferedOutputStream(OutputStream output);
// used to create a new buffer with the default size.
```

35. What are `FilterStreams`?

Stream filter or Filter Streams returns a stream consisting of the elements of this stream that match the given predicate. While working `filter()` it doesn't actually perform filtering but instead creates a new stream that, when traversed, contains the elements of initial streams that match the given predicate.

Example:

```
FileInputStream fis = new FileInoutStream("file_path");
FilterInputStream = new BufferedInputStream(fis);
```

36. What is an I/O filter?

An I/O filter also defined as an Input Output filter is an object that reads from one stream and writes data to input and output sources. It used `java.io` package to use this filter.

37. How many ways you can take input from the console?

There are two methods to take input from the console in Java mentioned below:

1. Using Command line argument
2. Using Buffered Reader Class
3. Using Console Class
4. Using Scanner Class

The program demonstrating the use of each method is given below.

Example:

```
Java
// Java Program to implement input
// using Command line argument
import java.io.*;
class GFG {
    public static void main(String[] args)
```

```

{
    // check if length of args array is
    // greater than 0
    if (args.length > 0) {
        System.out.println(
            "The command line arguments are:");
        // iterating the args array and printing
        // the command line arguments
        for (String val : args)
            System.out.println(val);
    }
    else
        System.out.println("No command line "
            + "arguments found.");
}
}
// Use below commands to run the code
// javac GFG.java
// java Main GeeksforGeeks
Java
// Java Program to implement
// Buffer Reader Class
import java.io.*;
class GFG {
    public static void main(String[] args)
        throws IOException
    {
        // Enter data using BufferedReader
        BufferedReader read = new BufferedReader(
            new InputStreamReader(System.in));
        // Reading data using readLine
        String x = read.readLine();
        // Printing the read line
        System.out.println(x);
    }
}

```

Output

```

null
Java
// Java program to implement input
// Using Console Class
public class GfG {
    public static void main(String[] args)

```



```

{
    // Using Console to input data from user
    String x = System.console().readLine();
    System.out.println("You entered string " + x);
}
}
Java
// Java program to demonstrate
// working of Scanner in Java
import java.util.Scanner;
class GfG {
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);
        String str = in.nextLine();
        System.out.println("You entered string " + str);
    }
}

```

Output:

GeeksforGeeks

38. Difference in the use of print, println, and printf.

print, println, and printf all are used for printing the elements but print prints all the elements and the cursor remains in the same line. println shifts the cursor to next line. And with printf we can use format identifiers too.

39. What are operators?

Operators are the special types of symbols used for performing some operations over variables and values.

40. How many types of operators are available in Java?

All types of operators in Java are mentioned below:

1. [Arithmetic Operators](#)
2. [Unary Operators](#)
3. [Assignment Operator](#)
4. [Relational Operators](#)
5. [Logical Operators](#)
6. [Ternary Operator](#)
7. [Bitwise Operators](#)
8. [Shift Operators](#)
9. [instance of operator](#)

Postfix operators are considered as the highest precedence according to Java operator precedence.

41. Explain the difference between >> and >>> operators.

Operators like >> and >>> seem to be the same but act a bit differently. >> operator shifts the sign bits and the >>> operator is used in shifting out the zero-filled bits.

Example:

Java

```
import java.io.*;
```

```
// Driver
```

```
class GFG {  
    public static void main(String[] args) {  
        int a = -16, b = 1;  
        // Use of >>  
        System.out.println(a >> b);  
        a = -17;  
        b = 1;  
        // Use of >>>  
        System.out.println(a >>> b);  
    }  
}
```

Output

-8

2147483639

42. Which Java operator is right associative?

There is only one operator which is right associative which is = operator.

43. What is dot operator?

The Dot operator in Java is used to access the instance variables and methods of class objects. It is also used to access classes and sub-packages from the package.

44. What is covariant return type?

The covariant return type specifies that the return type may vary in the same direction as the subclass. It's possible to have different return types for an overriding method in the child class, but the child's return type should be a subtype of the parent's return type and because of that overriding method becomes variant with respect to the return type.

We use covariant return type because of the following reasons:

- Avoids confusing type casts present in the class hierarchy and makes the code readable, usable, and maintainable.
- Gives liberty to have more specific return types when overriding methods.
- Help in preventing run-time ClassCastExceptions on returns.

45. What is the transient keyword?

The transient keyword is used at the time of serialization if we don't want to save the value of a particular variable in a file. When JVM comes across a transient keyword, it ignores the original value of the variable and saves the default value of that variable data type.

46. What's the difference between the methods sleep() and wait()?

Sleep()	Wait()
The sleep() method belongs to the thread class.	Wait() method belongs to the object class.
Sleep does not release the lock that the current thread holds.	wait() release the lock which allows other threads to acquire it.
This method is a static method.	This method is not a static method.
Sleep() does not throw an InterruptedException.	InterruptedException is shown if the thread is interrupted while waiting.
Mainly used to delay a thread for some specific time duration.	Mainly used to pause a thread until notified by another thread.
Sleep() Has Two Overloaded Methods: <ul style="list-style-type: none"> • sleep(long millis) millis: milliseconds • sleep(long millis, int nanos) nanos: Nanoseconds 	Wait() Has Three Overloaded Methods: <ul style="list-style-type: none"> • wait() • wait(long timeout) • wait(long timeout, int nanos)

47. What are the differences between String and StringBuffer?

String	StringBuffer
Store of a sequence of characters.	Provides functionality to work with the strings.
It is immutable.	It is mutable (can be modified and other string operations could be performed on them.)
No thread operations in a string.	It is thread-safe (two threads can't call the methods of StringBuffer simultaneously)

48. What are the differences between StringBuffer and StringBuilder?

StringBuffer	StringBuilder
StringBuffer provides functionality to work with the strings.	StringBuilder is a class used to build a mutable string.
It is thread-safe (two threads can't call the methods of StringBuffer simultaneously)	It is not thread-safe (two threads can call the methods concurrently)
Comparatively slow as it is synchronized.	Being non-synchronized, implementation is faster

49. Which among String or String Buffer should be preferred when there are a lot of updates required to be done in the data?

The string is preferred over StringBuffer as StringBuilder is faster than StringBuffer, but StringBuffer objects are the preferred over as it provides more thread safety.

50. Why is StringBuffer called mutable?

StringBuffer class in Java is used to represent a changeable string of characters. It offers an alternative to the immutable String class by enabling you to change a string's contents without constantly creating new objects. Mutable (modifiable) strings are created with the help of the StringBuffer class. The StringBuffer class in Java is identical to the String class except that it is changeable.

Example:

Java

// Java Program to demonstrate use of stringbuffer

```
public class StringBufferExample {
    public static void main(String[] args)
    {
        StringBuffer s = new StringBuffer();
        s.append("Geeks");
        s.append("for");
        s.append("Geeks");
        String message = s.toString();
        System.out.println(message);
    }
}
```

Output

GeeksforGeeks

51. How is the creation of a String using new() different from that of a literal?

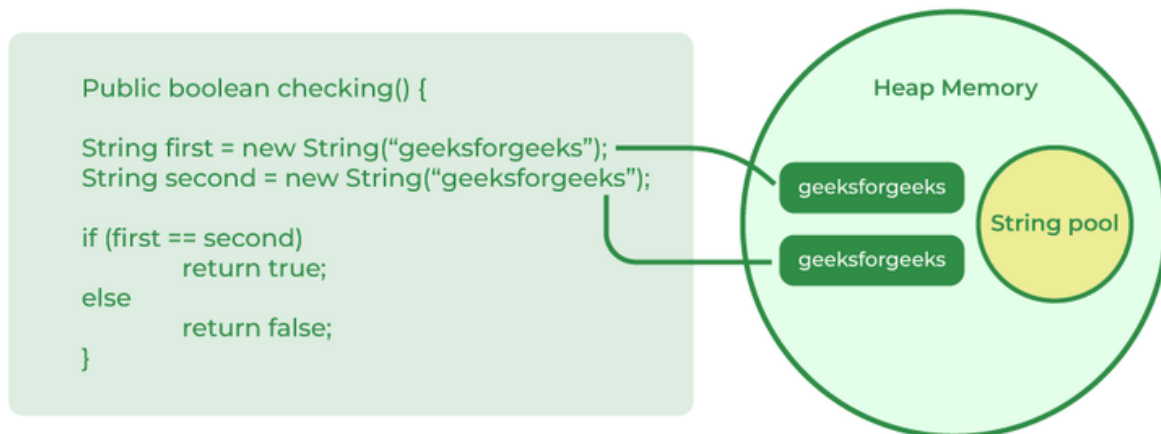
String using new() is different from the literal as when we declare string it stores the elements inside the stack memory whereas when it is declared using new() it allocates a

dynamic memory in the heap memory. The object gets created in the heap memory even if the same content object is present.

Syntax:

