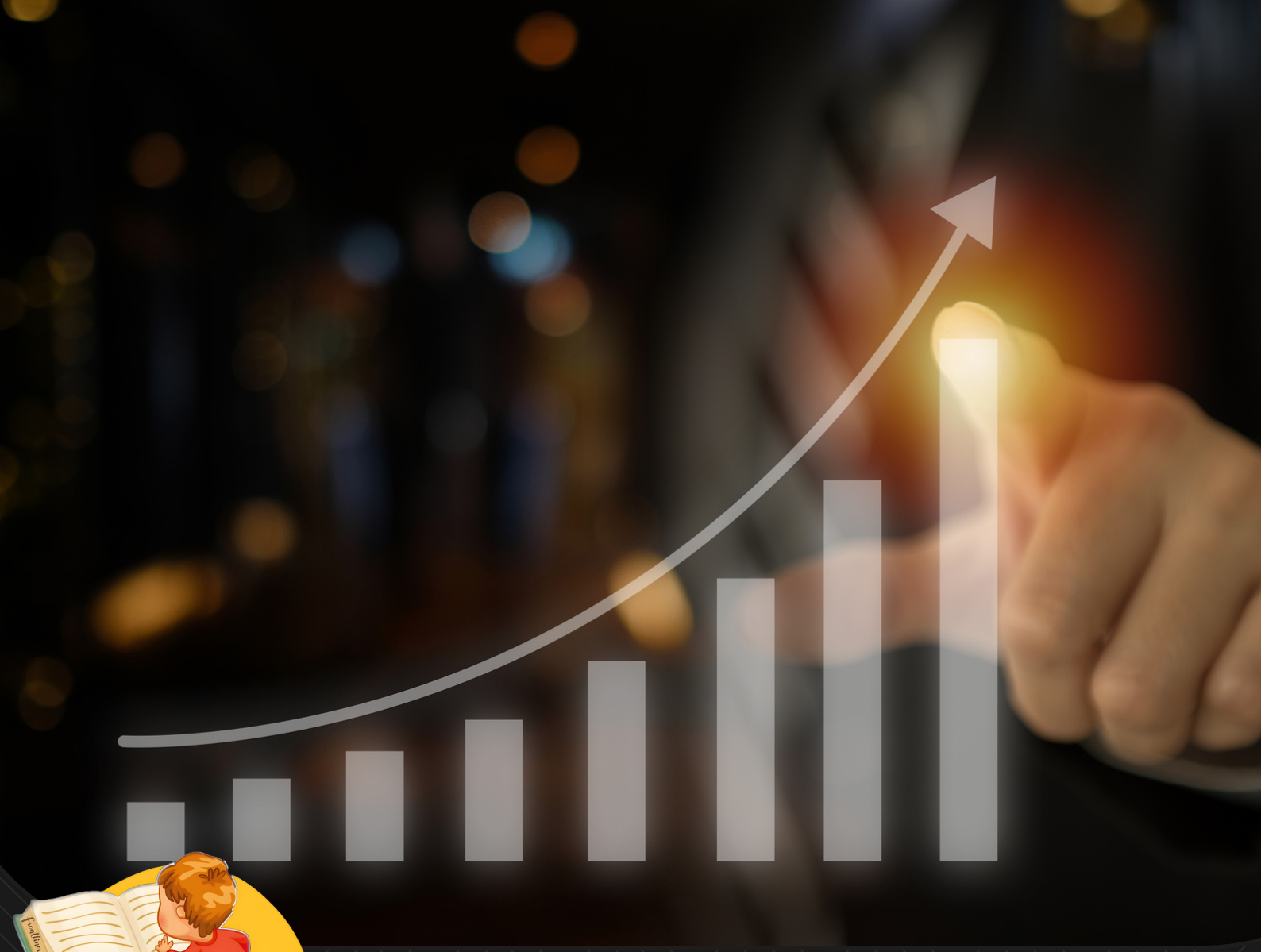

FLM'S JSL COMPLETE COURSE GUIDE

JAVA ESSENTIALS: A COMPLETE GUIDE FOR ASPIRING SOFTWAEE ENGINEERS

EXPLORE JAVA HIGH-LEVEL INFORMATION, CORE SYLLABUS, INTERVIEW ESSENTIALS,MCQ'S,CODING QUESTIONS,HANDY CHEAT SHEET

START LEARNING TODAY

FLM JSL Complete Guide:

When we are learning any programming language like why we choose this language,Whatever you had chosen, self understanding is very important.This is the Complete Guide for those who are ready to learn Java.

Here,Comes the first Question

Why Only Java? Why not Python,JavaScript etc ….

We always pick any course or something we want to learn based on some suggestions from friends,seniors or colleagues…Suggestions are good but we can't blindly follow and take anything. It's better to have a good self understanding about anything we are going through.

The question is why only java and why not other languages

- Is it Because java is faster compared to other languages?
- Is it because of the 27 years legacy java has?
- Is it because of the high popularity of name & fame ?

To get better understanding, we are going through a detailed guide about why to learn

Java is being developed, Considering the drawbacks of c,and c++

There are drawbacks in c and c++ like

Memory management in C: It is difficult because there we have to deal with the concepts of Pointers.Pointers involve an extra level of indirection, as they store the memory address of a variable rather than its value.you need to manage memory allocation and deallocation manually. Forgetting to free allocated memory can lead to memory leaks.

Platform Independence: In C/C++, Code needs to be changed and recompiled for different computers.

Error Handling: In C/C++ Handling errors can make your code complicated.

Object-Oriented Programming (OOP): C++ supports OOP, but C does not, and C++ can get complex with more advanced OOP features.

These are some of the major drawbacks in C,C++,How Java overcame all these drawbacks and became an evergreen programming language we will discuss.

All the drawbacks discussed above are some of the most important Features of Java.Let's talk about Java Features.

Core Features of Java

- ★ Java is Platform Independent
- ★ Java is Object-Oriented
- ★ Java is Strongly Typed language
- ★ Java has Automatic Memory Management
- ★ Java is Multithreaded
- ★ Java has wide range of Libraries and Frameworks
- ★ Java is Backward Compatible.

Java is Platform Independent

Suppose,In C you written a simple helloworld program(helloworld.c) and then you compiled the program,it generates an **.exe** file,means this executable file directly converts your source code into machine understandable code(means 0's and 1's) you thought yes,then what is the problem here? Let me tell you,This .exe file is platform dependent. One means if you'd written this program in a windows system,and you want to run this in another OS like mac or unix it won't support.

In Java,when you compile a java program,the java compiler doesn't convert this source code into machine code rather it generates a **bytecode(.class file)** which is platform independent and you can run this in any operating system.

Java is Object Oriented

What are the OOPS Concepts in Java?

Encapsulation: It's like packing a lunch box. You put everything you need inside, and you have a specific way to open it. This keeps your food safe and organised. In Java, encapsulation keeps data secure and organised, making it easier to understand and maintain the code.

Inheritance: Think of it as a family tree. You inherit traits from your parents, just like classes in Java can inherit properties from other classes. This allows for code reuse, making it faster to develop new features without reinventing the wheel.

Polymorphism: Imagine a TV remote. You press the same power button to turn on different TVs. Similarly, in Java, you can use the same method in different ways depending on the object it's applied to. This flexibility makes Java versatile and adaptable to changing needs.

Abstraction: It's like using a smartphone. You don't need to know how it works internally to use its features. Abstraction in Java allows you to focus on what the code does, not how it does it, making it easier to work with complex systems.

Java is Strongly Typed Language

In java,when you are defining any value we have to compulsory mention its datatype,In python the concept of data types is not there.But java is strongly typed language we have to mention which type of data we are dealing with,otherwise it will lead to runtime exceptions.

Java has Automatic memory management

In java we write any code in classes and for that we will create objects,this objects will be stored in memory.unused objects must be handled definitely for effective memory management,For this we have garbage collector in java,this garbage collector automatically handles unused/unreferenced objects and deletes them from the memory.

Java is MultiThreaded

Multithreading in Java allows a program to do multiple tasks at the same time, like listening to music while typing a document. It helps make programs more efficient by using the computer's resources wisely. Java provides built-in features to create and manage threads easily, making it simple to add multitasking to your programs.

Java is Backward Compatible

Java supports Backward compatibility,meaning Java has been there from 1996,the code which was written in earlier Java versions,is supported till to the latest versions.

So,now we have an idea why we are learning java,what are its core features and how it overcame the drawbacks of C/C++.Next,Before we step into learning java it’s better to know the syllabus and what we are learning in each module.

Core Java Syllabus Breakdown

Java Fundamentals

1.Introduction to Java
   A. First Program in Java
   B. Variables
   C. Data Types
   D. Operators
2.Control Flow Statements
   A. If-else
   B. Switch-Case
   C. For loop & For Each Loop
   D. While loop do-while loop
   E. Continue statement
   F. Break statement
3.Arrays
4.Strings
5.Methods

Understanding the basics is the key to a solid foundation.

Data Types and Variables

Variable in Java is a data container that stores the data values during Java program execution. Every variable is assigned a data type, which designates the type and quantity of values it can hold. Variable is a memory location name of the data. The Java variables have mainly three types: Local, Instance and Static.

Data Types are divided into two group -

- Primitive - byte,short,int,long,float,double,boolean and char
- Non-Primitive - String, Arrays and Classes

Conditionals

Java has the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed if the same condition is false
- Use **else if** to specify a new condition to test; if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed
- Use **?,:** operator to specify one line condition
- **While** loop used to execute a block of code repeatedly as long as a specified condition is true.

- **do-while** loop: Similar to the while loop, but the code block is executed at least once before the condition is tested.
- **continue** statement: Used to skip the current iteration of a loop and proceed to the next iteration.
- **break** statement: Used to exit a loop or switch statement before it has completed all its iterations.
- **for loop**: Used to execute a block of code a specific number of times.

Arrays

An array is an indexed collection of a fixed number of homogeneous data elements. The main advantage of arrays is that we can represent multiple values with the same name So that readability of the code will be improved.

OOPS Concepts

Object-oriented programming is a core of Java Programming, which is used for designing a program using classes and objects. This can also be characterised as data controlling for accessing the code.

- Class
- Interfaces
- Object
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Exception Handling

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.
There are three types of exceptions -

1. Checked Exception - exceptions checked at compile time. Example - IOException
2. Unchecked Exception - exceptions checked at run time. Example - NullPointerException
3. Error - It is irrecoverable. Example - OutOfMemoryError

MultiThreading

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

Collection Framework

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

Garbage Collection

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program.

These are the most important concepts in core java.If you are going to learn advanced concepts like Java Frameworks you should be clear and confident in these core concepts.

Next,We will look into the most important interview questions which are asked in many java Interviews.

100 Most Important Interview Questions with Answers

Q1. What is the difference between JDK and JRE?

- JDK stands for Java Development Kit and is a full-featured software development kit for Java, including the JRE (Java Runtime Environment), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed for Java development.
- JRE stands for Java Runtime Environment and provides the libraries, Java Virtual Machine (JVM), and other components to run applications and applets written in Java.

Q2. Why do people say that Java is a 'write once and run anywhere' language?

- This phrase is used because Java is designed to be portable, meaning that Java code can run on any device that has a compatible JVM, without the need for recompilation. This is achieved through the use of bytecode, which is compiled from Java code and can be interpreted on any platform by the JVM.

