CORE JAVA INTERVIEW QUESTIONS

Q1: How many types of memory areas are allocated by JVM?

Heap: It is the runtime data area in which the memory is allocated to the objects

Stack: It holds Local Variable

Program Counter Register: PC (program counter) register contains the address of the Java virtual machine instructio n currently being executed.

Q2: What is classloader?

Classloader is a subsystem of JVM which is used to load class files.

There are three built-in classloaders in Java:

Application ClassLoader

Bootstrap ClassLoader

Extension ClassLoader

Q3: What is the default value of the local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

Q4: What are the various access specifiers in Java?

There are four access specifiers:

Public: The classes, methods, or variables which are defined as public, can be accessed by any class or method.

Protected: Protected can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.

Default: Default are accessible within the package only. By default, all the classes, methods, and variables are of def ault scope.

Private: The private class, methods, or variables defined as private can be accessed within the class only.

Q5: What is the purpose of static methods and variables? Can we override the static methods?

The methods or variables defined as static are shared among all the objects of the class. The static is the part of the c lass and not of the object.

we do not need to create the object to access such variables

For example,The name of the college is the common attribute to all the students. Therefore, the college name will be defined as static.

The static variable is used to refer to the common property of all objects (that is not unique for each object)

ex: The company name of employees, college name of students, etc. Static variable gets memory only once in the cl ass area at the time of class loading.

No, we can't override static methods.

Q6: What is the constructor?

The constructor is a special type of method that is used to initialize the state of an object.

It is invoked when the class is instantiated, and the memory is allocated for the object.

Every time, an object is created using the new keyword, the default constructor of the class is called.

The name of the constructor must be similar to the class name.

The constructor must not have an explicit return type.

Q7: How many types of constructors are used in Java?

Default Constructor: The purpose of the default constructor is to assign the default value to the objects.

The java compiler creates a default constructor implicitly if there is no constructor in the class.

Parameterized Constructor: The parameterized constructor is the one which can initialize the instance variables with the given values.

In other words, we can say that the constructors which can accept the arguments are called parameterized constructo rs.

Q8: Is constructor inherited? Can you make a constructor final? Can we overload the constructors?Can we make con

structors static?
No, The constructor is not inherited.
No, the constructor can't be final
Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by cha nging the data type of the parameters
Static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created.the compiler will show the compiler error.

Q) constructor chain??
In constructor chain, a constructor is called from another constructor in the same class this process is known as const ructor chaining.
It occurs through inheritance. When we create an instance of a derived class,
all the constructors of the inherited class (base class) are first invoked,
after that the constructor of the calling class (derived class) is invoked.

We can achieve constructor chaining in two ways:
1) Within the same class: If the constructors belong to the same class, we use this
2) From the base class: If the constructor belongs to different classes (parent and child classes),
we use the super keyword to call the constructor from the base class.

Q9: Can we execute a program without main() method?
Yes, we can execute the program without the main method is using static block because it is loaded before the mainc lass.

Q10: What is this keyword in java?
this keyword is a reference variable that refers to the current object.
It can be used to refer to current class properties such as instance methods, variable, constructors, etc.

Q11: What is the Inheritance?
Inheritance is a mechanism by which one object acquires all the properties and behavior of another object of another class.
It is used for Code Reusability and Method Overriding.
Inheritance represents the IS-A relationship which is also known as a parent-child relationship.
There are five types of inheritance in Java.
Single-level inheritance
Multi-level inheritance
Multiple Inheritance
Hierarchical Inheritance
Hybrid Inheritance
Multiple inheritance is not supported in Java through class.

Q12: Why is multiple inheritance not supported in java?
There will be ambiguity which class method to call.

Q13: What is aggregation?
Aggregation is best described as a has-a relationship
Aggregation represents the weak relationship
Aggregation can be defined as the relationship between two classes where the aggregate class contains a reference to the class it owns.

Q14: What is composition?

Holding the reference of a class within some other class is known as composition.
It represents a stronger relationship between two objects.
Example: A class contains students. A student cannot exist without a class. There exists composition between class a nd students.
Aggregation represents the weak relationship whereas composition represents the strong relationship.
For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

Q15: Why does Java not support pointers?
The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe

Q16: What is super in java?
The super keyword in Java is a reference variable that is used to refer to the immediate parent class object

Q17: What is object cloning?
The object cloning is used to create the exact copy of an object. The clone() method of the Object class is used to clo ne an object.

Q18: What is method overloading? Can we overload the main() method?
Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature.
There are 3 ways to do it:
By changing no of arguments
By changing DataTypes
By changing sequence of DataTypes
Yes, we can have any number of main methods in a Java program by using method overloading.