```
String x = new String("ABC");
```

String pool by means of new operator



52. What is an array in Java?

An Array in Java is a data structure that is used to store a fixed-size sequence of elements of the same type. Elements of an array can be accessed by their index, which starts from 0 and goes up to a length of minus 1. Array declaration in Java is done with the help of square brackets and size is also specified during the declaration.

Syntax:

```
int[] Arr = new int[5];
```

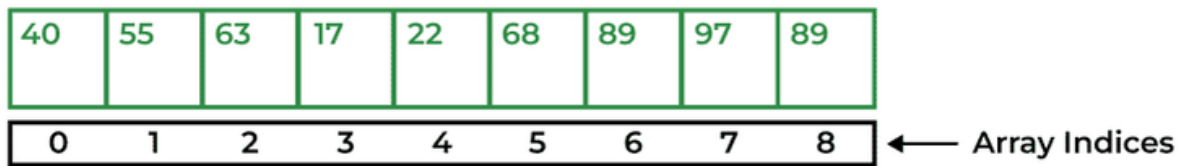
53. On which memory arrays are created in Java?

Arrays in Java are created in heap memory. When an array is created with the help of a new keyword, memory is allocated in the heap to store the elements of the array. In Java, the heap memory is managed by the Java Virtual Machine(JVM) and it is also shared between all threads of the Java Program. The memory which is no longer in use by the program, JVM uses a garbage collector to reclaim the memory. Arrays in Java are created dynamically which means the size of the array is determined during the runtime of the program. The size of the array is specified during the declaration of the array and it cannot be changed once the array is created.

54. What are the types of an array?

There are two types of arrays i.e., Primitive arrays and References Arrays.

- **Single-Dimensional Arrays:** Arrays that have only one dimension i.e., an array of integers or an array of strings are known as single-dimensional arrays.



Array Length = 9

First Index = 0

Last Index = 8

Syntax:

```
data_type[] Array_Name = new data_type[ArraySize];
```

- **Multi-Dimensional Arrays:** Arrays that have two or more dimensions such as two-dimensional or three-dimensional arrays.

55. Why does the Java array index start with 0?

The index of an array signifies the distance from the start of the array. So, the first element has 0 distance therefore the starting index is 0.

Syntax:

```
[Base Address + (index * no_of_bytes)]
```

56. What is the difference between `int array[]` and `int[] array`?

Both `int array[]` and `int[] array` are used to declare an array of integers in java. The only difference between them is on their syntax no functionality difference is present between them.

`int arr[]` is a C-Style syntax to declare an Array.

`int[] arr` is a Java-Style syntax to declare an Array.

However, it is generally recommended to use Java-style syntax to declare an Array. As it is easy to read and understand also it is more consistent with other Java language constructs.

57. How to copy an array in Java?

In Java there are multiple ways to copy an Array based on the requirements.

- **clone() method in Java:** This method in Java is used to create a shallow copy of the given array which means that the new array will share the same memory as the original array.

```
int[] Arr = { 1, 2, 3, 5, 0};
```

```
int[] tempArr = Arr.clone();
```

- **arraycopy() method:** To create a deep copy of the array we can use this method which creates a new array with the same values as the original array.

```
int[] Arr = {1, 2, 7, 9, 8};
```

```
int[] tempArr = new int[Arr.length];
```

```
System.arraycopy(Arr, 0, tempArr, 0, Arr.length);
```

- **copyOf() method:** This method is used to create a new array with a specific length and copies the contents of the original array to the new array.

```
int[] Arr = {1, 2, 4, 8};
```

```
int[] tempArr = Arrays.copyOf(Arr, Arr.length);
```

- **copyOfRange() method:** This method is very similar to the copyOf() method in Java, but this method also allows us to specify the range of the elements to copy from the original array.

```
int[] Arr = {1, 2, 4, 8};
```

```
int[] temArr = Arrays.copyOfRange(Arr, 0, Arr.length);
```

58. What do you understand by the jagged array?

A jagged Array in Java is just a two-dimensional array in which each row of the array can have a different length. Since all the rows in a 2-d Array have the same length but a jagged array allows more flexibility in the size of each row. This feature is very useful in conditions where the data has varying lengths or when memory usage needs to be optimized.

Syntax:

```
int[][] Arr = new int[][] {
    {1, 2, 8},
    {7, 5},
    {6, 7, 2, 6}
};
```

59. Is it possible to make an array volatile?

In Java, it is not possible to make a volatile. Volatile keywords in Java can only be applied to individual variables but not to arrays or collections. The value of the Variable is always read from and written to the main memory when it is defined as volatile rather than being cached in a thread's local memory. This makes it easier to make sure that all threads that access the variable can see changes made to it.

60. What are the advantages and disadvantages of an array?

ARRAY								
Indices	0	1	2	3	4	5	6	7
	2	10	5	16	20	6		
Memory location	100	101	102	103	104	105	106	107

The advantages of Arrays are:

- Direct and effective access to any element in the collection is made possible by arrays. An array's elements can be accessed using an $O(1)$ operation, which means that the amount of time needed to do so is constant and independent of the array's size.
- Data can be stored effectively in memory using arrays. The size of an array is known at compile time since its elements are stored in contiguous memory regions.
- Due to the fact that the data is stored in contiguous memory areas, arrays provide quick data retrieval.
- Arrays are easy to implement and understand, making them an ideal choice for beginners learning computer programming.

Disadvantages of Arrays are:

- Arrays are created with a predetermined size that is chosen at that moment. This means that if the array's size needs to be extended, a new array will need to be made, and the data will need to be copied from the old array to the new array, which can take a lot of time and memory.
- There may be unused memory space in an array's memory space if the array is not completely occupied. If you have poor recall, this can be a problem.
- Compared to other data structures like linked lists and trees, arrays might be rigid due to their fixed size and limited support for sophisticated data types.
- Because an array's elements must all be of the same data type, it does not support complex data types like objects and structures.

61. What is an object-oriented paradigm?

Paradigm literally means a pattern or a method. Programming paradigms are the methods to solve a program that is of four types namely, Imperative, logical, functional, and object-oriented. When objects are used as base entities upon which the methods are applied, encapsulation or inheritance functionalities are performed, it is known as an object-oriented paradigm.

62. What are the main concepts of OOPs in Java?

The main concepts of OOPs in Java are mentioned below:

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

63. What is the difference between an object-oriented programming language and an object-based programming language?

Object-Oriented Programming Language	Object-Based Programming Language
Object-oriented programming language covers larger concepts like inheritance, polymorphism, abstraction, etc.	The scope of object-based programming is limited to the usage of objects and encapsulation.
It supports all the built-in objects	It doesn't support all the built-in objects
Examples: Java, C#, etc.	Examples: Java script, visual basics, etc.

64. How is the 'new' operator different from the 'newInstance()' operator in Java?

the new operator is used to create objects, but if we want to decide the type of object to be created at runtime, there is no way we can use the new operator. In this case, we have to use the [newInstance\(\) method](#).

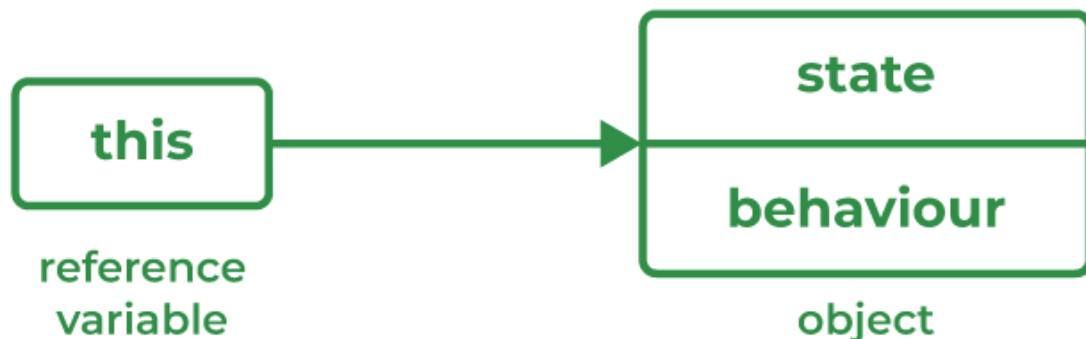
65. What are Classes in Java?

In Java, Classes are the collection of objects sharing similar characteristics and attributes. Classes represent the blueprint or template from which objects are created. Classes are not real-world entities but help us to create objects which are real-world entities.

66. What is the difference between static (class) method and instance method?

Static(Class) method	Instance method
Static method is associated with a class rather than an object.	The instance method is associated with an object rather than a class.
Static methods can be called using the class name only without creating an instance of a class.	The instance methods can be called on a specific instance of a class using the object reference.
Static methods do not have access to this keyword.	Instance methods have access to this keyword.
Static methods can access only static members of the class.	Instance methods can access both static and non-static methods of the class.
Static methods cannot be overridden because they are resolved at compile time, not at run time. This means that the compiler decides which method to call based on the reference type, not on the object type.	Instance methods can be overridden because they are resolved at run time, not at compile time. This means that the compiler decides which method to call based on the object type, not on the reference type.

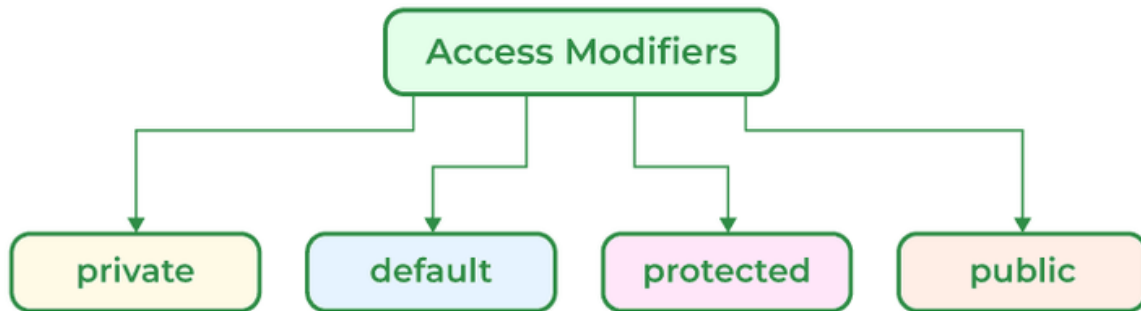
67. What is this keyword in Java?



‘this’ is a keyword used to reference a variable that refers to the current object.

68. What are Brief Access Specifiers and Types of Access Specifiers?

Access Modifiers in Java



Access Specifiers in Java help to restrict the scope of a class, constructor, variable, method, or data member. There are four types of Access Specifiers in Java mentioned below:

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

69. What will be the initial value of an object reference which is defined as an instance variable?

The initial value of an object reference which is defined as an instance variable is a NULL value.

70. What is an object?

The object is a real-life entity that has certain properties and methods associated with it. The object is also defined as the instance of a class. An object can be declared using a new keyword.

71. What are the different ways to create objects in Java?

Methods to create objects in Java are mentioned below:

1. Using new keyword
2. Using new instance
3. Using clone() method
4. Using deserialization
5. Using the newInstance() method of the Constructor class

To know more about methods to create objects in Java refer to [this article](#).

72. What are the advantages and disadvantages of object cloning?

There are many advantages and disadvantages of using object cloning as mentioned below:

Advantages:

- In Java, the '=' assignment operator cannot be used for cloning as it simply creates a copy of reference variables. To overcome such discrepancy the clone() method of Object class can be used over the assignment operator.

- The clone() method is a protected method of class Object which means that only the Employee class can clone Employee objects. This means no class other than Employee can clone Employee objects since it does not know the Employee class' attributes.
- Code size decreases as repetition decreases.
- Allows replication (kind of like prototype pattern) manually initializing each field creates large code if object is complex, its faster with cloning.

Disadvantages:

- As the Object.clone() method is protected, so need to provide our own clone() and indirectly call Object.clone() from it.
- If we don't have any methods then we need to provide a Cloneable interface as we need to provide JVM information so that we can perform a clone() on our object. if not, we cant clone clone does shallow copy of fields if we just return super.clone() from clone method that can be problematic.

73. What are the advantages of passing this into a method instead of the current class object itself?

There are a few advantages of passing this into a method instead of the current class object itself these are:

- this is the final variable because of which this cannot be assigned to any new value whereas the current class object might not be final and can be changed.
- this can be used in the synchronized block.

74. What is the constructor?

Constructor is a special method that is used to initialize objects. Constructor is called when a object is created. The name of constructor is same as of the class.

Example:

```
// Class Created
class XYZ{
    private int val;

    // Constructor
    XYZ(){
        val=0;
    }
};
```

75. What happens if you don't provide a constructor in a class?

If you don't provide a constructor in a class in Java, the compiler automatically generates a default constructor with no arguments and no operation which is a default constructor.

76. How many types of constructors are used in Java?

There are two types of constructors in Java as mentioned below:

1. Default Constructor
2. Parameterized Constructor

Default Constructor: It is the type that does not accept any parameter value. It is used to set initial values for object attributes.

```
class_Name();
```

```
// Default constructor called
```

Parameterized Constructor: It is the type of constructor that accepts parameters as arguments. These are used to assign values to instance variables during the initialization of objects.

```
class_Name(parameter1, parameter2.....);
```

```
// All the values passed as parameter will be
```

```
// allocated accordingly
```

77. What is the purpose of a default constructor?

Constructors help to create instances of a class or can be said to create objects of a class. Constructor is called during the initialization of objects. A default constructor is a type of constructor which do not accept any parameter, So whatever value is assigned to properties of the objects are considered default values.

78. What do you understand by copy constructor in Java?

The copy constructor is the type of constructor in which we pass another object as a parameter because which properties of both objects seem the same, that is why it seems as if constructors create a copy of an object.

79. Where and how can you use a private constructor?

A private constructor is used if you don't want any other class to instantiate the object to avoid subclassing. The use private constructor can be seen as implemented in the example.

Example:

Java

```
// Java program to demonstrate implementation of Singleton
```

```
// pattern using private constructors.
```

```
import java.io.*;
```

```
class GFG {
```

```
    static GFG instance = null;
```

```
    public int x = 10;
```

```
    // private constructor can't be accessed outside the
```

```
    // class
```

```
    private GFG() {}
```

```
    // Factory method to provide the users with instances
```

```
    static public GFG getInstance()
```

```
    {
```

```
        if (instance == null)
```

```
            instance = new GFG();
```

```
        return instance;
```

```
    }
```

```
}
```

```
// Driver Class
```

```
class Main {
```

```
    public static void main(String args[])
```

```
    {
```

```

    GFG a = GFG.getInstance();
    GFG b = GFG.getInstance();
    a.x = a.x + 10;
    System.out.println("Value of a.x = " + a.x);
    System.out.println("Value of b.x = " + b.x);
}
}

```

Output

Value of a.x = 20

Value of b.x = 20

80. What are the differences between the constructors and methods?

Java constructors are used for initializing objects. During creation, constructors are called to set attributes for objects apart from this few basic differences between them are:

1. Constructors are only called when the object is created but other methods can be called multiple times during the life of an object.
2. Constructors do not have a return type, whereas methods have a return type, which can be void or any other type.
3. Constructors are used to setting up the initial state but methods are used to perform specific actions.

81. What is an Interface?

An interface in Java is a collection of static final variables and abstract methods that define the contract or agreement for a set of linked classes. Any class that implements an interface is required to implement a specific set of methods. It specifies the behavior that a class must exhibit but not the specifics of how it should be implemented.

Syntax:

```

interface
{
    // constant fields
    // methods that are abstract by default
}

```

Example:

```

Java
// Java Program to demonstrate Interface
import java.io.*;
interface Shape {
    double getArea();
    double getPerimeter();
}
class Circle implements Shape {
    private double radius;
    public Circle(double radius) { this.radius = radius; }
    public double getArea()
    {

```