Q3. Do you think 'main' used for the main method is a keyword in Java?

- No, 'main' is not a keyword in Java. It's an identifier chosen by convention as the name of the method that the JVM looks for as the starting point of the program.

Q4. Can we write the main method as public void static instead of public static void?

- No, the main method must be declared as 'public static void'. This is the signature the JVM expects as the entry point of the application. The 'void' keyword indicates that the main method does not return any value.

Q5. In Java, if we do not specify any value for local variables, then what will be the default value of the local variables?

- Local variables in Java do not have a default value. If you try to use a local variable without initializing it, the compiler will give you an error, forcing you to initialise it before you can use it.

Q6. What are the main principles of Object-Oriented Programming?

The main principles of Object-Oriented Programming (OOP) are Encapsulation, Abstraction, Inheritance, and Polymorphism.

- Encapsulation is the mechanism of restricting access to some of the object's components, which means that the internal representation of an object can't be seen from outside of the objects' definition. Access to this data is typically through a public interface.
- Abstraction means dealing with the level of complexity by hiding unnecessary details from the user. This can be achieved using abstract classes and interfaces.
- Inheritance is the process where one class acquires the properties and behaviours of another class.
- Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. The specific implementation that gets executed is determined at runtime.

Q7. In Java, what is the default value of an object reference defined as an instance variable in an Object?

- The default value of an object reference variable is null if it is defined as an instance variable in a class and not explicitly initialized.

Q8. Why do we need a constructor in Java?

- A constructor in Java is needed to create new objects and initialize them with default or specific values. It’s called when an instance of a class is created and is used to set up the initial state of the object.

Q9. Why do we need a default constructor in Java classes?

- A default constructor is provided by Java if no other constructors are provided. It’s needed so that an instance of the class can be created with default settings. If a class doesn't have any explicit constructor, the Java compiler will automatically provide a no-argument constructor, known as the default constructor.

Q10. What is the value returned by Constructor in Java?

- Constructors do not return values and do not have a return type, not even void. They are only used for initializing new objects.

Q11. What is the purpose of the 'this' keyword in Java?

- The this keyword in Java is a reference to the current object, the one that is being used to call the method or constructor. It is used to differentiate between class attributes and parameters with the same name, or to pass the current instance to another method or constructor.

Q12. Which class in Java is the superclass of every other class?

- The Object class is the root of the class hierarchy in Java. Every class has Object as a superclass, all objects, including arrays, implement the methods of this class.

Q13. Why does Java not support multiple inheritance?

- Java does not support multiple inheritance to avoid the complexity and ambiguity caused by it, such as the "diamond problem," where a class inherits the same method from multiple superclasses. However, Java allows multiple inheritance of types, which can be achieved by implementing multiple interfaces.

Q14. What is the purpose of the 'super' keyword in Java?

- The super keyword in Java is used to refer to the immediate parent class of the current class. It can be used to call the superclass's methods and constructors.

Q15. In Java, why do we use a static variable?

- A static variable is shared among all instances of a class; it belongs to the class rather than a particular object. This means that only one instance of a static field exists regardless of how many times the class has been instantiated. Static variables are often used to store common properties of all objects (for instance, a company name that's the same for all employees).

Q16. What happens when the static modifier is not mentioned in the signature of the main method?

- If the static modifier is omitted from the main method signature, the JVM will not be able to invoke the main method without creating an instance of the class, which goes against the Java specification for program entry points. The program will compile, but the JVM will throw an error when it attempts to run the application, saying that it cannot find the main method with the correct signature.

Q17. What is the other name of Method Overloading?

- Method Overloading is also referred to as compile-time polymorphism or static polymorphism. This is because the method to execute is determined at compile-time based on the method signature.

Q18. How will you implement method overloading in Java?

- Method overloading in Java is implemented by defining multiple methods in the same class, with the same name but different parameters. They may differ in the number of parameters, the type of parameters, or both.

Q19. What kinds of argument variations are allowed in Method Overloading?

In method overloading, the argument variations allowed include:

- Different number of parameters in each overloaded method.
- Different types of parameters in each method.
- If the number of parameters is the same, at least one parameter must have a different type.

Q20. Is it allowed to overload the main() method in Java?

- Yes, the main method can be overloaded with different parameter lists. However, the JVM will always call the original main method with a single string array as an argument. Other overloaded main methods can be called like any other method with overloading.

Q21. How do we implement method overriding in Java?

- Method overriding in Java is done by creating a method in the subclass with the same name, return type, and parameters as a method in its superclass. The @Override annotation, while not required, can be used to explicitly declare an overriding method.

Q22. Are we allowed to override a static method in Java?

- Static methods cannot be overridden in the way that instance methods can be because they are not dispatched on the instance of a class, but rather the class itself. However, a static method can be hidden by another static method in a subclass, which is sometimes referred to as method hiding rather than overriding.

Q23. Is it allowed to override an overloaded method?

- Yes, an overloaded method in the superclass can be overridden in the subclass as long as it follows the overriding rules, meaning it must have the same name, return type, and parameter list.

Q24. What is an abstract class in Java?

- An abstract class in Java is a class that cannot be instantiated on its own and is declared with the abstract keyword. It can contain abstract methods—methods without an implementation—as well as implemented methods. It's used as a base class for other classes that are expected to implement the abstract methods.

Q25. Is it allowed to mark a method abstract without marking the class abstract?

- No, if a class contains an abstract method, then the class itself must also be declared as abstract. You cannot have an abstract method in a non-abstract class.

Q26. Is it allowed to mark a method abstract as well as final?

- No, a method cannot be both abstract and final. An abstract method is one that is declared without an implementation (without braces, and followed by a semicolon), meaning it must be overridden in a subclass. A final method cannot be overridden by subclasses, so the two modifiers are mutually exclusive.

Q27. Can we instantiate an abstract class in Java?

- No, you cannot instantiate an abstract class directly. However, you can instantiate an anonymous class that extends the abstract class, or you can instantiate a subclass that provides implementations for the abstract class's abstract methods.

Q28. What is an interface in Java?

- An interface in Java is an abstract type that is used to specify a behaviour that classes must implement. Interfaces are declared using the interface keyword, and they can contain constant declarations and method signatures, which are by default abstract and public.

Q29. Is it allowed to mark an interface method as static?

- Yes, starting with Java 8, you can define static methods in interfaces which can be called independently of any object. These are not abstract methods and must provide a body.

Q30. Why can an Interface not be marked as final in Java?

- An interface cannot be marked as final because the very purpose of an interface is to be implemented by classes. Marking an interface as final would contradict its intended use as it would prevent any class from implementing it.

Q31. What is a marker interface?

- A marker interface is an interface with no methods or fields declaration; its sole purpose is to convey metadata about a class. Examples from the Java standard library include Serializable and Cloneable.

Q32. What is the difference between an abstract class and an interface in Java?

- An abstract class can have instance methods that implement a default behavior. Interfaces can only declare methods (until Java 8, after which they can also have default and static methods). A class can inherit from just one abstract class, but it can implement multiple interfaces.

Q33. Does Java allow us to use private and protected modifiers for variables in interfaces?

- Before Java 9, variables in interfaces are implicitly public, static, and final. Since Java 9, you can have private methods, which means you can encapsulate common code between default methods, but you cannot have private or protected variables.

Q34. How can you change the value of a final variable in Java?

- Once a final variable has been assigned a value, it cannot be changed. If the final variable is a reference, this means the reference cannot change, but the object it points to can be modified if it is mutable.

Q35. Can a class be marked final in Java?

- Yes, a class can be marked final. A final class cannot be subclassed, which is useful when creating an immutable class or securing a class against modification.

Q36. How can we prohibit inheritance in Java?

- Inheritance can be prohibited by marking a class as final, which means no class can extend it.

Q37. Is it allowed to declare the main method as final?

- Yes, you can declare the main method as final. This will prevent any subclass from hiding the main method, though it's generally not a concern since the main method is static and often resides in a final class or a class that is not intended to be subclassed.

Q38. Is it mandatory to import java.lang package every time?

- No, it is not mandatory to import java.lang package as it is imported automatically by the Java compiler for every program.

Q39. Can you import the same package or class twice in your class?

- Yes, you can technically import the same package or class more than once, but it has no effect. The Java compiler is smart enough to ignore duplicate imports.

Q40. What is serialization?

- Serialization is the process of converting an object into a byte stream to save an object's state to a file or send it over the network, which can later be restored through deserialization.

Q41. What is Deserialization?

- Deserialization is the reverse process of serialization, where a byte stream is converted back into an object. This is typically used to retrieve the object's state from a file or a network.

Q42. What is transient?

- The transient keyword in Java is used to indicate that a field should not be serialized. When an object is serialized, transient fields are skipped and not included in the serialized representation.

Q43. What is Garbage Collection in Java?

- Garbage Collection (GC) in Java is the process of automatically freeing memory by destroying objects that are no longer reachable or used in the program. This helps in managing memory efficiently and reduces the chances of memory leaks.

Q44. Why do we use the finalize() method in Java?

- The finalize() method is called by the garbage collector before an object is destroyed. It can be used to perform any cleanup operations such as releasing resources (like closing files or database connections) before the object is garbage collected.

Q45. What is the meaning of Immutable in the context of the String class in Java?

- In Java, an immutable object is one that cannot be modified after it is created. The String class is an example of an immutable class. Once a String object is created, its value cannot be changed. Any method that seems to modify a String actually creates a new String object.

Q46. How many ways are there in Java to create a String object?

- There are two ways to create a String object in Java:
    - By using a string literal: String str = "Hello";
    - By using the new keyword: String str = new String("Hello");

Q47. What is the basic difference between a String and StringBuffer object?

- The main difference is that String objects are immutable, while StringBuffer objects are mutable. This means that StringBuffer objects can be modified after they are created, which is useful for scenarios where you need to make frequent modifications to a string of characters.

Q48. How will you create an immutable class in Java?

To create an immutable class in Java:

- Declare the class as final so it can't be extended.
- Make all fields private and final.
- Do not provide setter methods.
- Ensure that methods that modify the state of the object return a new object instead of modifying the current one.
- If the class has mutable object fields, ensure that these are not directly exposed or modified.

Q49. In Java, what are the differences between Checked and Unchecked Exceptions?

- Checked exceptions are checked at compile-time and must be either caught in a try-catch block or declared to be thrown by the method using the throws keyword. Examples include IOException and SQLException.
- Unchecked exceptions are not checked at compile-time and do not need to be declared or caught. They are a subclass of RuntimeException, and common examples include NullPointerException and ArrayIndexOutOfBoundsException.

Q50. What is the base class for Error and Exception classes in Java?

- The base class for both Error and Exception classes in Java is the Throwable class.

Q51. What is a finally block in Java?

- A finally block in Java is a block that is used to execute important code such as closing resources (e.g., files, database connections) regardless of whether an exception is thrown or caught. It is always executed after the try and catch blocks, even if an unexpected exception occurs.

Q52. Can we create a finally block without creating a catch block?

- Yes, a finally block can be used without a catch block. In such cases, the finally block will be executed after the try block completes. However, if an exception is thrown in the try block and not caught, it will still propagate after the finally block is executed.

Q53. Do we have to always put a catch block after a try block?