Q19: Why is method overloading not possible by changing the return type in java?
Method overloading is not possible by changing the return type of the program due to avoid the ambiguity. It gives d uplicate method error.

Q20: Can we overload the methods by making them static?
No, We cannot overload the methods by just applying the static keyword because static method belongs to class

Q21: What is method overriding??
Method overriding is one of the way by which java achieve Run Time Polymorphism
Overriding is a feature that allows a subclass or child class to provide a specific
implementation of a method that is already provided by one of its super-classes or parent classes
When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class,
then the method in the subclass is said to override the method in the super-class.
Rules for Method overriding
The method must have the same name as in the parent class.
The method must have the same signature as in the parent class.
Two classes must have an IS-A relationship between them.

Q22: Can we override the static method? Can we override the overloaded method? Can we override the private meth

ods?
No because the static method is the part of the class, and it is bound with class whereas instance method is bound wit h the object,
and static gets memory in class area, and instance gets memory in a heap.
Yes we can override overloaded method
No we cannot override private methods

Q23: What is the final variable?
the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its va lue.
Final methods cannot be overridden but We can overload a final method,
Final class cannot be extended

Q24: What is the final blank variable?
A final variable, not initialized at the time of declaration, is known as the final blank variable.
we have to initialize it by using the class constructor.

Q25: Can you declare the main method as final?Can we declare a constructor as final?Can we declare an interface as final?
Can we declare abstarct method as final?

Yes, We can declare the main method as public static final void main(String[] args){}.
No The constructor can never be declared as final because it is never inherited
No, we cannot declare an interface as final because the interface must be implemented by some class to provide its d efinition.
abstract method cannot be final as we need to override them in the subclass to give its definition.

Q26: What is the abstraction?
Abstraction is a process of hiding the implementation details and showing only functionality to the user.
In Java, there are two ways to achieve the abstraction.
Abstract Class
Interface

Q27: What is the difference between abstraction and encapsulation?
Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

Q28 What is the abstract class?
A class that is declared as abstract is known as an abstract class.
It needs to be extended and its method implemented.
It cannot be instantiated.
It can have abstract methods, non-abstract methods, constructors, and static methods.
It can also have the final methods

Q29.Can you use abstract and final both with a method?
No, because we need to override the abstract method to provide its implementation, whereas we can't override the fi nal method.

Q30.What is the interface?
It can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction.
There can be only abstract methods in the Java interface
interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class.

Q31 Can you declare an interface method static?Can the Interface be final?
No, because methods of an interface are abstract by default, and we can not use static and abstract together.
No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

Q32 What are the differences between abstract class and interface?
An abstract class can have the constructor.
The interface cannot have the constructor.

An abstract class can have static methods.
The interface cannot have static methods.
You can extend one abstract class.
you can implement multiple interfaces.
A Java abstract class can have class members like private, protected, etc.
Members of a Java interface are public by default.

Q33 How many types of exception can occur in a Java program?
Checked Exception: Checked exceptions are the one which are checked at compile-time. For example, SQLExceptio n, ClassNotFoundException, etc.
Unchecked Exception: Unchecked exceptions are the one which are handled at runtime because they can not be chec ked at compile-time.
For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException,etc.
Error: Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionE rror, etc.

Q34 What is finally block?
The "finally" block is used to execute the important code of the program. It is executed whether an exception is hand led or not
The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Q35 What is the difference between throw and throws?
The throw keyword is used to throw an exception explicitly.
The throws keyword is used to declare an exception.
The throw keyword is used within the method.
The throws keyword is used with the method signature.
You cannot throw multiple exceptions.
You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

Q36 What is String Pool?
String pool is the space reserved in the heap memory that can be used to store the strings.
The main advantage of using the String pool is whenever we create a string literal; the JVM checks the "string const ant pool" first.

Q37 Why are the objects immutable in java?
Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin".
If one reference variable changes the value of the object, it will be affected by all the reference variables.
That is why string objects are immutable in java.

Q38. What are the differences between String StringBuffer and StringBuilder?
The String class is immutable.
The StringBuffer class is mutable.
The String is slow and consumes more memory
The StringBuffer is fast and consumes less memory when you cancat strings.
StringBuffer is synchronized, i.e., thread safe. It means two threads can't call the methods of StringBuffer simultane ously.
StringBuilder is non-synchronized,i.e., not thread safe. It means two threads can call the methods of StringBuilder si multaneously.
StringBuffer is less efficient than StringBuilder.
StringBuilder is more efficient than StringBuffer.

Q39 What are the advantages of Java inner classes?
Nested classes represent a special type of relationship that is it can access all the members (data members and metho

ds) of the outer class including private.
It requires less code to write.