```

        return Math.PI * radius * radius;
    }
    public double getPerimeter()
    {
        return 2 * Math.PI * radius;
    }
}
class GFG {
    public static void main(String[] args)
    {
        Circle circle = new Circle(5.0);
        System.out.println("Area of circle is "
            + circle.getArea());
        System.out.println("Perimeter of circle is"
            + circle.getPerimeter());
    }
}

```

Output

Area of circle is 78.53981633974483
 Perimeter of circle is 31.41592653589793

82. Give some features of the Interface.

An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.

Features of the Interface are mentioned below:

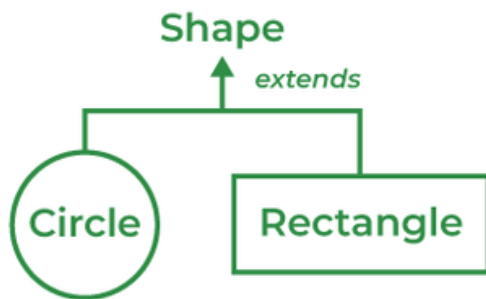
- The interface can help to achieve total abstraction.
- Allows us to use multiple inheritances in Java.
- Any class can implement multiple interfaces even when one class can extend only one class.
- It is also used to achieve loose coupling.

83. What is a marker interface?

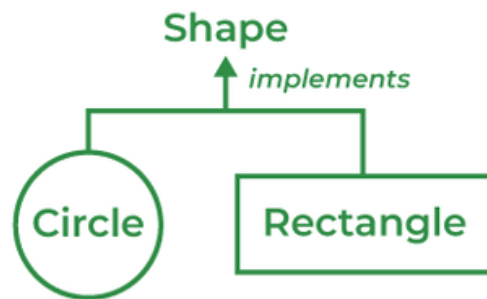
An Interface is recognized as an empty interface (no field or methods) it is called a marker interface. Examples of marker interfaces are Serializable, Cloneable, and Remote interfaces.

84. What are the differences between abstract class and interface?

Abstract Class



Interface



Abstract Class

Both abstract and non-abstract methods may be found in an abstract class.

Abstract Class supports Final methods.

Multiple inheritance is not supported by the Abstract class.

Abstract Keyword is used to declare Abstract class.

extend keyword is used to extend an Abstract Class.

Abstract Class has members like protected, private, etc.

Interface Class

The interface contains only abstract methods.

The interface class does not support Final methods.

Multiple inheritances is supported by Interface Class.

Interface Keyword is used to declare the interface class.

implements keyword is used to implement the interface.

All class members are public by default.

85. What do you mean by data encapsulation?

Encapsulation in JAVA



Data Encapsulation is the concept of OOPS properties and characteristics of the classes that The interface is binded together. Basically, it bundles data and methods that operate on that data within a single unit. Encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class.

86. What are the advantages of Encapsulation in Java?

The advantages of Encapsulation in Java are mentioned below:

1. Data Hiding: it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class.
2. Increased Flexibility: We can make the variables of the class read-only or write-only depending on our requirements.
3. Reusability: Encapsulation also improves the re-usability and is easy to change with new requirements.
4. Testing code is easy: Code is made easy to test for unit testing.

87. What is the primary benefit of Encapsulation?

The main advantage of Encapsulation in Java is its ability to protect the internal state of an object from external modification or access. It is the is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class. The main benefit is of providing a way to control and manage the state and the behavior of an object and also protecting it from modification and unauthorized access at the same time.

Example:

Java

// Java Program to demonstrate use of Encapsulation

```
import java.io.*;
```

```
class Person {
```

```
    private String Name;
```

```
    private int age;
```

```
    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; }
}
// Driver class
class GFG {
    // main function
    public static void main(String[] args)
    {
        Person p = new Person();
        p.setName("Rohan");
        p.setAge(29);
        System.out.println("Name is " + p.getName());
        System.out.println("Age is " + p.getAge());
    }
}

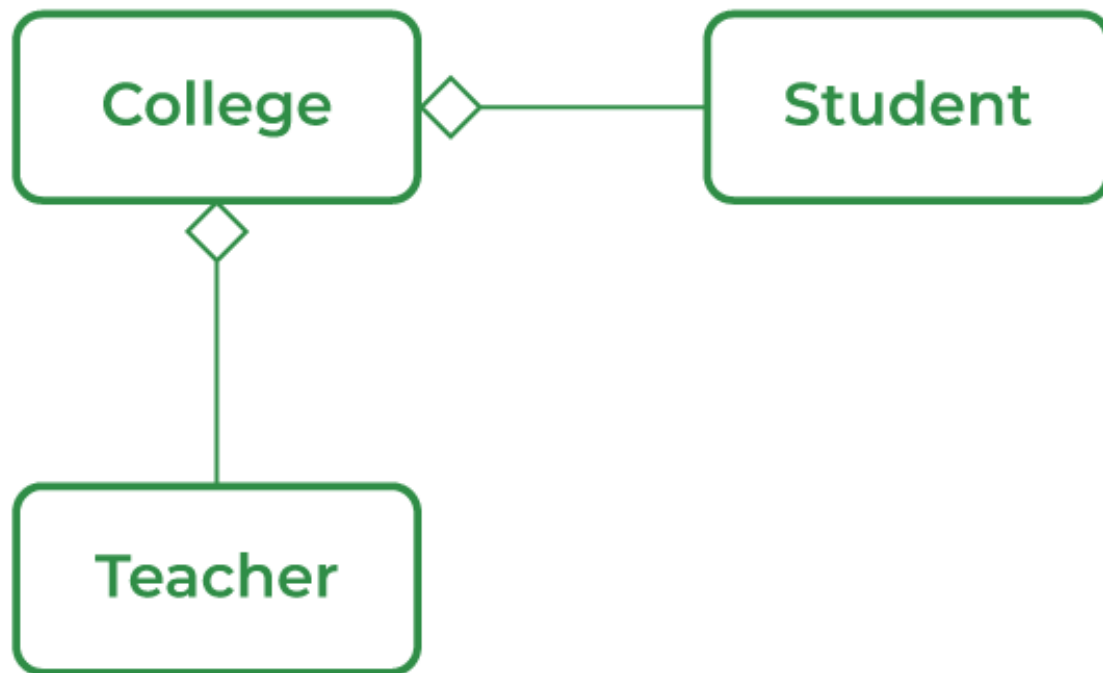
```

Output

Name is Rohan

Age is 29

88. What do you mean by aggregation?



Aggregation is a term related to the relationship between two classes best described as a “has-a” relationship. This kind is the most specialized version of association. It is a unidirectional association means it is a one-way relationship. It contains the reference to another class and is said to have ownership of that class.

89. What is the ‘IS-A ‘ relationship in OOPs Java?

'IS-A' is a type of relationship in OOPs Java where one class inherits another class.

90. Define Inheritance.

When an object that belongs to a subclass acquires all the properties and behavior of a parent object that is from the superclass, it is known as inheritance. A class within a class is called the subclass and the latter is referred to as the superclass. Sub class or the child class is said to be specific whereas the superclass or the parent class is generic.

Inheritance provides code reusability.

91. What are the different types of inheritance in Java?

Inheritance is the method by which the Child class can inherit the features of the Super or Parent class. In Java, Inheritance is of four types:

- **Single Inheritance:** When a child or subclass extends only one superclass, it is known to be single inheritance. Single-parent class properties are passed down to the child class.
- **Multilevel Inheritance:** When a child or subclass extends any other subclass a hierarchy of inheritance is created which is known as multilevel inheritance. In other words, one subclass becomes the parent class of another.
- **Hierarchical Inheritance:** When multiple subclasses derive from the same parent class is known as Hierarchical Inheritance. In other words, a class that has a single parent has many subclasses.
- **Multiple Inheritance:** When a child class inherits from multiple parent classes is known as Multiple Inheritance. In Java, it only supports multiple inheritance of interfaces, not classes.

92. What is multiple inheritance? Is it supported by Java?

A component of the object-oriented notion known as multiple inheritances allows a class to inherit properties from many parent classes. When methods with the same signature are present in both superclasses and subclasses, an issue arises. The method's caller cannot specify to the compiler which class method should be called or even which class method should be given precedence.

Note: Java doesn't support Multiple Inheritance

Example:

Java

// Java Program to show multiple Inheritance

```
import java.io.*;
interface Animal {
    void eat();
}
interface Mammal {
    void drink();
}
class Dog implements Animal, Mammal {
    public void eat() { System.out.println("Eating"); }
    public void drink() { System.out.println("Drinking"); }
    void bark() { System.out.println("Barking"); }
}
```