- No, it's not mandatory to have a catch block after a try block. You can have either a catch block, a finally block, or both after a try block. However, if you don't handle an

exception with a catch block, you should at least provide a finally block to clean up resources.

Q54. In what scenarios, a finally block will not be executed?

- A finally block will not be executed if the JVM exits unexpectedly (e.g., System.exit() is called) or if a fatal error occurs that causes the process to abort.

Q55. What is the difference between throw and throws in Java?

- throw is a keyword used to explicitly throw an exception from a method or any block of code.
- throws is a keyword used in the method signature to declare that the method might throw one or more exceptions.

Q56. What is the difference between Collection and Collections Framework in Java?

- The Collection is an interface in Java that represents a group of objects known as its elements.
- The Collections Framework is a set of classes and interfaces that implement commonly reusable collection data structures and algorithms, which includes interfaces like List, Set, and Map, and classes like ArrayList, LinkedList, HashSet, and HashMap.

Q57. What is the root interface of the Collection hierarchy in Java?

- The root interface of the Collection hierarchy in Java is the Collection interface itself. It is at the top of the collection hierarchy and is extended by other collection interfaces like List, Set, and Queue.

Q58. How will you convert a List to a Set?

- You can convert a List to a Set in Java by passing the list as an argument to the constructor of a Set implementation, like HashSet. For example:

```
List<Integer> list = Arrays.asList(1, 2, 3, 4);
Set<Integer> set = new HashSet<>(list);
```

Q59. What are the differences between the two data structures: a Vector and an ArrayList?

Both Vector and ArrayList are implementations of the List interface. The main differences are:

- Vector is synchronized, meaning it is thread-safe, while ArrayList is not synchronized and is not thread-safe.
- Vector has a legacy design and is considered slower than ArrayList because of its synchronized methods.

- ArrayList is generally preferred over Vector for non-threaded applications due to its better performance.

Q60. In which scenario, LinkedList is better than ArrayList in Java?

- LinkedList is better than ArrayList when you need faster insertion and deletion operations, as these operations are generally O(1) in LinkedList and O(n) in ArrayList. LinkedList is also better when you need to frequently add or remove elements from the beginning or middle of the list.

Q61. What are the differences between a List and Set collection in Java?

List:
- Allows duplicate elements.
- Maintains the order of insertion.
- Examples: ArrayList, LinkedList.

Set:
- Does not allow duplicate elements.
- Does not guarantee the order of elements (except in some implementations like LinkedHashSet).
- Examples: HashSet, TreeSet, LinkedHashSet.

Q62. What are the differences between a HashSet and TreeSet collection in Java?

HashSet:
- Uses a hash table for storage.
- Does not maintain any order of elements.
- Offers constant time performance for basic operations (add, remove, contains) under ideal conditions.

TreeSet:
- Uses a red-black tree for storage.
- Maintains a sorted order of elements.
- Offers logarithmic time performance for basic operations.

Q63. In Java, how will you decide when to use a List, Set, or a Map collection?

- List: Use when you need an ordered collection that allows duplicates.
- Set: Use when you need a collection that does not allow duplicates and order is not important (or use LinkedHashSet if insertion order matters).
- Map: Use when you need to store key-value pairs and want fast retrieval based on keys.

Q64. What are the differences between a HashMap and a Hashtable in Java?

HashMap:
- Not synchronised (not thread-safe).
- Allows one null key and multiple null values.

- Iterators are fail-fast.

Hashtable:

- Synchronized (thread-safe).
- Does not allow null keys or null values.
- Enumerations are not fail-fast.

Q65. How does hashCode() method work in Java?

- The hashCode() method in Java returns an integer hash code for an object, which is used in hashing-based collections like HashMap, HashSet, and Hashtable to bucket objects. The method is designed to distribute objects evenly across the buckets for efficient retrieval.

Q66. Is it a good idea to use Generics in collections?

- Yes, using generics in collections is a good idea because it provides compile-time type safety, reducing the risk of ClassCastException. It also makes the code more readable and eliminates the need for type casting when retrieving elements.

Q67. How will you copy elements from a Source List to another list?

You can use the addAll method to copy elements from one list to another. For example:

```
List<Integer> sourceList = Arrays.asList(1, 2, 3);
List<Integer> destinationList = new ArrayList<>();
destinationList.addAll(sourceList);
```

Q68. What is the difference between an Iterator and ListIterator in Java?

Iterator:

- Can be used with any collection.
- Supports forward iteration only.
- Provides methods hasNext(), next(), and remove().

ListIterator:

- Can be used with lists only.
- Supports both forward and backward iteration.
- Provides additional methods like hasPrevious(), previous(), nextIndex(), previousIndex(), add(), and set().

Q69. What is the reason for overriding equals() method?

- The equals() method is overridden to define the equality of instances of a class based on their content rather than their memory addresses. This is especially important for classes used in collections that rely on equality checks (e.g., keys in a HashMap).

Q70. How will you reverse a List in Java?

You can use the Collections.reverse() method to reverse a list in Java. For example:

```
List<Integer> list = Arrays.asList(1, 2, 3);
Collections.reverse(list);
```

Q71. What are the main differences between HashMap and ConcurrentHashMap in Java?

HashMap:

- Not thread-safe.
- Allows one null key and multiple null values.
- Iterators are fail-fast.

ConcurrentHashMap:

- Thread-safe without locking the entire map.
- Does not allow null keys or null values.
- Provides better concurrency by partitioning the map.

Q72. Why does the Map interface not extend the Collection interface in Java?

- The Map interface does not extend the Collection interface because it represents a mapping of keys to values, which is a different data structure compared to collections that store individual elements. Maps have unique keys, and each key maps to a single value, which is a different concept compared to collections like lists and sets.

Q73. What is the contract of hashCode() and equals() methods in Java?

- The contract between hashCode() and equals() methods is:
  - If two objects are equal according to the equals() method, then calling the hashCode() method on each of the two objects must produce the same integer result.
  - If two objects are not equal according to the equals() method, there is no requirement on the hashCode() method, but it is desirable for the method to produce distinct integer results for distinct objects to improve the performance of hash tables.

Q74. How can you make a Collection class read-only in Java?

You can make a collection class read-only by using the unmodifiable wrappers provided by the Collections class. For example:

```
List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3));
List<Integer> readOnlyList = Collections.unmodifiableList(list);
```

Q75. When is UnsupportedOperationException thrown in Java?

- The UnsupportedOperationException is thrown when an unsupported operation is attempted on a collection that does not support that operation. For example, attempting to add an element to an unmodifiable collection will throw this exception.

Q76. Let's say there is a Customer class. We add objects of the Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of the Customer class?

You can sort the Customer objects in the ArrayList by using the Collections.sort() method with a custom comparator that compares the firstName attribute of the Customer class. For example:

```
Collections.sort(customerList, new Comparator<Customer>() {
 @Override
 public int compare(Customer c1, Customer c2) {
 return c1.getFirstName().compareTo(c2.getFirstName());
 }
});
```

Q77. Can you explain how HashMap works in Java?

- A HashMap in Java works by using a hash table. When you put a key-value pair into the map, the key's hash code is used to determine where to store the entry in the underlying array. If two keys have the same hash code or different hash codes that result in the same index after hashing, a collision occurs, and the entries are stored in a linked list or tree at that index. When retrieving a value, the same hash code is used to find the correct index and then the list or tree is searched for the entry with the matching key.

Q78. What is the default priority of a thread in Java?

- The default priority of a thread in Java is 5, which is the constant Thread.NORM_PRIORITY.

Q79. What are the three different priorities that can be set on a Thread in Java?

The three different priorities that can be set on a thread in Java are:

- Thread.MIN_PRIORITY (value 1)
- Thread.NORM_PRIORITY (value 5)
- Thread.MAX_PRIORITY (value 10)

Q80. Is it possible to call run() method instead of start() on a thread in Java?

- Yes, it is possible to call the run() method directly instead of start() on a thread in Java. However, doing so will not create a new thread of execution; instead, the run() method will be executed in the current thread, just like any other method call.

Q81. Can we start a thread two times in Java?

- No, once a thread has been started, it cannot be started again. If you attempt to start a thread that has already been started, a java.lang.IllegalThreadStateException will be thrown.

Q82. In Java, is it possible to lock an object for exclusive use by a thread?

- Yes, in Java, it is possible to lock an object for exclusive use by a thread using the synchronized keyword. When a thread enters a synchronized method or block, it acquires the lock on the specified object, and no other thread can enter a synchronized method or block on the same object until the lock is released.

Q83. What is the life cycle of a Thread?

The life cycle of a thread in Java includes the following states:
New: The thread is created but not yet started.
Runnable: The thread is ready to run and is waiting for CPU time.
Running: The thread is executing its run method.
Blocked: The thread is blocked waiting for a monitor lock.
Waiting: The thread is waiting indefinitely for another thread to perform a particular action.
Timed Waiting: The thread is waiting for a specified amount of time.
Terminated: The thread has completed its execution and is terminated.

Q84. What is synchronization in Java?

- Synchronization in Java is a mechanism to control access to shared resources by multiple threads. It is used to ensure that only one thread can access a particular resource at a time, preventing data inconsistency and thread interference. This is typically achieved using the synchronized keyword.

Q85. Can you explain how HashMap works in Java?

- A HashMap in Java works by using a hash table. When you put a key-value pair into the map, the key's hash code is used to determine where to store the entry in the underlying array. If two keys have the same hash code or different hash codes that result in the same index after hashing, a collision occurs, and the entries are stored in a linked list or tree at that index. When retrieving a value, the same hash code is used to find the correct index and then the list or tree is searched for the entry with the matching key.

Q86. What is the difference between final, finally, and finalize in Java?

final: A keyword that can be applied to variables, methods, and classes. A final variable cannot be reassigned, a final method cannot be overridden, and a final class cannot be subclassed.
finally: A block that is used to execute important code such as closing resources, regardless of whether an exception is thrown or caught. It is always executed after the try and catch blocks.
finalize: A method that is called by the garbage collector before an object is destroyed. It is used to perform any cleanup operations such as releasing resources.

Q87. How do you ensure thread safety in a Java application?

- Use synchronization to control access to shared resources.
- Use thread-safe classes provided by the Java Collections Framework, such as ConcurrentHashMap.
- Use atomic variables provided by the java.util.concurrent.atomic package.
- Use lock objects provided by the java.util.concurrent.locks package.

Q88. What is the purpose of the volatile keyword in Java?

- The volatile keyword is used to indicate that a variable's value will be modified by different threads. It ensures that the value of the variable is always read from and written to main memory, providing visibility of changes to all threads.

Q89. Can you explain the difference between shallow copy and deep copy in Java?

- Shallow copy: Creates a new object that is a copy of the original object, but the fields are not copied; instead, both objects share the same field references.
- Deep copy: Creates a new object that is a copy of the original object, and all fields of the original object are also copied, so the new object is completely independent of the original.

Q90. What is the significance of the toString() method in Java?

- The toString() method returns a string representation of an object. It is often overridden in classes to provide a meaningful description of the object, which is useful for debugging and logging.

Q91. How do you handle exceptions in a Java application?

- Use try-catch blocks to catch and handle exceptions.
- Use the throws keyword in the method signature to declare that a method might throw an exception.
- Use the throw keyword to explicitly throw an exception.

Q92. What are the differences between a Checked Exception and an Unchecked Exception in Java?

- Checked Exception: Must be either caught in a try-catch block or declared to be thrown by the method using the throws keyword. Examples include IOException and SQLException.
- Unchecked Exception: Do not need to be declared or caught. They are a subclass of RuntimeException, and common examples include NullPointerException and ArrayIndexOutOfBoundsException.

Q93. Can you explain the concept of autoboxing and unboxing in Java?

- Autoboxing: The automatic conversion of primitive types (e.g., int) to their corresponding wrapper class objects (e.g., Integer) by the Java compiler.
- Unboxing: The automatic conversion of wrapper class objects to their corresponding primitive types.

Q94. What is the use of the super keyword in Java?

- The super keyword is used to refer to the immediate parent class of the current class. It can be used to call the superclass's methods and constructors.

Q95. How do you implement a Singleton pattern in Java?

A Singleton pattern ensures that a class has only one instance and provides a global point of access to it. It can be implemented by:

- Making the constructor private.
- Creating a private static instance of the class.
- Providing a public static method that returns the instance.

Q96. What is the difference between an abstract class and an interface in Java?

- An abstract class can have instance methods that implement a default behavior, whereas interfaces can only declare methods (until Java 8, after which they can also have default and static methods). A class can inherit from only one abstract class but can implement multiple interfaces.

Q97. How can you create an immutable object in Java?

To create an immutable object in Java:

- Declare the class as final so it can't be extended.
- Make all fields private and final.
- Do not provide setter methods.
- Ensure that methods that modify the state of the object return a new object instead of modifying the current one.
- If the class has mutable object fields, ensure that these are not directly exposed or modified.

Q98. What are the benefits of using Generics in Java?

- Generics provide compile-time type safety, reducing the risk of ClassCastException. They also make the code more readable and eliminate the need for type casting when retrieving elements from collections.

Q99. Can you explain the concept of the Java Memory Model?

- The Java Memory Model (JMM) defines how threads interact with memory in a Java application. It ensures visibility of changes to variables between threads and establishes rules for reordering of instructions to ensure consistent and predictable behaviour in concurrent programs.

Q100. What is the use of the transient keyword in Java?

- The transient keyword is used to indicate that a field should not be serialized when an object is converted to a byte stream. This is useful for fields that contain sensitive information or are derived from other fields and do not need to be persisted.

These Are the most important and asked interview questions in java,After learning concepts come here and check whether you are good at answering this questions!!!

Next,we are moving to important coding questions asked in interviews.

25 commonly asked Java programming questions in interviews

Q1. How do you reverse a string in Java?

```
public class Main {
    public static void main(String[] args) {
        String str = "Hello";
        char[] chars = str.toCharArray();
        for (int i = chars.length - 1; i >= 0; i--) {
            System.out.print(chars[i] + " ");
        }
    }
}
```

Explanation:

Ee program lo,first manam str ane string ni define chesukunna and dhanni reverse cheyali,so simple ga str anedi string kada dhanni character array loki marchukunnam with the use of toCharArray() function,Ee function oka string character array la masthundhi.Tharuvatha aa character array length find out chesam with the help of length function for finding length of arrays.Next for loop lo i anedi array last index nundi start ayyi > 0 varaku okko character print chestham.

Q2. How do you determine if a string is a palindrome?

```
public class Main {
    public static void main(String[] args) {
        String str = "madam";
        String reversed = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed =reversed + str.charAt(i);
        }
        boolean isPalindrome = str.equals(reversed);
        System.out.println("Is palindrome: " + isPalindrome);
    }
}
```

Explanation:

Ee program lo first oka string str ane variable tho define chesukunnam,next oka empty string reversed anedi define chesam,Palindrome ante oka string ni reverse chesthe,aa reversed string original string tho match aythe palindrome antam,Suppose if you reverse the string madam,the reversed string also madam,For loop anedi string index last nundi traverse ayyi,each character anedi reversed empty string tho attach avthundhi.charAt() ane function oka string lo character retrieve cheyadaniki use avthundhi with the help of index.so,for loop iterations cmplt ayyaka original and reversed strings untay manaki,then equals ane method two strings content ni compare chesthundhi,avi rendu equal aythe true return avthundhi ledhante false return avthundhi.True ayyindhi ante Palindrome ani True print chestham.

Q3. Find the number of occurrences of a character in a String?

```
public class Main {
 public static void main(String[] args) {
 String str = "Hello";
 char ch = 'l';
 int count = 0;
 for (int i = 0; i < str.length(); i++) {
 if (str.charAt(i) == ch) {
 count++;
 }
 }
 System.out.println("Number of occurrences: " + count);
 }
}
```

Explanation:

So,ee program oka string lo oka character enni sarlu repeat ayyindhi count return cheyali,Here we taken a string hello and we need to find how many times l letter repeated.For that,oka for loop thesukoni starting index to ending index traverse chesthu lopala oka if condition pettukunnam,means manam retrieve chese each character manam count find out cheyalsina character ki equal aythe count ane variable ni increment chestham,ante yenni sarlu aa character repeat anni sarlu if condition true ayyi count anedi increment avthundhi and aa count ni loop bayata print chestham.

Q4. How to find out if the given two strings are anagrams or not?

```
public class Main {
 public static void main(String[] args) {
 String str1 = "listen";
 String str2 = "silent";
 char[] chars1 = str1.toCharArray();
 char[] chars2 = str2.toCharArray();
 Arrays.sort(chars1);
 Arrays.sort(chars2);
 boolean isAnagram = Arrays.equals(chars1, chars2);
 System.out.println("Are anagrams: " + isAnagram);
 }
}
```

Explanation:

Ee program lo two strings anevi anagrams aa anedi find out cheyali,Anagram ante oka string lo unde characters(alphabets)second string lo unnaya anedi compare

chesukuntam,anni characters second strings lo kuda unte aa two strings anevi anagrams antamu.For that,ekkada two strings str1 and str2 unnay,to strings ni toCharArray() function tho character array la convert chesukuntam,Convert ayyaka Arrays.sort() ane function tho sort chestham,Ee sort function alphabetical order lo sort chesthundhi.Then Arrays.equals method tho both arrays ni compare chestham rendu equal aythe true return avthundhi so anagram avthundhi.

Q5. How do you calculate the number of vowels and consonants in a String?

```
public class Main {
 public static void main(String[] args) {
 String str = "Hello World";
 int vowels = 0, consonants = 0;
 for (int i = 0; i < str.length(); i++) {
 char ch = Character.toLowerCase(str.charAt(i));
 if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 
'u') {
 vowels++;
 } else if (ch >= 'a' && ch <= 'z') {
 consonants++;
 }
 }
 System.out.println("Vowels: " + vowels + ", Consonants: " + 
consonants);
 }
}
```

Explanation:

Ee program lo oka string lo unna vowels and consonants count return cheyali,For that str ane oka string ni define chesam vowels,constants ane variables ni 0 ki intialize chesam,for loop anedi starting to ending traverse avthundhi each characher retrive ayyi if condition pettukunamm if the retrived character (ch) matches with any vowel letters(a,e,i,o,u)then we will increment the vowel count by 1 and if it's not matched then goes to else part and the retrieved character matches with other alphabets other than a,e,i,o,u and the constants count will be incremented.

Q6. How do you get the matching elements in an integer array?

```
public class Main {
 public static void main(String[] args) {
 int[] arr1 = {1, 2, 3, 4, 5};
 int[] arr2 = {3, 4, 5, 6, 7};
 Set<Integer> set = new HashSet<>();
 for (int num : arr1) {
 set.add(num);
 }
 System.out.print("Matching elements: ");
 for (int num : arr2) {
 if (set.contains(num)) {
 System.out.print(num + " ");
 }
 }
 }
}
```

Explanation:

In this program, we're finding the matching elements between two integer arrays, arr1 and arr2. We start by adding all the elements of arr1 into a HashSet called set. This allows for efficient lookup of elements in arr1. Then, we iterate through arr2 and check if any of its elements are contained in the set. If a match is found, it means that the element is present in both arrays, and we print it out. The use of a HashSet for arr1 ensures that the lookup operation is O(1) on average, making the overall algorithm more efficient.

Q7. How do you reverse an array?

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.print("Reversed array: ");
        for (int i = arr.length - 1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Explanation:

Ee program oka array ni reverse cheyadam,simple ga oka for loop thesukoni last index nundi traverse chesthu each element ni print chestham.

Q8. How would you swap two numbers without using a third variable?

```
public class Main {
 public static void main(String[] args) {
 int a = 5, b = 10;
 System.out.println("Before swap: a = " + a + ", b = " + b);
 a = a + b;
 b = a - b;
 a = a-b;
 System.out.println("After swap: a = " + a + ", b = " + b);
 }
}
```

```
Explanation:
      Initial values:
            a = 5
            b = 10
      First step (a = a + b):
            a is assigned the sum of a and b, so a becomes 15 (a = 5
            + 10).
      Second step (b = a - b):
            b is assigned the difference between the new a and b, so
            b becomes 5 (b = 15 - 10).
      Third step (a = a - b):
            a is assigned the difference between the new a and the
            new b, so a becomes 10 (a = 15 - 5).
After these steps, the values of a and b are successfully swapped,
with a being 10 and b being 5
```

Q9. Print a Fibonacci series using recursion?

```
public class Main {
 public static int fibonacci(int n) {
 if (n <= 1) return n;
 return fibonacci(n - 1) + fibonacci(n - 2);
 }

 public static void main(String[] args) {
 int n = 10; // Number of terms in the Fibonacci series
 for (int i = 0; i < n; i++) {
 System.out.print(fibonacci(i) + " ");
 }
 }
}
```

Explanation:

The Fibonacci series is generated using recursion. The fibonacci method recursively calls itself to calculate the Fibonacci number for a given position n. If n is 0 or 1, the method returns n directly (base cases). For n greater than 1, the method returns the sum of the Fibonacci numbers at positions n - 1 and n - 2. The main method prints the first 10 terms of the Fibonacci series by calling the fibonacci method for each position from 0 to 9.

Q10. How do you find the factorial of an integer?

```
public class Main {
    public static void main(String[] args) {
        int n = 5;
        int factorial = 1;
        for (int i = 1; i <= n; i++) {
            factorial=factorial*i;
        }
        System.out.println("Factorial of " + n + " is " +
factorial);
    }
}
```

Explanation:

Ee program ,oka number yokka factorial value find out chestham.Intially,n=5 ane number tho assign chesukunnam,and factorial ane variable 1 tho assign chesukunnam,Next,For loop 1 to n varaku iterate avthundhi,ante 1 to 5 ekkada,each time factorial variable i tho multiply avthundhi and factorial value update avthundhi.

```
     Iteration 1: i = 1, factorial = 1 * 1 = 1
     Iteration 2: i = 2, factorial = 1 * 2 = 2
     Iteration 3: i = 3, factorial = 2 * 3 = 6
     Iteration 4: i = 4, factorial = 6 * 4 = 24
     Iteration 5: i = 5, factorial = 24 * 5 = 120
```

After these iterations, the final value of factorial is 120, which is the factorial of 5.

Q11. How would you find the second largest number in an array?