```

class GFG {
    public static void main(String[] args)
    {
        Dog d = new Dog();
        d.eat();
        d.drink();
        d.bark();
    }
}

```

Output

Eating
Drinking
Barking

93. How is inheritance in C++ different from Java?

Inheritance in C++	Inheritance in Java
C++ lets the user to inherit multiple classes.	Java doesn't support multiple inheritances.
When a class is created in C++, it doesn't inherit from the object class, instead exists on its own.	Java is always said to have a single inheritance as all the classes inherit in one or the other way from the object class.

94. Is there any limitation to using Inheritance?

Yes, there is a limitation of using Inheritance in Java, as because of inheritance one can inherit everything from super class and interface because of which subclass is too clustered and sometimes error-prone when dynamic overriding or dynamic overloading is done in certain situations.

95. Although inheritance is a popular OOPs concept, it is less advantageous than composition. Explain.

Inheritance is a popular concept of Object-Oriented Programming (OOP), in which a class can inherit the properties and methods from any other class, which is referred to as a Parent or superclass. On the other hand in Composition, a class can contain an instance of another class as a member variable which is often referred to as part or a component.

Below are some reasons why composition is more advantageous than inheritance:

- **Tight Coupling:** Whenever any changes are made to the superclass, these changes can affect the behavior of all its child or Subclasses. This problem makes code less flexible and also creates issues during maintenance. This problem also leads to the Tight coupling between the classes.
- **Fragile Base Class Problem:** When the changes to the base class can break the functionality of its derived classes. This problem can make it difficult to add new

features or modify the existing ones. This problem is known as the Fragile Base class problem.

- **Limited Reuse:** Inheritance in Java can lead to limited code reuse and also code duplication. As a subclass inherits all the properties and methods of its superclass, sometimes it may end up with unnecessary code which is not needed. This leads to a less maintainable codebase.

96. What is an association?

The association is a relation between two separate classes established through their Objects. It represents Has-A's relationship.

97. What do you mean by aggregation?

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other. It represents **part-of** the relationship.

98. What is the composition of Java?

Composition implies a relationship where the child **cannot exist independently** of the parent. For example Human heart, the heart doesn't exist separately from a Human.

99. State the difference between Composition and Aggregation.

Aggregation	Composition
It defines a "has a" relationship between the objects	It represents the part-of relationship
Objects are independent of each other.	Objects are dependent on each other.
Represent it by using the filled diamond.	Represent it by using the empty diamond.
Child objects don't have a lifetime.	Child objects have a lifetime.

100. Can the constructor be inherited?

No, we can't inherit a constructor.

101. What is Polymorphism?

Polymorphism is defined as the ability to take more than one form It is of two types namely, Compile time polymorphism or method overloading- a function called during compile time. For instance, take a class 'area'. Based on the number of parameters it may calculate the area of a square, triangle, or circle. Run time polymorphism or method overriding- links during run time. The method inside a class overrides the method of the parent class.

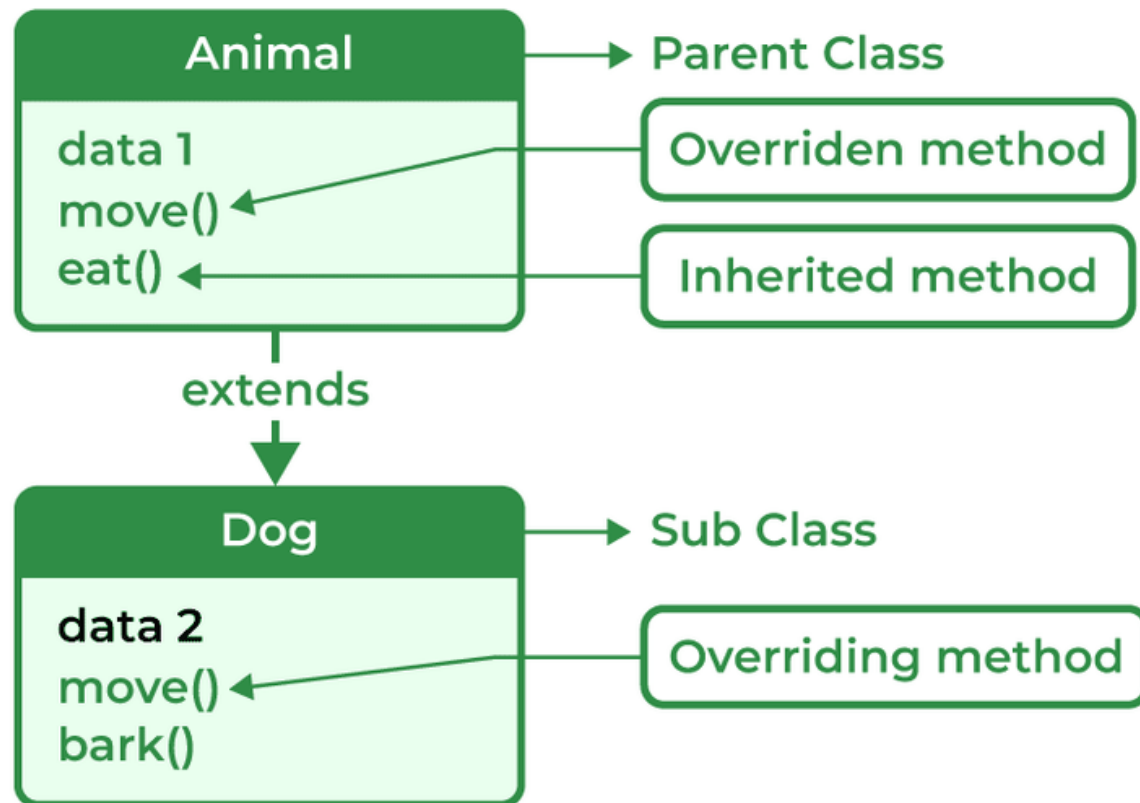
102. What is runtime polymorphism or dynamic method dispatch?

Dynamic method dispatch is a resolving mechanism for method overriding during the run time. Method overriding is the one where the method in a subclass has the same name, parameters, and return type as a method in the superclass. When the over-ridden method is called through a superclass reference, java determines which version (superclass or subclass) of that method is to be executed based upon the type of an object being referred

to at the time the call occurs. Thus the decision is made at run time. This is referred to as dynamic method dispatch.

103. What is method overriding?

Method overriding, also known as run time polymorphism is one where the child class contains the same method as the parent class. For instance, we have a method named 'gfg()' in the parent class. A method gfg() is again defined in the sub-class. Thus when gfg() is called in the subclass, the method within the class is executed. Here, gfg() within the class overrides the method outside.



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

104. What is method overloading?

In Java, Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading in Java is also known as **Compile-time Polymorphism**, **Static Polymorphism**, or **Early binding**. In Method overloading compared to the parent argument, the child argument will get the highest priority.

105. Can we override the static method?

No, as static methods are part of the class rather than the object so we can't override them.

106. Can we override the overloaded method?

Yes, since the overloaded method is a completely different method in the eyes of the compiler. Overriding isn't the same thing at all. The decision as to which method to call is deferred to runtime.

107. Can we overload the main() method?

Yes in Java we can overload the main method to call the main method with the help of its predefined calling method.

108. What are method overloading and method overriding?

Method Overloading: It is also known as Compile Time Polymorphism. In method overloading two or more methods are shared in the same class with a different signature.

Example:

Java

// Java Program to demonstrate use of Method Overloading

```
import java.io.*;
class GFG {
    static int multiply(int a, int b) { return a * b; }
    static int multiply(int a, int b, int c)
    {
        return a * b * c;
    }
    static int multiply(int a, int b, int c, int d)
    {
        return a * b * c * d;
    }
    public static void main(String[] args)
    {
        System.out.println("multiply() with 2 parameters");
        System.out.println(multiply(4, 5));
        System.out.println("multiply() with 3 parameters");
        System.out.println(multiply(2, 3, 4));
        System.out.println("multiply() with 4 parameters");
        System.out.println(multiply(2, 3, 4, 1));
    }
}
```

Output

multiply() with 2 parameters

20

multiply() with 3 parameters

24

multiply() with 4 parameters

Method Overriding: Method Overriding occurs when a subclass can provide the implementation of a method which is already defined in the parent class or superclass. The return type, name and arguments must be similar to the methods in superclass.

Example:

Java

// Java Program to demonstrate use of Method Overriding

```
import java.io.*;
class Vehicle {
    void drive()
    {
        System.out.println("drive() method of base class");
        System.out.println("driving the Car.");
    }
}
class Car extends Vehicle {
    void drive()
    {
        System.out.println(
            "drive() method of derived class");
        System.out.println("Car is driving.");
    }
}
class GFG {
    public static void main(String[] args)
    {
        Car c1 = new Car();
        Vehicle v1 = new Vehicle();
        c1.drive();
        v1.drive();
        Vehicle vehicle = new Car();
        // drive() method of Vehicle class is overridden by
        // Car class drive()
        vehicle.drive();
    }
}
```

Output

```
drive() method of derived class
Car is driving.
drive() method of base class
driving the Car.
drive() method of derived class
Car is driving.
```

Method Overloading	Method Overriding
When two or multiple methods are in the same class with different parameters but the same name.	When a subclass provides its own implementation of a method that is already defined in the parent class.
Method overloading can only happen in the same class or between a subclass or parent class.	Method overriding can only happen in Subclass.
When an error occurs it is caught at the compile time of the program.	When an error occurs it is caught at Runtime of the program.
Example of Compile Time Polymorphism.	Example of Run Time Polymorphism.
Method Overloading may or may not require Inheritance.	Method overriding always needs Inheritance.
It occurs within the class.	It is performed in two classes with an inheritance relationship.

109. Can we override the private methods?

It is not possible to override the private methods in Java. Method overriding is where the method in the subclass is implemented instead of the method from the parent class. The private methods are accessible only within the class in which it is declared. Since this method is not visible to other classes and cannot be accessed, it cannot be overridden.

110. Can we change the scope of the overridden method in the subclass?

In Java, it is not possible to modify the overridden method's scope. The subclass method's scope must be equal to or wider than the Superclass method's overridden method's scope. The overridden method in the subclass, for instance, can have a public scope or a more accessible scope like protected or default if the overridden method in the superclass has a public scope. It cannot, however, have a more exclusive scope like private.

111. Can we modify the throws clause of the superclass method while overriding it in the subclass?

We can modify the throws clause of the Superclass method with some limitations, we can change the throws clause of the superclass method while overriding it in the subclass. The subclass overridden method can only specify unchecked exceptions if the superclass method does not declare any exceptions. If the superclass method declares an exception, the subclass method can declare the same exception, a subclass exception, or no exception at all. However, the subclass method cannot declare a parent exception that is broader than the ones declared in the superclass method.

112. Can you have virtual functions in Java?

Yes, Java supports virtual functions. Functions are by default virtual and can be made non-virtual using the final keyword.

113. What is Abstraction?

Abstraction refers to the act of representing essential features without including background details. The detailed information or the implementation is hidden. The most common example of abstraction is a car, we know how to turn on the engine, accelerate and move, however, the way engine works, and its internal components are complex logic hidden from the general users. This is usually done to handle the complexity.

114. What is Abstract class?

A class declared as abstract, cannot be instantiated i.e., the object cannot be created. It may or may not contain abstract methods but if a class has at least one abstract method, it must be declared abstract.

Example of an abstract class with abstract method:

```
Java
// Java Program to implement
// abstract method
import java.io.*;
// Abstract class
abstract class Fruits {
    abstract void run();
}
// Driver Class
class Apple extends Fruits {
    void run()
    {
        System.out.println("Abstract class example");
    }
// main method
public static void main(String args[])
{
    Fruits obj = new Apple();
    obj.run();
}
}
```

Output

Abstract class example

115. When Abstract methods are used?

An abstract method is used when we want to use a method but want to child classes to decide the implementation in that case we use Abstract methods with the parent classes.

116. How can you avoid serialization in the child class if the base class is implementing the Serializable interface?

Serialization in the child class if the base class is implementing the Serializable interface then we can avoid it by defining the writeObject() method and throwing NotSerializableException().

117. What is Collection Framework in Java?

Collections are units of objects in Java. The collection framework is a set of interfaces and classes in Java that are used to represent and manipulate collections of objects in a variety of ways. The collection framework contains classes(ArrayList, Vector, LinkedList, PriorityQueue, TreeSet) and multiple interfaces (Set, List, Queue, Deque) where every interface is used to store a specific type of data.

118. Explain various interfaces used in the Collection framework.

Collection framework implements

1. Collection Interface
2. List Interface
3. Set Interface
4. Queue Interface
5. Deque Interface
6. Map Interface

Collection interface: Collection is the primary interface available that can be imported using java.util.Collection.

Syntax:

```
public interface Collection<E> extends Iterable
```

119. How can you synchronize an ArrayList in Java?

An ArrayList can be synchronized using two methods mentioned below:

1. Using Collections.synchronizedList()
2. Using CopyOnWriteArrayList

Using Collections.synchronizedList():

```
public static List<T> synchronizedList(List<T> list)
```

Using CopyOnWriteArrayList:

1. Create an empty List.
2. It implements the List interface
3. It is a thread-safe variant of ArrayList
4. T represents generic

120. Why do we need a synchronized ArrayList when we have Vectors (which are synchronized) in Java?

ArrayList is in need even when we have Vectors because of certain reasons:

1. ArrayList is faster than Vectors.
2. ArrayList supports multithreading whereas Vectors only supports single-thread use.
3. ArrayList is safer to use, as Vectors supports single threads and individual operations are less safe and take longer to synchronize.
4. Vectors are considered outdated in Java because of their synchronized nature.

121. Why can't we create a generic array?

Generic arrays can't be created because an **array** carries type information of its elements at runtime because of which during runtime it throw 'ArrayStoreException' if the elements'

type is not similar. Since generics type information gets erased at compile time by Type Erasure, the array store check would have been passed where it should have failed.

122. Can you explain how elements are stored in memory for both regular arrays and ArrayLists in Java? . Explain.

The elements of a regular array in Java are stored in contiguous memory locations, meaning that each element is stored in a sequential block of memory. This allows easy access to any element by its index because the address can be calculated using the base address of the array and the size of each element

In contrast, the ArrayList class implements a dynamic array, which means that its size can change as elements are added or removed. ArrayList elements are also stored in contiguous memory locations, similar to arrays. However, when an ArrayList reaches its capacity and more elements need to be added, a new, larger underlying array is created. The elements from the old array are then copied to the new one. This process ensures that the ArrayList can grow dynamically while keeping the elements in contiguous memory locations.

123. Explain the method to convert ArrayList to Array and Array to ArrayList.

Conversion of List to ArrayList

There are multiple methods to convert List into ArrayList



Methods:

1. `Arrays.asList()`
2. `Collections.addAll()`
3. `add()` method

Programmers can convert an Array to ArrayList using `asList()` method of the Arrays class. It is a static method of the Arrays class that accepts the List object.

Syntax:

```
Arrays.asList(item)
```

Example:

Java

// Java program to demonstrate conversion of

// Array to ArrayList of fixed-size.

```
import java.util.*;
```

```
// Driver Class
```

```
class GFG {
```

```
    // Main Function
```

```

public static void main(String[] args)
{
    String[] temp = { "Abc", "Def", "Ghi", "Jkl" };
    // Conversion of array to ArrayList
    // using Arrays.asList
    List conv = Arrays.asList(temp);
    System.out.println(conv);
}
}

```

Output

[Abc, Def, Ghi, Jkl]

Conversion of ArrayList to Array



Methods:

1. Object[]toArray()
2. T[]toArray(T[]a)
3. get() method

Java programmers can convert ArrayList to

Syntax:

List_object.toArray(new String[List_object.size()])

Example:

Java

```
import java.util.List;
```

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

// Driver Class

```
class GFG {
```

```
    // Main Function
```

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

```
        // List declared
```

```
        List<Integer> arr = new ArrayList<>();
```

```
        arr.add(1);
```

```
        arr.add(2);
```

```
        arr.add(3);
```

```

arr.add(2);
arr.add(1);

// Conversion
Object[] objects = arr.toArray();

// Printing array of objects
for (Object obj : objects)
    System.out.print(obj + " ");
}
}

```

Output

1 2 3 2 1

124. How does the size of ArrayList grow dynamically? And also state how it is implemented internally.

Due to ArrayLists array-based nature, it grows dynamically in size ensuring that there is always enough room for elements. When an ArrayList element is first created, the default capacity is around 10-16 elements which basically depends on the Java version. ArrayList elements are copied over from the original array to the new array when the capacity of the original array is full. As the ArrayList size increases dynamically, the class creates a new array of bigger sizes and it copies all the elements from the old array to the new array. Now, the reference of the new array is used internally. This process of dynamically growing an array is known as resizing.

125. What is a Vector in Java?

Vectors in Java are similar and can store multiple elements inside them. Vectors follow certain rules mentioned below:

1. Vector can be imported using `Java.util.Vector`.
2. Vector is implemented using a dynamic array as the size of the vector increases and decreases depending upon the elements inserted in it.
3. Elements of the Vector using index numbers.
4. Vectors are synchronized in nature means they only used a single thread (only one process is performed at a particular time).
5. The vector contains many methods that are not part of the collections framework.

Syntax:

```
Vector gfg = new Vector(size, increment);
```

126. How to make Java ArrayList Read-Only?

An ArrayList can be made read only using the method provided by Collections using the `Collections.unmodifiableList()` method.

Syntax:

```
array_readonly = Collections.unmodifiableList(ArrayList);
```

Example:

Java

```
import java.util.*;
```

```

public class Main {
    public static void main(String[] argv) throws Exception {
        try {
            // creating object of ArrayList
            ArrayList<Character> temp = new ArrayList<>();
            // populate the list
            temp.add('X');
            temp.add('Y');
            temp.add('Z');
            // printing the list
            System.out.println("Initial list: " + temp);

            // getting readonly list
            // using unmodifiableList() method
            List<Character> new_array = Collections.unmodifiableList(temp);

            // printing the list
            System.out.println("ReadOnly ArrayList: " + new_array);

            // Attempting to add element to new Collection
            System.out.println("\nAttempting to add element to the ReadOnly ArrayList");
            new_array.add('A');
        } catch (UnsupportedOperationException e) {
            System.out.println("Exception is thrown : " + e);
        }
    }
}

```

Output

Initial list: [X, Y, Z]

ReadOnly ArrayList: [X, Y, Z]

Attempting to add element to the ReadOnly ArrayList

Exception is thrown : java.lang.UnsupportedOperationException

127. What is a priority queue in Java?

Priority Queue

Initial Queue = { }

Operation	Return value	Queue Content
insert (C)		C
insert (O)		C O
insert (D)		C O D
remove max	O	C D
insert (I)		C D I
insert (N)		C D I N
remove max	N	C D I
insert (G)		C D I G

A priority queue is an abstract data type similar to a regular queue or stack data structure. Elements stored in elements are depending upon the priority defined from low to high. The PriorityQueue is based on the priority heap.

Syntax:

Java