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        // Manual sorting (Bubble Sort)
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
        // The second largest number is the second last element in
the sorted array
        int secondLargest = arr[arr.length - 2];
        System.out.println("Second largest number: " +
secondLargest);
    }
}
```

Explanation:

Sorting the array (Bubble Sort):

- In the first iteration of the outer loop (i = 0), the inner loop compares adjacent elements and swaps them if they are in the wrong order. After this iteration, the largest element 5 bubbles up to the last position.
- In the second iteration (i = 1), the inner loop stops one element before the last (since the last element is already sorted), and the second largest element 4 bubbles up to the second last position.
- This process continues until the entire array is sorted in ascending order. After sorting, the array becomes {1, 2, 3, 4, 5}.

Finding the second largest number:

- After sorting, the largest number is at the last position (arr.length - 1), which is 5 in this case.
- The second largest number is the one before the largest, at the second last position (arr.length - 2). In this example, it's 4.

Output:

- The program prints "Second largest number: 4", which is the second largest number in the original array.

Q12. How do you remove all occurrences of a given character from the input string?

```
public class Main {
 public static void main(String[] args) {
 String str = "Hello World";
 char ch = 'o';
 String result = "";
 for (int i = 0; i < str.length(); i++) {
 if (str.charAt(i) != ch) {
 result += str.charAt(i);
 }
 }
 System.out.println("Result: " + result);
 }
}
```

Explanation:

In this program, we remove all occurrences of a given character (ch) from the input string (str). We iterate through each character of the string using a for loop. If the current character is not equal to ch, we add it to the result string. This way, only characters that are not equal to ch are included in the result. Finally, we print the result string, which contains the original string with all occurrences of ch removed.

Q13. Showcase Inheritance with the help of a program?

```
class Animal {
 void eat() {
 System.out.println("Eating...");
 }
}

class Dog extends Animal {
 void bark() {
 System.out.println("Barking...");
 }
}

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

```
}
```

Q14. Explain overloading and overriding with the help of a program?

```
class Overload {
 void demo(int a) {
 System.out.println("a: " + a);
 }

 void demo(int a, int b) {
 System.out.println("a and b: " + a + "," + b);
 }
}

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

class DogOverride extends Override {
 @Override
 void eat() {
 System.out.println("Dog is eating");
 }
}

public class Main {
 public static void main(String[] args) {
 Overload obj = new Overload();
 obj.demo(10);
 obj.demo(10, 20);

 Override a = new Override();
 DogOverride d = new DogOverride();
 a.eat();
 d.eat();
 }
}
```

Q15. How do you check if the given number is prime?

```
public class Main {
    public static void main(String[] args) {
        int n = 29;
        boolean isPrime = true;
        if (n <= 1) {
```

```
            isPrime = false;
        } else {
            for (int i = 2; i <= Math.sqrt(n); i++) {
                if (n % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }
        System.out.println(n + " is prime: " + isPrime);
    }
}
```

Explanation:

In this program, we check if the number n (29 in this example) is a prime number. A prime number is a number that is greater than 1 and has no positive divisors other than 1 and itself.
Initial Check: First, we check if n is less than or equal to 1. If it is, n cannot be prime, so we set isPrime to false.
Prime Check: If n is greater than 1, we use a for loop to check for divisors. We only need to check up to the square root of n because if n has a divisor greater than its square root, it must also have a divisor smaller than its square root. If we find a number i that divides n evenly (n % i == 0), n is not prime, so we set isPrime to false and break out of the loop.
Output: Finally, we print whether n is prime based on the value of isPrime.
In this example, 29 is a prime number, so the output will be "29 is prime: true".

Q16. How do you sum all the elements in an array?

```
public class Main {
 public static void main(String[] args) {
 int[] arr = {1, 2, 3, 4, 5};
 int sum = 0;
 for (int num : arr) {
 sum += num;
 }
 System.out.println("Sum of array elements: " + sum);
 }
}
```

Explanation:

Initial Check: Modhataga, manam check chesthunnam n 1 ki equal or chinnadi aithe, appudu n prime number kaadu ani, so isPrime ni false ga set chestham.

Prime Check: If n 1 kanna peddadi aithe, we use a for loop to check for divisors. We only need to check n yoka square root varaku,because if n ki oka divisor n yoka square root kanna peddadi unte, it must also have a divisor n yoka square root kanna chinnadi untundi. Ee loop lo, if manam okka number i ni kanugonnam, which divides n evenly (n % i == 0), appudu n prime number kaadu, so isPrime ni false ga set chestham and loop nundi break chestham.

Output: Finally, manam print chestham whether n is a prime number based on isPrime yoka value.

Ee example lo, 29 is a prime number, so output will be "29 is prime: true".

Q17. Write a Java program to check if a vowel is present in a String.

```
public class Main {
 public static void main(String[] args) {
 String str = "Hello World";
 boolean hasVowel = false;
 for (int i = 0; i < str.length(); i++) {
 char ch = Character.toLowerCase(str.charAt(i));
 if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch ==
'u') {
 hasVowel = true;
 break;
 }
 }
 System.out.println("Contains vowel: " + hasVowel);
 }
}
```

Explanation:

Ee program lo oka string vowel anedi present ayyundha ledha anedi check chesthunam,First initial ga oka string ni str tho define chesukuntam,next hasVowel anedi intial ga false ki set chesukuntam,next each character ni retrive chesthu based on index value,aa retrive aina character a, e, i, o, u lo ediyna okati aithe,appudu manam set chesthunnam hasVowel ni true ki,and loop nundi bayataki vachestham,Chivari lo, manam print chesthunnam whether the string contains a vowel or not anedi based on hasVowel variable.
Ee program output ga "Contains vowel: true" ani vasthundi, endukante "Hello World" ane string lo vowels unnayi.

Q18. How do you implement a binary search in Java?

```
public class Main {
 public static int binarySearch(int[] arr, int key) {
 int low = 0, high = arr.length - 1;
 while (low <= high) {
 int mid = (low + high) / 2;
 if (arr[mid] == key) {
 return mid;
 } else if (arr[mid] < key) {
 low = mid + 1;
 } else {
 high = mid - 1;
 }
 }
 return -1;
 }

 public static void main(String[] args) {
 int[] arr = {1, 2, 3, 4, 5};
 int key = 3;
 int result = binarySearch(arr, key);
 if (result != -1) {
 System.out.println("Element found at index " + result);
 } else {
 System.out.println("Element not found");
 }
 }
}
```

Explanation:
In this program, we are implementing a binary search algorithm to find a given key in a sorted array arr.

Initialization: We start with two pointers, low set to 0 (the first index of the array) and high set to arr.length - 1 (the last index of the array).

Binary Search Loop:

- The loop continues as long as low is less than or equal to high.
- We calculate the middle index mid as the average of low and high.
- If the element at mid is equal to the key, we have found the key, and we return the index mid.
- If the element at mid is less than the key, it means the key must be in the right half of the array, so we set low to mid + 1.

- If the element at mid is greater than the key, it means the key must be in the left half of the array, so we set high to mid - 1.

Result:

- If the loop ends and we haven't returned an index, it means the key is not in the array, so we return -1.
- In the main method, we call binarySearch with an array and a key. If the result is not -1, we print the index where the element is found; otherwise, we print "Element not found".

In this example, the key 3 is found at index 2 in the array {1, 2, 3, 4, 5}.

Q19. Write a Java program that sorts HashMap by value.

```
import java.util.*;

public class Main {
 public static void main(String[] args) {
 HashMap<String, Integer> map = new HashMap<>();
 map.put("One", 1);
 map.put("Two", 2);
 map.put("Three", 3);

 List<Map.Entry<String, Integer>> list = new
ArrayList<>(map.entrySet());
 list.sort(Map.Entry.comparingByValue());

 System.out.println("Sorted HashMap by value: ");
 for (Map.Entry<String, Integer> entry : list) {
 System.out.println(entry.getKey() + ": " + entry.getValue());
 }
 }
}
```

Explanation:

In this Java program, we begin by initializing a HashMap called map and populate it with key-value pairs, where the keys are strings ("One", "Two", "Three") and the values are integers (1, 2, 3). To sort the HashMap by its values, we convert its entry set into a List of Map.Entry objects using the entrySet() method and initialize a new ArrayList called list with this set. We then use the sort method on the list object, passing Map.Entry.comparingByValue() as the comparator to sort the entries

based on their values. This comparator compares the values of Map.Entry objects, allowing us to sort the list in ascending order of values. Finally, we iterate over the sorted list and print each entry's key and value, effectively displaying the HashMap sorted by its values. This program showcases how to sort a HashMap by its values using a List and a custom comparator in Java.

Q20. Can you prove that a String object in Java is immutable programmatically?

```
public class Main {
 public static void main(String[] args) {
 String str1 = "Hello";
 String str2 = str1;
 str1 = str1 + " World";

 System.out.println("str1: " + str1);
 System.out.println("str2: " + str2);
 System.out.println("Is String immutable: " + !str1.equals(str2));
 }
}
```

Explanation:

In this program, we demonstrate that a String object in Java is immutable. Initially, we have a String `str1` with the value "Hello". Then, we create a new String `str2` and assign `str1` to it. Next, we concatenate " World" to `str1`, creating a new String object with the value "Hello World". Despite this modification to `str1`, `str2` still holds the original value "Hello". When we compare `str1` and `str2` using the `equals` method, we find that they are not equal, indicating that the original `str2` was not modified. This behavior demonstrates the immutability of String objects in Java, as once a String object is created, its value cannot be changed.

Q21. How do you illustrate a try-catch example in Java?

```
public class Main {
 public static void main(String[] args) {
 try {
 int result = 10 / 0;
 System.out.println("Result: " + result);
 } catch (ArithmeticException e) {
 System.out.println("ArithmeticException caught: " +
e.getMessage());
 }
 }
}
```

Q22. How do you create a functional interface?

```
@FunctionalInterface
interface MyFunctionalInterface {
 void execute();
}

public class Main {
 public static void main(String[] args) {
 MyFunctionalInterface myFunc = () ->
System.out.println("Executing functional interface method");
 myFunc.execute();
 }
}
```

Q23. Write a program to remove duplicate elements from an array in Java.

```
import java.util.*;

public class Main {
 public static void main(String[] args) {
 int[] arr = {1, 2, 2, 3, 4, 4, 5};
 Set<Integer> set = new LinkedHashSet<>();
 for (int num : arr) {
 set.add(num);
 }
 int[] uniqueArr = new int[set.size()];
 int i = 0;
 for (int num : set) {
 uniqueArr[i++] = num;
 }
```

```
 System.out.println("Array without duplicates: " +
Arrays.toString(uniqueArr));
 }
}
```

Explanation:
we remove duplicate elements from an array. We start by initializing an array arr with duplicate elements. Then, we create a LinkedHashSet called set, which preserves the order of elements and does not allow duplicates. We iterate through each element of the array using a for-each loop and add them to the set. Since set does not allow duplicates, only unique elements will be added. Next, we create a new array uniqueArr with the size of the set. We then iterate through the set and copy each element to the uniqueArr. Finally, we print the uniqueArr, which contains the array without any duplicate elements.

Q24. How do you swap the first and last elements of an array in Java?

```
public class Main {
 public static void main(String[] args) {
 int[] arr = {1, 2, 3, 4, 5};
 int temp = arr[0];
 arr[0] = arr[arr.length - 1];
 arr[arr.length - 1] = temp;

 System.out.println("Array after swapping first and last elements:
");
 for (int num : arr) {
 System.out.print(num + " ");
 }
 }
}
```

Explanation:

In this Java program, we swap the first and last elements of an array. We start by initializing an array arr with some elements. Then, we use a temporary variable temp to store the value of the first element (arr[0]). Next, we assign the value of the last element (arr[arr.length - 1]) to the first element. Finally, we assign the value of temp (which holds the original value of the first element) to the last element.

After swapping the elements, we print the modified array to show the result.

Q25. How do you check if a string contains only digits in Java?

```
public class Main {
 public static void main(String[] args) {
 String str = "12345";
 boolean isNumeric = true;
 for (int i = 0; i < str.length(); i++) {
 if (!Character.isDigit(str.charAt(i))) {
 isNumeric = false;
 break;
 }
 }
 System.out.println("String contains only digits: " + isNumeric);
 }
}
```

Explanation:

Ee program lo oka string lo numeric digits unnaya ledha ani find out chestham,str ane String define chesukunnam first,boolean variable isNumeric and set it to true initially, assuming that the string contains only digits.

Next, we iterate through each character of the string using a for loop. For each character, we use the Character.isDigit() method to check if it is a digit. If we encounter a character that is not a digit, we set isNumeric to false and break out of the loop, as the string does not contain only digits.

Finally, we print the value of isNumeric, which indicates whether the string contains only digits (true) or not (false).

These are some of the important and commonly asked programming questions in many entry level interviews.You can write these programs in your own logic also.

Next,we will see some Quiz questions to check our understanding of Core concepts.

50 Core java Quiz Questions

Q1. Which part of the JVM is responsible for loading class files?

A) Method Area
B) Heap Space
C) Class Loader
D) Call Stack

Answer: C) Class Loader
Explanation: The Class Loader is a subsystem of JVM that is used to load class files.

Q2. In the JVM, where are static variables stored?

A) Stack Memory
B) Heap Memory
C) Method Area
D) Native Method Stack

Answer: C) Method Area
Explanation: Static variables are stored in the Method Area, as they are class level data.

Q3. What does JIT compiler stand for and what is its role in the JVM?

A) Just-In-Time compiler; it compiles bytecode to native machine code at runtime.
B) Java Internal Translator; it translates Java code into bytecode.
C) Just-In-Thread compiler; it manages thread execution.
D) Java Implementation Translator; it implements the Java API classes.

Answer: A) Just-In-Time compiler; it compiles bytecode to native machine code at runtime.
Explanation: The JIT compiler is part of the JVM that improves the performance of Java applications by compiling bytecode into native machine code at run time.

Q4. Which of these is not a valid Java identifier?

A) _myVariable
B) $myVariable
C) 2myVariable
D) myVariable2

Answer: C) 2myVariable

Explanation: Identifiers cannot start with a digit.

Q5. What is the default value of a local variable in Java?

A) null
B) 0
C) Depends on the type
D) Not assigned a default value

Answer: D) Not assigned a default value
Explanation: Local variables are not assigned default values and must be initialized before use.

Q6. Which operator is used to compare two values for equality in Java?

A) =
B) ==
C) ===
D) !=

Answer: B) ==
Explanation: The == operator is used for comparison to check if two values are equal.

Q7. What will be the result of the expression 11 % 3 in Java?

A) 2
B) 3
C) 1
D) 11

Answer: A) 2
Explanation: The % operator returns the remainder of division. Thus, 11 % 3 is 2.

Q8. What will the following code snippet print?

```
int a = 10;
if(a < 20) {
 System.out.println("Less than 20");
} else {
 System.out.println("20 or more");
}
```

A) Less than 20
B) 20 or more
C) 10
D) No output
Answer: A) Less than 20

Explanation: Since a is less than 20, the if block will execute.

Q9. Which loop construct will execute the body of the loop at least once even if the condition is false initially?

A) for
B) while
C) do-while
D) foreach

Answer: C) do-while
Explanation: The do-while loop executes its body at least once before checking the condition.

Q10. What does the break statement do in a switch case construct?

A) Pauses the execution
B) Stops the condition check
C) Exits the switch case block
D) None of the above

Answer: C) Exits the switch case block
Explanation: The break statement is used to exit from the switch case block once a case is executed.

Q11. What will be the output of the following code?

```
for(int i = 0; i < 5; i++) {
 if(i == 3) {
 break;
 }
 System.out.print(i + " ");
}
```

A) 0 1 2 3
B) 0 1 2
C) 0 1 2 3 4
D) 1 2 3

Answer: B) 0 1 2
Explanation: The loop will terminate when i becomes 3 due to the break statement.

Q12. What will the following code snippet print?

```
int i = 0;
while(i < 3) {
 System.out.println("Hello");
 i++;
}
```

A) Hello (printed once)
B) Hello (printed twice)
C) Hello (printed three times)
D) No output

Answer: C) Hello (printed three times)
Explanation: The loop will print "Hello" three times before i becomes 3.

Q13. Which of the following is not a valid for loop declaration in Java?

A) for(int i = 0; i < 5; i++)
B) for(; ;)
C) for(int i = 0, j = 5; i + j < 10; i++, j--)
D) for(int i = 0; i < 5)

Answer: D) for(int i = 0; i < 5)
Explanation: The for loop declaration is missing the iteration statement.

Q14. What is the output of the following code?

```
int x = 10;
int y = 20;
if(x > y) {
 System.out.println("X is greater");
} else if(x < y) {
 System.out.println("Y is greater");
} else {
 System.out.println("X and Y are equal");
}
```

A) X is greater
B) Y is greater
C) X and Y are equal
D) No output

Answer: B) Y is greater
Explanation: The condition x < y is true, so "Y is greater" will be printed.

Q15. Which of the following loops will not compile in Java?

A) do { System.out.println("Hello"); } while(false);
B) while(false) { System.out.println("Hello"); }
C) for(;false;) { System.out.println("Hello"); }
D) All of the above will compile

Answer: D) All of the above will compile
Explanation: All the loops are syntactically correct, although some may not execute their bodies.

Q16. What is the scope of a local variable in Java?

A) Throughout the class
B) Within the block in which it is defined
C) Throughout the package
D) Throughout the method

Answer: B) Within the block in which it is defined
Explanation: A local variable is accessible only within the block where it is declared.

Q17. Which of the following is true about instance variables in Java?

A) They are shared among all objects of the class.
B) They are stored in the stack memory.
C) They can be accessed directly within static methods.
D) They are unique to each object of the class.

Answer: D) They are unique to each object of the class.
Explanation: Instance variables are associated with a specific object and are not shared among objects.

Q18. What will be the output of the following code?

```
int a = 10;
int b = ++a + a++;
System.out.println(b);
```

A) 21
B) 22
C) 23
D) 24
Answer: B) 22
Explanation: ++a increments a to 11 before addition, and a++ increments a to 12 after addition.

Q19. Which keyword is used to declare a constant in Java?

A) const
B) final
C) static
D) immutable

Answer: B) final
Explanation: The final keyword is used to declare constants in Java.

Q20. What is the output of the following code?

```
int x = 5;
int y = x++;
int z = ++x;
System.out.println(y + " " + z);
```

A) 5 7
B) 6 7
C) 5 6
D) 6 6

Answer: A) 5 7
Explanation: y gets the value of x before incrementing, and z gets the value of x after incrementing twice.

Q21. What is the purpose of a constructor in Java?

A) To create objects
B) To initialize objects
C) To return values
D) To define methods

Answer: B) To initialize objects
Explanation: Constructors are used to initialize the state of an object when it is created.

Q22. Which of the following is a valid constructor declaration in Java?

A) public void MyClass() {}
B) public MyClass {}
C) public MyClass() {}
D) MyClass() void {}

Answer: C) public MyClass() {}
Explanation: A constructor has the same name as the class and does not have a return type.

Q23. In Java, which keyword is used to inherit from a superclass?

A) extends
B) implements
C) super
D) this

Answer: A) extends
Explanation: The extends keyword is used for inheritance in Java.

Q24. What will the following code snippet print?

```
class Animal {
 String sound() { return "Animal sound"; }
}

class Dog extends Animal {
 String sound() { return "Bark"; }
}

public class Test {
 public static void main(String[] args) {
 Animal myDog = new Dog();
 System.out.println(myDog.sound());
 }
}
```

A) Animal sound
B) Bark
C) Compilation Error
D) Runtime Error

Answer: B) Bark
Explanation: This is an example of method overriding. The sound method of the Dog class overrides the sound method of the Animal class.

Q25. Which of the following is true about the super keyword in Java?

A) It is used to call the constructor of the current class.
B) It refers to the immediate parent class object.
C) It can be used to access static methods of the parent class.
D) It is used to define a method that cannot be overridden.

Answer: B) It refers to the immediate parent class object.
Explanation: The super keyword is used to refer to the parent class object and is often used to access methods and variables of the superclass.

Q26. What is polymorphism in Java?

A) The ability of a variable to hold different data types
B) The ability of a method to perform different tasks based on the input
C) The ability of a class to have multiple constructors
D) The ability of an object to take on many forms

Answer: D) The ability of an object to take on many forms
Explanation: Polymorphism is a concept that allows an object to take on multiple forms, usually through method overriding or method overloading.

Q27. Which of the following demonstrates method overloading?

A) A method that changes its behavior based on the input parameters
B) A method with the same name but different parameters in the same class
C) A subclass providing a specific implementation of a method that is already provided by its superclass
D) A class having multiple methods with the same name and same parameters

Answer: B) A method with the same name but different parameters in the same class
Explanation: Method overloading is a feature that allows a class to have more than one method with the same name but different parameters.

Q28. What is abstraction in Java?

A) Hiding the implementation details and showing only the functionality
B) Breaking down a complex system into simpler parts
C) The process of object creation
D) The ability of an object to take on multiple forms

Answer: A) Hiding the implementation details and showing only the functionality
Explanation: Abstraction is a concept of hiding the internal implementation and showing only the necessary features of an object.

Q29. Which keyword is used to achieve abstraction in Java?

A) abstract
B) final
C) static
D) private

Answer: A) abstract
Explanation: The abstract keyword is used to create abstract classes and methods, which are used to achieve abstraction in Java.

Q30. What is the purpose of a try-catch block in Java?

A) To compile the code
B) To handle exceptions
C) To define a loop
D) To declare variables

Answer: B) To handle exceptions
Explanation: A try-catch block is used to catch and handle exceptions that may occur in a block of code.

Q31. Which of the following is a checked exception in Java?

A) NullPointerException
B) ArrayIndexOutOfBoundsException
C) IOException
D) ArithmeticException

Answer: C) IOException
Explanation: Checked exceptions are exceptions that are checked at compile-time, and IOException is an example of a checked exception.

Q32. What is the output of the following code?

```
String s1 = "Java";
String s2 = new String("Java");
System.out.println(s1 == s2);
```

A) true
B) false
C) Compilation Error
D) Runtime Error

Answer: B) false
Explanation: The == operator compares references, not values. s1 and s2 refer to different objects.

Q33. Which of the following methods is used to find the length of a string in Java?

A) length()
B) size()
C) getSize()
D) getLength()

Answer: A) length()
Explanation: The length() method is used to find the length of a string in Java.