```
import java.util.*;
```

```
class PriorityQueueDemo {  
    // Main Method  
    public static void main(String args[]) {  
        // Creating empty priority queue  
        PriorityQueue<Integer> var1 = new PriorityQueue<Integer>();  
        // Adding items to the pQueue using add()  
        var1.add(10);  
        var1.add(20);  
        var1.add(15);  
        // Printing the top element of PriorityQueue  
        System.out.println(var1.peek());  
    }  
}
```

Output

10

128. Explain the LinkedList class.

LinkedList class is Java that uses a doubly linked list to store elements. It inherits the AbstractList class and implements List and Deque interfaces. Properties of the LinkedList Class are mentioned below:

1. LinkedList classes are non-synchronized.
2. Maintains insertion order.
3. It can be used as a list, stack, or queue.

Syntax:

```
LinkedList<class> list_name=new LinkedList<class>();
```

129. What is the Stack class in Java and what are the various methods provided by it?

A Stack class in Java is a LIFO data structure that implements the Last In First Out data structure. It is derived from a Vector class but has functions specific to stacks. The Stack class in java provides the following methods:

- **peek():** returns the top item from the stack without removing it
- **empty():** returns true if the stack is empty and false otherwise
- **push():** pushes an item onto the top of the stack
- **pop():** removes and returns the top item from the stack
- **search():** returns the 1, based position of the object from the top of the stack. If the object is not in the stack, it returns -1

130. What is Set in the Java Collections framework and list down its various implementations?

Sets are collections that don't store duplicate elements. They don't keep any order of the elements. The Java Collections framework provides several implementations of the Set interface, including:

- **HashSet:** HashSet in Java, stores the elements in a has table which provides faster lookups and faster insertion. HashSet is not ordered.
- **LinkedHashSet:** LinkedHashSet is an implementation of HashSet which maintains the insertion order of the elements.
- **TreeSet:** TreeSet stores the elements in a sorted order that is determined by the natural ordering of the elements or by a custom comparator provided at the time of creation.

131. What is the HashSet class in Java and how does it store elements?

The HashSet class implements the Set interface in the Java Collections Framework and is a member of the HashSet class. Unlike duplicate values, it stores a collection of distinct elements. In this implementation, each element is mapped to an index in an array using a hash function, and the index is used to quickly access the element. It produces an index for the element in the array where it is stored based on the input element. Assuming the hash function distributes the elements among the buckets appropriately, the HashSet class provides constant-time performance for basic operations (add, remove, contain, and size).

132. What is LinkedHashSet in Java Collections Framework?

The LinkedHashSet is an ordered version of Hashset maintained by a doubly-linked List across all the elements. It is very helpful when iteration order is needed. During iteration in LinkedHashSet, elements are returned in the same order they are inserted.

Syntax:

```
LinkedHashSet<E> hs = new LinkedHashSet<E>();
```


Example:

Java

```
import java.io.*;  
import java.util.*;
```

```
// Driver Class
```

```
class GFG {
```

```
    // Main Function
```

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

```
        // HashSet declared
```

```
        HashSet<Integer> hs = new HashSet<Integer>();
```

```
        // Add elements in HashSet
```

```
        hs.add(1);
```

```
        hs.add(2);
```

```
        hs.add(5);
```

```
        hs.add(3);
```

```
        // Print values
```

```
        System.out.println("Values:" + hs);
```

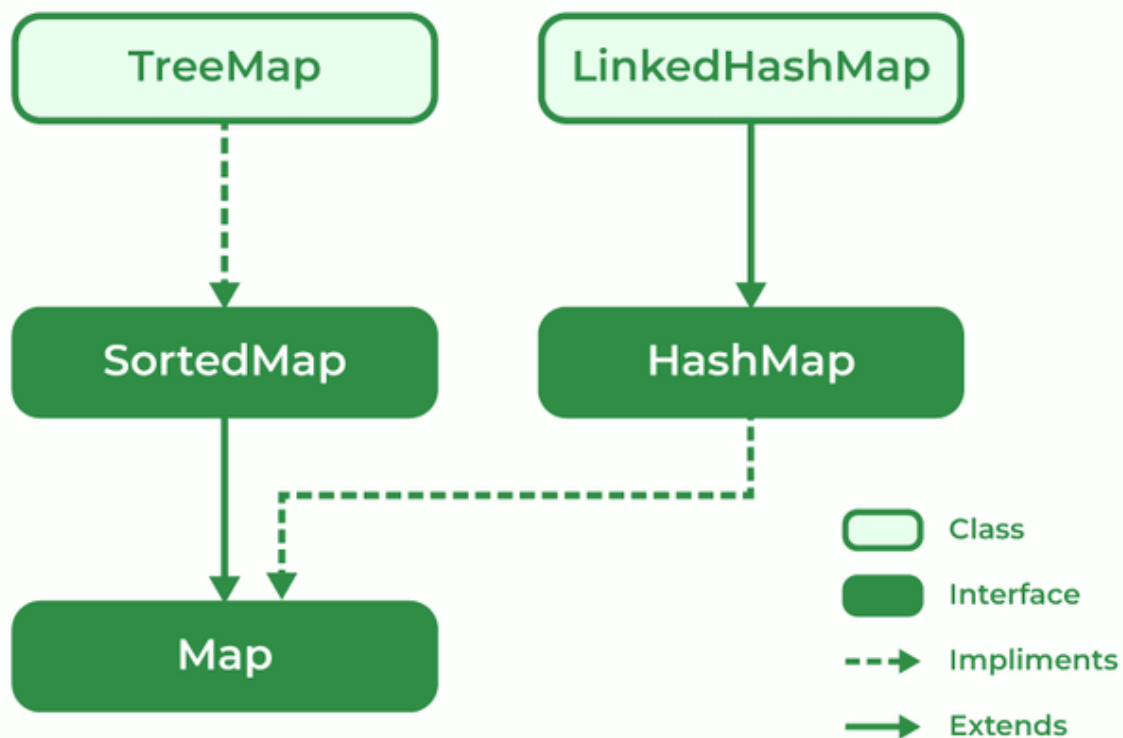
```
    }
```

```
}
```

Output

Values:[1, 2, 5, 3]

133. What is a Map interface in Java?



The map interface is present in the Java collection and can be used with Java.util package. A map interface is used for mapping values in the form of a key-value form. The map contains all unique keys. Also, it provides methods associated with it like containsKey(), contains value (), etc.

There are multiple types of maps in the map interface as mentioned below:

1. SortedMap
2. TreeMap
3. HashMap
4. LinkedHashMap

134. Explain Treemap in Java

TreeMap is a type of map that stores data in the form of key-value pair. It is implemented using the red-black tree. Features of TreeMap are :

1. It contains only unique elements.
2. It cannot have a NULL key
3. It can have multiple NULL values.
4. It is non-synchronized.
5. It maintains ascending order.

135. What is EnumSet?

EnumSet is a specialized implementation of the Set interface for use with enumeration type. A few features of EnumSet are:

1. It is non-synchronized.
2. Faster than HashSet.
3. All of the elements in an EnumSet must come from a single enumeration type.

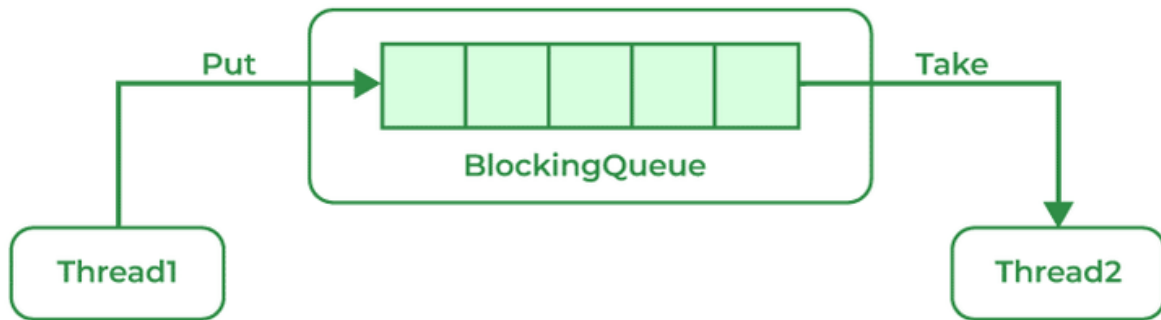
4. It doesn't allow null Objects and throws NullPointerException for exceptions.
5. It uses a fail-safe iterator.

Syntax:

```
public abstract class EnumSet<E extends Enum<E>>
```

Parameter: E specifies the elements.

136. What is BlockingQueue?



A blocking queue is a Queue that supports the operations that wait for the queue to become non-empty while retrieving and removing the element, and wait for space to become available in the queue while adding the element.

Syntax:

```
public interface BlockingQueue<E> extends Queue<E>
```

Parameters: E is the type of elements stored in the Collection

137. What is the ConcurrentHashMap in Java and do you implement it?

ConcurrentHashMap is implemented using Hashtable.

Syntax:

```
public class ConcurrentHashMap<K, V>
```

```
extends AbstractMap<K, V>
```

```
implements ConcurrentMap<K, V>, Serializable
```

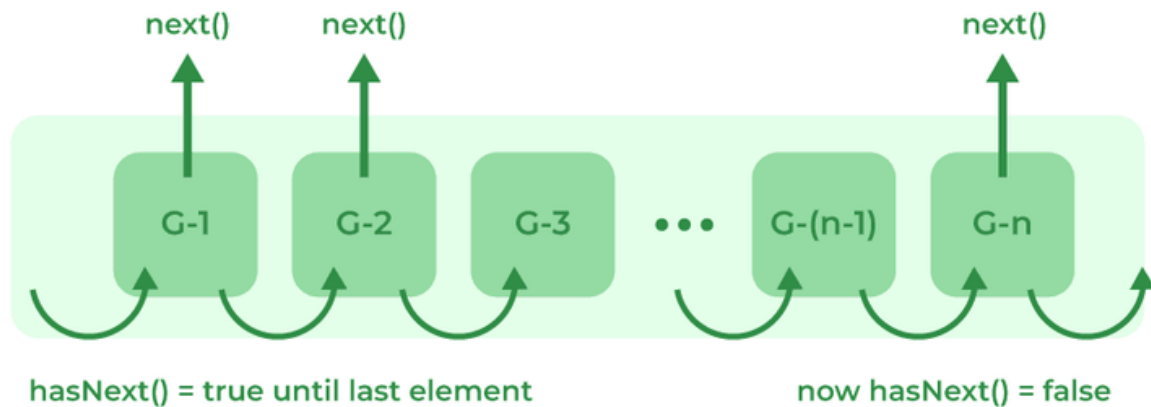
Parameters: K is the key Object type and V is the value Object type

138. Can you use any class as a Map key?

Yes, we can use any class as a Map Key if it follows certain predefined rules mentioned below:

1. The class overriding the equals() method must also override the hashCode() method
2. The concurrentHashMap class is thread-safe.
3. The default concurrency level of ConcurrentHashMap is 16.
4. Inserting null objects in ConcurrentHashMap is not possible as a key or as value.

139. What is an Iterator?



Java Iterator : Forward Direction

The Iterator interface provides methods to iterate over any Collection in Java. Iterator is the replacement of Enumeration in the Java Collections Framework. It can get an iterator instance from a Collection using the `_iterator()` method. It also allows the caller to remove elements from the underlying collection during the iteration.

140. What is an enumeration?

Enumeration is a user-defined data type. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. The main objective of the enum is to define user-defined data types.

Example:

```
// A simple enum example where enum is declared
// outside any class (Note enum keyword instead of
// class keyword)
enum Color
{
    RED, GREEN, BLUE;
}
```

141. What is the difference between Collection and Collections?

Collection	Collections
The Collection is an Interface.	Collections is a class.
It provides the standard functionality of data structure.	It is to sort and synchronize the collection elements.
It provides the methods that can be used for the data structure.	It provides static methods that can be used for various operations.

142. Differentiate between Array and ArrayList in Java.

Array	ArrayList
Single-dimensional or multidimensional	Single-dimensional
For and for each used for iteration	Here iterator is used to traverse riverArrayList
length keyword returns the size of the array.	size() method is used to compute the size of ArrayList.
The array has Fixed-size.	ArrayList size is dynamic and can be increased or decreased in size when required.
It is faster as above we see it of fixed size	It is relatively slower because of its dynamic nature
Primitive data types can be stored directly in unlikely objects.	Primitive data types are not directly added to unlikely arrays, they are added indirectly with help of autoboxing and unboxing
They can not be added here hence the type is in the unsafe.	They can be added here hence makingArrayList type-safe.
The assignment operator only serves the purpose	Here a special method is used known as add() method

143. What is the difference between Array and Collection in Java?

Array	Collections
Array in Java has a fixed size.	Collections in Java have dynamic sizes.
In an Array, Elements are stored in contiguous memory locations.	In Collections, Elements are not necessarily stored in contiguous memory locations.

Array	Collections
Objects and primitive data types can be stored in an array.	We can only store objects in collections.
Manual manipulation is required for resizing the array.	Resizing in collections is handled automatically.
The array has basic methods for manipulation.	Collections have advanced methods for manipulation and iteration.
The array is available since the beginning of Java.	Collections were introduced in Java 1.2.

144. Difference between ArrayList and LinkedList.

ArrayList	LinkedList
ArrayList is Implemented as an expandable Array.	LinkedList is Implemented as a doubly-linked list.
In ArrayList, Elements are stored in contiguous memory locations	LinkedList Elements are stored in non-contiguous memory locations as each element has a reference to the next and previous elements.
ArrayLists are faster for random access.	LinkedLists are faster for insertion and deletion operations
ArrayLists are more memory efficient.	LinkedList is less memory efficient
ArrayLists Use more memory due to maintaining the array size.	LinkedList Uses less memory as it only has references to elements
The search operation is faster in ArrayList.	The search operation is slower in LinkedList

145. Differentiate between ArrayList and Vector in Java.

ArrayList	Vector
ArrayList is implemented as a resizable array.	Vector is implemented as a synchronized, resizable array.
ArrayList is not synchronized.	The vector is synchronized.
ArrayLists are Faster for non-concurrent operations.	Vector is Slower for non-concurrent operations due to added overhead of synchronization.
ArrayLists were Introduced in Java 1.2.	Vector was Introduced in JDK 1.0.
Recommended for use in a single-threaded environment.	Vectors are Recommended for use in a multi-threaded environment.
The default initial capacity of ArrayLists is 10.	In Vectors, the default initial capacity is 10 but the default increment is twice the size.
ArrayList performance is high.	Vector performance is low.

146. What is the difference between Iterator and ListIterator?

Iterator	ListIterator
Can traverse elements present in Collection only in the forward direction.	Can traverse elements present in Collection both in forward and backward directions.
Used to traverse Map, List, and Set.	Can only traverse List and not the other two.
Indexes can't be obtained using Iterator	It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing the List.
Can't modify or replace elements present in Collection	Can modify or replace elements with the help of set(E e)

Iterator	ListIterator
Can't add elements, and also throws ConcurrentModificationException.	Can easily add elements to a collection at any time.
Certain methods of Iterator are next(), remove(), and hasNext().	Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e).

147. Differentiate between HashMap and Hashtable.

HashMap	Hashtable
HashMap is not synchronized	Hashtable is synchronized
One key can be a NULL value	NULL values not allowed
The iterator is used to traverse HashMap.	Both Iterator and Enumertar can be used
HashMap is faster.	Hashtable is slower as compared to HashMap.

148. What is the difference between Iterator and Enumeration?

Iterator	Enumeration
The Iterator can traverse both legacies as well as non-legacy elements.	Enumeration can traverse only legacy elements.
The Iterator is fail-fast.	Enumeration is not fail-fast.
The Iterators are slower.	Enumeration is faster.
The Iterator can perform a remove operation while traversing the collection.	The Enumeration can perform only traverse operations on the collection.

149. What is the difference between Comparable and Comparator?

Comparable	Comparator
The interface is present in java.lang package.	The Interface is present in java.util package.
Provides compareTo() method to sort elements.	Provides compare() method to sort elements.
It provides single sorting sequences.	It provides multiple sorting sequences.
The logic of sorting must be in the same class whose object you are going to sort.	The logic of sorting should be in a separate class to write different sorting based on different attributes of objects.
Method sorts the data according to fixed sorting order.	Method sorts the data according to the customized sorting order.
It affects the original class.	It doesn't affect the original class.
Implemented frequently in the API by Calendar, Wrapper classes, Date, and String.	It is implemented to sort instances of third-party classes.

150. What is the difference between Set and Map?

Set	Map
The Set interface is implemented using java.util package.	The map is implemented using java.util package.
It can extend the collection interface.	It does not extend the collection interface.
It does not allow duplicate values.	It allows duplicate values.
The set can sort only one null value.	The map can sort multiple null values.