Q34. Which interface represents a group of objects in Java as a single unit?

A) Array
B) List
C) Set
D) Map

Answer: B) List
Explanation: The List interface represents a group of objects as a single unit in an ordered collection.

Q35. What is the difference between a Set and a List in Java?

A) A Set allows duplicate elements, while a List does not.
B) A List allows duplicate elements, while a Set does not.
C) A Set is ordered, while a List is unordered.
D) A List is faster than a Set for searching elements.

Answer: B) A List allows duplicate elements, while a Set does not.
Explanation: A List can contain duplicate elements and maintains the insertion order, whereas a Set contains unique elements and does not guarantee any order.

Q36. What is multithreading in Java?

A) Running multiple threads sequentially
B) Running multiple processes in parallel
C) Running multiple threads in parallel
D) Running a single thread multiple times

Answer: C) Running multiple threads in parallel
Explanation: Multithreading in Java is the capability to execute multiple threads simultaneously to maximize the utilization of CPU time.

Q37. Which method is used to start a thread in Java?

A) run()
B) start()
C) execute()
D) begin()

Answer: B) start()
Explanation: The start() method is used to start a newly created thread. It causes the thread to begin execution and the JVM calls the run() method of this thread.

Q38. What is the purpose of the synchronized keyword in Java?

A) To start a new thread
B) To stop a thread
C) To ensure that only one thread can access a resource at a time
D) To check the status of a thread

Answer: C) To ensure that only one thread can access a resource at a time
Explanation: The synchronized keyword is used to control the access of multiple threads to any shared resource and to prevent thread interference.

Q39. What will happen if the sleep() method is called on a thread?

A) The thread will be permanently stopped.
B) The thread will run continuously without any pause.
C) The thread will pause its execution for a specified period.
D) The thread will immediately release all its resources.

Answer: C) The thread will pause its execution for a specified period.
Explanation: The sleep() method is used to pause the execution of the current thread for a specified period of milliseconds.

Q40. Which of the following is true about the wait() and notify() methods in Java?

A) They are used to start and stop threads.
B) They are defined in the Thread class.
C) They are used for inter-thread communication.
D) They can be called outside a synchronized block.

Answer: C) They are used for inter-thread communication.
Explanation: The wait() and notify() methods are used for inter-thread communication and must be called from within a synchronized block.

Q41. What is the purpose of the Comparable interface in Java?

A) To compare the size of two objects
B) To enable object sorting based on natural ordering
C) To compare the hashcodes of two objects
D) To enable multithreading

Answer: B) To enable object sorting based on natural ordering
Explanation: The Comparable interface is used to impose a natural ordering on the objects of the implementing class. It has a compareTo method that is used for sorting objects.

Q42. How does the Comparator interface differ from the Comparable interface in Java?

A) Comparator is used for natural ordering, while Comparable is used for custom ordering.
B) Comparable is used for natural ordering, while Comparator is used for custom ordering.
C) Comparator can compare any two objects, while Comparable can only compare strings.
D) Comparable can compare any two objects, while Comparator can only compare numbers.

Answer: B) Comparable is used for natural ordering, while Comparator is used for custom ordering.
Explanation: The Comparable interface is used for natural ordering of objects (e.g., alphabetical order for strings), while the Comparator interface is used to define custom orderings.

Q43. What is a lambda expression in Java 8?

A) A way to create anonymous classes
B) A method that always returns a value
C) A short block of code that takes parameters and returns a value
D) A special syntax for creating threads

Answer: C) A short block of code that takes parameters and returns a value
Explanation: Lambda expressions are used to provide a clear and concise way to represent one method interface using an expression. They are especially useful in collections filtering and mapping.

Q44. What is the purpose of the Stream API in Java 8?

A) To handle file input and output
B) To create multithreaded applications
C) To perform operations on collections more efficiently
D) To connect to databases

Answer: C) To perform operations on collections more efficiently
Explanation: The Stream API in Java 8 is used to process collections of objects in a functional manner.

Q45. Which of the following is a functional interface in Java 8?

A) Runnable
B) Comparator
C) Both A and B
D) None of the above
Answer: C) Both A and B

Explanation: Both Runnable and Comparator are functional interfaces in Java 8. A functional interface is an interface that contains only one abstract method.

Q46. What is the purpose of the default keyword in Java 8 interfaces?

A) To create default constructors for classes
B) To provide a default implementation for an interface method
C) To set default values for variables
D) To specify default access modifiers

Answer: B) To provide a default implementation for an interface method
Explanation: The default keyword allows an interface to provide a default implementation for a method, which is useful for backward compatibility with older interface versions.

Q47. What is the output of the following lambda expression?

```
(int a, int b) -> a + b
```

A) The sum of two integers
B) A compilation error
C) A string concatenation
D) An integer array

Answer: A) The sum of two integers
Explanation: This lambda expression takes two integers as input and returns their sum.

Q48. Which method reference can be used as an alternative to the lambda expression (s) -> System.out.println(s)?

A) System.out::println
B) System.out.println
C) System::println
D) println::System.out

Answer: A) System.out::println
Explanation: Method references provide a way to refer to methods by their names. System.out::println is a method reference that is equivalent to the lambda expression (s) -> System.out.println(s).

Q49. What is the purpose of the Optional class in Java 8?

A) To provide a container object that may or may not contain a non-null value
B) To create optional method parameters
C) To mark methods as optional in interfaces
D) To handle optional configuration settings

Answer: A) To provide a container object that may or may not contain a non-null value
Explanation: The Optional class is a way of representing optional values in Java. It is used to avoid NullPointerException and to write more readable code by avoiding excessive null checks.

Q50. What feature was introduced in Java 8 to support parallel array processing?

A) Stream API
B) Lambda Expressions
C) ParallelSort
D) ForkJoin Framework

Answer: A) Stream API
Explanation: The Stream API in Java 8 supports parallel array processing, which can improve performance by utilizing multiple cores of the computer's CPU for processing large data sets.

And,let's see some of the Coding Problems asked in different mnc's and let's try to solve them in java.

Coding Questions Asked in Various MNC's

1)Problem Statement – An automobile company manufactures both a two wheeler (TW) and a four wheeler (FW). A company manager wants to make the production of both types of vehicle according to the given data below:(TCS NQT-2023)

- 1st data, Total number of vehicle (two-wheeler + four-wheeler)=v
- 2nd data, Total number of wheels = W

The task is to find how many two-wheelers as well as four-wheelers need to manufacture as per the given data.

Example :
Input :
200 -> Value of V
540 -> Value of W
Output :
TW =130 FW=70
Explanation:
130+70 = 200 vehicles
(70*4)+(130*2)= 540 wheels
Constraints :

- 2<=W
- W%2=0
- V<W

Print "INVALID INPUT" , if inputs did not meet the constraints.

The input format for testing
The candidate has to write the code to accept two positive numbers separated by a new line.

- First Input line – Accept value of V.
- Second Input line- Accept value for W.

The output format for testing

- Written program code should generate two outputs, each separated by a single space character(see the example)
- Additional messages in the output will result in the failure of test case

Solution in Java

```
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
              Scanner sc=new Scanner(System.in);
              int v=sc.nextInt();
              int w=sc.nextInt();
              float res=((4*v)-w)/2;
              if(w>=2 && (w%2==0) && v<w)
              System.out.println("TW= "+(int)(res)+" FW=
"+(int)(v-res));
              else
              System.out.println("INVALID INPUT");
    }
}
```

Explanation:

In this problem, we have to find out how many two-wheelers (TW) and four-wheelers (FW) to manufacture based on the total number of vehicles (v) and total number of wheels (W) given.
We start by taking input values for 'v' and 'W' using a Scanner object. Then, we calculate the number of two-wheelers using the formula ((4 * v) - W) / 2. This formula is derived from the fact that each four-wheeler has 4 wheels and each two-wheeler has 2 wheels.
If the input values meet the constraints (W >= 2, W is even, and v < W), we print the number of two-wheelers and four-wheelers required. Otherwise, we print "INVALID INPUT".
For example, if v=200 and W=540, we calculate TW = ((4 * 200) - 540) / 2 = 130 and FW = 200 - 130 = 70, which means we need to

manufacture 130 two-wheelers and 70 four-wheelers to meet the given conditions.

2)Problem Statement – Given a string S(input consisting) of '*' and '#'. The length of the string is variable. The task is to find the minimum number of '*' or '#' to make it a valid string. The string is considered valid if the number of '*' and '#' are equal. The '*' and '#' can be at any position in the string.(TCS NQT-2023)

Note : The output will be a positive or negative integer based on number of '*' and '#' in the input string.

- (*>#): positive integer
- (#>*): negative integer
- (#=*): 0

Example 1:

Input 1:

- ###*** -> Value of S

Output :

- 0 → number of * and # are equal

Solution:

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

         String str="Hello";
         int count1=0,count2=0;
         for(int i=0;i< str.length();i++)
      {
             if(str.charAt(i)=='*')
                 count1++;##
             else if(str.charAt(i)=='#')
                 count2++;
            }
         System.out.println(count1-count2);
      }
}
```

Explanation:

In this problem, we are given a string consisting of '' *and '#'. We need to find the minimum number of* '' or '#' to make the string valid, meaning that the number of '*' and '#' should be equal.

We start by initializing two variables, count1 and count2, to keep track of the number of '' *and '#' in the string, respectively. We then iterate through each character in the string. If the character is* '', we increment count1. If the character is '#', we increment count2.

Finally, we calculate the difference between count1 and count2. If the result is positive, it means there are more '' *than '#', so the output is a positive integer. If the result is negative, it means there are more '#' than* '', so the output is a negative integer. If the result is zero, it means the string is already valid, so the output is zero.

3)Problem Statement:(Infosys SP/DSE Role)

Your birthday is coming soon and one of your friends, Alex, is thinking about a gift for you. He knows that you really like integer arrays with interesting properties.

He selected two numbers, N and K and decided to write down on paper all integer arrays of length K (in form a[1], a[2], …, a[K]), where every number a[i] is in range from 1 to N, and, moreover, a[i+1] is divisible by a[i] (where 1 < i <= K), and give you this paper as a birthday present.

Alex is very patient, so he managed to do this. Now you're wondering, how many different arrays are written down on this paper?

Since the answer can be really large, print it modulo 10000.
Input:

The first line contains an integer, n, denoting the maximum possible value in the arrays.
The next line contains an integer, k, denoting the length of the arrays.

Input	Output	Output Description
2 1	2	The required length is 1, so there are only two possible arrays: [1] and [2].
2 2	3	All possible arrays are [1, 1], [1, 2], [2, 2]. [2, 1] is invalid because 1 is not divisible by 2.

3 2	5	All possible arrays are [1, 1], [1, 2], [1, 3], [2, 2], [3, 3].

Solution:

```
import java.util.*;
class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int k, n;
        n = sc.nextInt();
        k = sc.nextInt();
        System.out.println(countt(n, k));
    }

    static int counter(int n, int k) {
        int count = 0;
        if (k == 1)
            return n;
        else {
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                    if (j % i == 0)
                        count++;
                }
            }
        }
        return count;
    }

    static int countt(int n, int k) {
        if (k == 1)
            return n;
        if (k == 2) {
            return counter(n, k);
        }
        int mid = k / 2;
        int x = countt(n, k - mid);
        int y = counter(n, mid);
        return x + y - 1;
    }
}
```

Explanation:

In this problem, we need to find the number of different arrays of length K where each element is in the range from 1 to N, and each element is divisible by its previous element.

Let's break down the solution:

The counter method calculates the number of valid arrays of length 2 (k=2) where each element is in the range from 1 to N. It does this by iterating over all pairs of numbers (i, j) from 1 to N and increments the count if j is divisible by i.

The countt method is a recursive method that calculates the total number of valid arrays of length K. If K is 1, then there are N valid arrays (one for each number from 1 to N). If K is 2, then it calls the counter method to get the count. For K greater than 2, it recursively calculates the count by splitting the array into two parts (of lengths K/2 and K-K/2) and calculates the count for each part separately.

The main method takes input values for N and K, and then calls the countt method to calculate the total number of valid arrays of length K. Finally, it prints the result modulo 10000.

That’s is the complete guide for Learning Java.If you prepare the concepts and practice them well,Then you are on your way to become a good software engineer.Along with java learn database management(sql)and LINUX,You are good to go!!!.

Here is a Quick cheat sheet,For your reference.

THE JAVA LANGUAGE CHEAT SHEET

Primitive Types:

INTEGER: byte(8bit),short(16bit),int(32bit),
long(64bit),**DECIM:**float(32bit),double(64bit)
,**OTHER:** boolean(1bit), char (Unicode)
HEX:0x1AF,**BINARY:**0b00101,**LONG:**8888888888888**L**
CHAR EXAMPLES: 'a','\n','\t','\'','\\','\"'

Primitive Operators

Assignment Operator: = (ex: int a=5,b=3;)
Binary Operators (two arguments): + - * / %
Unary Operators: + - ++ --
Boolean Not Operator (Unary): !
Boolean Binary: == != > >= < <=
Boolean Binary Only: && ||
Bitwise Operators: ~ & ^ | << >> >>>
Ternary Operator: bool?valtrue:valfalse;

Casting, Conversion

```
int x = (int)5.5; //works for numeric types
int x = Integer.parseInt("123");
float y = Float.parseFloat("1.5");
int x = Integer.parseInt("7A",16); //fromHex
String hex = Integer.toString(99,16);//toHex
//Previous lines work w/ binary, other bases
```

java.util.Scanner, input, output

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt(); //stops at whitespace
String line = sc.nextLine(); //whole line
System.out.println("bla"); //stdout
System.err.print("bla"); //stderr,no newline
```

java.lang.Number types

```
Integer x = 5; double y = x.doubleValue();
double y = (double)x.intValue();
//Many other methods for Long, Double, etc
```

java.lang.String Methods

```
//Operator +, e.g. "fat"+"cat" -> "fatcat"
boolean equals(String other);
int length();
char charAt(int i);
String substring(int i, int j); //j not incl
boolean contains(String sub);
boolean startsWith(String pre);
boolean endsWith(String post);
int indexOf(String p); //-1 if not found
int indexOf(String p, int i); //start at i
int compareTo(String t);
//"a".compareTo("b") -> -1
String replaceAll(String str, String find);
String[] split(String delim);
```

StringBuffer, StringBuilder

StringBuffer is synchronized StringBuilder
(Use StringBuilder unless multithreaded)
Use the .apend(xyz) methods to concat
toString() converts back to String

java.lang.Math

```
Math.abs(NUM),Math.ceil(NUM),Math.floor(NUM)
,Math.log(NUM),Math.max(A,B),Math.min(C,D),
Math.pow(A,B),Math.round(A),Math.random()
```

IF STATEMENTS:

```
if( boolean_value ) { STATEMENTS }
else if( bool )     { STATEMENTS }
else if( ..etc )    { STATEMENTS }
else                { STATEMENTS }
//curly brackets optional if one line
```

LOOPS:

```
while( bool )         { STATEMENTS }
for(INIT;BOOL;UPDATE) { STATEMENTS }
//1INIT 2BOOL 3STATEMENTS 4UPDATE 5->Step2
do{ STATEMENTS }while( bool );
//do loops run at least once before checking
break;    //ends enclosing loop (exit loop)
continue; //jumps to bottom of loop
```

ARRAYS:

```
int[] x = new int[10]; //ten zeros
int[][] x = new int[5][5]; //5 by 5 matrix
int[] x = {1,2,3,4};
x.length; //int expression length of array
int[][] x = {{1,2},{3,4,5}}; //ragged array
String[] y = new String[10]; //10 nulls
//Note that object types are null by default

//loop through array:
for(int i=0;i<arrayname.length;i++) {
  //use arrayname[i];
}

//for-each loop through array
int[] x = {10,20,30,40};
for(int v : x) {
  //v cycles between 10,20,30,40
}

//Loop through ragged arrays:
for(int i=0;i<x.length;i++)
  for(int j=0;j<x[i].length;j++) {
     //CODE HERE
  }

//Note, multi-dim arrays can have nulls
//in many places, especially object arrays:
Integer[][] x = {{1,2},{3,null},null};
```

FUNCTIONS / METHODS:

```
Static Declarations:
public static int functionname( … )
private static double functionname( … )
static void functionname( … )
Instance Declarations:
public void functionname( … )
private int functionname( … )
Arguments, Return Statement:
int myfunc(int arg0, String arg1) {
  return 5; //type matches int myfunc
}
//Non-void methods must return before ending
//Recursive functions should have an if
//statement base-case that returns at once
```

CLASS/OBJECT TYPES:

```
INSTANTIATION:
public class Ball {//only 1 public per file
  //STATIC FIELDS/METHODS
  private static int numBalls = 0;
  public static int getNumBalls() {
    return numBalls;
  }
  public static final int BALLRADIUS = 5;

  //INSTANCE FIELDS
  private int x, y, vx, vy;
  public boolean randomPos = false;

  //CONSTRUCTORS
  public Ball(int x, int y, int vx, int vy)
  {
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    numBalls++;
  }
  Ball() {
    x = Math.random()*100;
    y = Math.random()*200;
    randomPos = true;
  }

  //INSTANCE METHODS
  public int getX(){ return x; }
  public int getY(){ return y; }
  public int getVX(){ return vx; }
  public int getVY(){ return vy; }
  public void move(){ x+=vx; y+=vy; }
  public boolean touching(Ball other) {
    float dx = x-other.x;
    float dy = y-other.y;
    float rr = BALLRADIUS;
    return Math.sqrt(dx*dx+dy*dy)<rr;
  }

}

//Example Usage:
public static void main(String[] args) {
  Ball x = new Ball(5,10,2,2);
  Ball y = new Ball();
  List<Ball> balls = new ArrayList<Ball>();
  balls.add(x); balls.add(y);
  for(Ball b : balls) {
    for(Ball o : balls) {
      if(b != o) { //compares references
        boolean touch = b.touching(o);
      }
    }
  }
}
```

POLYMORPHISM:

Single Inheritance with "extends"

```
class A{ }
class B extends A{ }
abstract class C { }
class D extends C { }
class E extends D
```

Abstract methods

```
abstract class F {
  abstract int bla();
}
class G extends F {
  int bla() {  //required method
   return 5;
  }
}
```

Multiple Inheritance of interfaces with "implements" (fields not inherited)

```
interface H {
  void methodA();
  boolean methodB(int arg);
}
interface I extends H{
  void methodC();
}
interface K {}
class J extends F implements I, K {
  int bla() { return 5; } //required from F
  void methodA(){} //required from H
  boolean methodB(int a) { //req from A
    return 1;
  }
  void methodC(){} //required from I
}
```

Type inference:

```
A x = new B(); //OK
B y = new A(); //Not OK
C z = new C(); //Cannot instantiate abstract
//Method calls care about right hand type
(the instantiated object)
//Compiler checks depend on left hand type
```

GENERICS:

```
class MyClass<T> {
  T value;
  T getValue() { return value; }
}
class ExampleTwo<A,B> {
  A x;
  B y;
}
class ExampleThree<A extends List<B>,B> {
  A list;
  B head;
}
//Note the extends keyword here applies as
well to interfaces, so A can be an interface
that extends List<B>
```

JAVA COLLECTIONS:

List<T>: Similar to arrays

```
ArrayList<T>: Slow insert into middle
//ArrayList has fast random access
LinkedList<T>: slow random access
//LinkedList fast as queue/stack
Stack: Removes and adds from end
```

List Usage:

```
boolean add(T e);
void clear(); //empties
boolean contains(Object o);
T get(int index);
T remove(int index);
boolean remove(Object o);
//remove uses comparator
T set(int index, E val);
Int size();
```

List Traversal:

```
for(int i=0i<x.size();i++) {
       //use x.get(i);
}

//Assuming List<T>:
for(T e : x) {
       //use e
}
```

Queue<T>: Remove end, Insert beginning

```
LinkedList implements Queue
```

Queue Usage:

```
T element(); // does not remove
boolean offer(T o); //adds
T peek(); //pike element
T poll(); //removes
T remove(); //like poll
Traversal: for(T e : x) {}
```

Set<T>: uses Comparable<T> for uniqueness

```
TreeSet<T>, items are sorted
HashSet<T>, not sorted, no order
LinkedHashSet<T>, ordered by insert
Usage like list: add, remove, size
Traversal: for(T e : x) {}
```

Map<K,V>: Pairs where keys are unique

```
HashMap<K,V>, no order
LinkedHashMap<K,V> ordered by insert
TreeMap<K,V> sorted by keys

V get(K key);
Set<K> keySet(); //set of keys
V put(K key, V value);
V remove(K key);
Int size();
Collection<V> values(); //all values

Traversal: for-each w/ keyset/values
```

java.util.PriorityQueue<T>

A queue that is always automatically sorted using the comparable function of an object

```
public static void main(String[] args) {
  Comparator<String> cmp= new LenCmp();
  PriorityQueue<String> queue =
     new PriorityQueue<String>(10, cmp);
  queue.add("short");
  queue.add("very long indeed");
  queue.add("medium");
  while (queue.size() != 0)
    System.out.println(queue.remove());
}
class LenCmp implements Comparator<String> {
  public int compare(String x, String y){
    return x.length() - y.length();
  }
}
```

java.util.Collections algorithms

Sort Example:

```
//Assuming List<T> x
Collections.sort(x); //sorts with comparator
```

Sort Using Comparator:

```
Collections.sort(x, new Comparator<T>{
  public int compareTo(T a, T b) {
    //calculate which is first
    //return -1, 0, or 1 for order:
    return someint;
  }
}
```

Example of two dimensional array sort:

```
public static void main(final String[] a){
   final String[][] data = new String[][] {
     new String[] { "20090725", "A" },
     new String[] { "20090726", "B" },
     new String[] { "20090727", "C" },
     new String[] { "20090728", "D" } };
     Arrays.sort(data,
       new Comparator<String[]>() {
        public int compare(final String[]
entry1, final String[] entry2) {
          final String time1 = entry1[0];
          final String time2 = entry2[0];
          return time1.compareTo(time2);
        }
     });

     for (final String[] s : data) {
       System.out.println(s[0]+" "+s[1]);
     }
   }
}
```

More collections static methods:

```
Collections.max( … ); //returns maximum
Collections.min( … ); //returns maximum
Collections.copy( A, B); //A list into B
Collections.reverse( A ); //if A is list
```