Q40 What is a nested class?
The nested class can be defined as the class which is defined inside another class or interface
A nested class can access all the data members of the outer class including private data members and methods
There are two types of nested classes, static nested class, and non-static nested class. The non-static nested class can also be called as inner-class

Q41. Can a class have an interface?
Yes, an interface can be defined within the class. It is called a nested interface.

Q42 What is Garbage Collection?
Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management
The gc() method is used to invoke the garbage collector for cleanup processing.
System.gc(); can be used in finalize method for GC
Finalize is used to perform clean up processing just before an object is garbage collected.It is a method. Daemon thr ead is GC thread

Q43 What do you understand by an IO stream?
The stream is a sequence of data that flows from source to destination. It is composed of bytes. 3 types of streams ar e
System.out: standard output stream
System.in: standard input stream
System.err: standard error stream

Q44 What is serialization?
Serialization in Java is a mechanism of writing the state of an object into a byte stream so that they can be transform ed through the network.
A class can become serializable by implementing the Serializable interface.

Q45 What is the transient keyword?
If you define any data member as transient, it will not be serialized.

Q46 What are wrapper classes?
Wrapper classes are classes that allow primitive types to be accessed as objects.
The process of converting primitives to objects is called autoboxing, and the process of converting objects to primiti ves is called unboxing.
There are eight wrapper classes present in java.lang package is given below.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Q47 What is a singleton class?
Singleton class is the class which can not be instantiated more than once. To make a class singleton, we either make its constructor private

Q48 What is a JavaBean?
it is a reusable software component. A bean encapsulates many objects into one object so that we can access this obj ect from multiple places

Q49 what is Ananomous class?? what is Ananomous inner class?
Class without any name is called Ananomous class. It is defined and instantiated in a single statement
A local inner class without any name is called Ananomous inner class. we cannot define constructor for these.

Q50 difference between == and .equals method??
== operator is used for address comparision and .equals is used for content comparision

Q51 what is SerialVersionID??
whenever an object is serializied the object is given with a version ID number for the object class. This ID is called SerialVersionID

Q52 what is pass By value and pass by reference??
when an object is passed by value a copy of object is passed so even changes made wont affect the original value.
When an object is passed by reference actual object is not passed and any changes made will affect the value as refer ence of object
is passed rather than actual object

MULTITHREADING INTERVIEW QUESTIONS
Q53 What is multithreading?
Multithreading is a process of executing multiple threads simultaneously.
Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performa nce.

Q54 What is the thread?
A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack f rame.
A process may contain multiple threads.

Q55 What are the states in the lifecycle of a Thread?
A thread can have one of the following states during its lifetime:
New: In this state, a Thread class object is created using a new operator, but the thread is not alive.
Thread doesn't start until we call the start() method.
Runnable: In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selec ted by the thread scheduler.
Running: In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
Waiting/Blocked: In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
Dead/Terminated: A thread is in terminated or dead state when the run() method exits.

Q56 What about the daemon threads?
The daemon threads are the low priority threads that provide the background support and services to the user threads .
Daemon thread gets automatically terminated by the JVM if the program remains with the daemon thread only, and all other user threads are ended/died.

Q57 What is the synchronization?
Synchronization is the capability to control the access of multiple threads to any shared resource. It is used:
To prevent thread interference.
To prevent consistency problem.

Q58 What is the purpose of the Synchronized block?

The Synchronized block can be used to perform synchronization on any specific resource of the method.
Only one thread at a time can execute on a particular resource, and all other threads which attempt to enter the synch ronized block are blocked.

Q59 What is the difference between notify() and notifyAll()?
The notify() is used to unblock one waiting thread whereas notifyAll() method is used to unblock all the threads in w aiting state.

Q60 What is the deadlock?
Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread.
In this situation, Neither of the thread executes nor it gets the chance to be executed

Q61 What is Thread Scheduler in java?
In Java, when we create the threads, they are supervised with the help of a Thread Scheduler, which is the part of JV M.
Thread scheduler is only responsible for deciding which thread should be executed.
Thread scheduler uses two mechanisms for scheduling the threads: Preemptive and Time Slicing.

Q62 What is race-condition?
A Race condition is a problem which occurs in the multithreaded programming when various threads execute simult aneously
accessing a shared resource at the same time.
The proper use of synchronization can avoid the Race condition

Q63 how we can achieve multithreading?
we can define a thread in the following two ways:
By extending Thread class: we are missing Inheritance benefits
By implementing Runnable interface: we are able to use the benefits of Inheritance.

Q64)What is context switching?
In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resu med from the same point later.
Context switching enables the multiple processes to share the same CPU.

Q65)When should we interrupt a thread?
We should interrupt a thread when we want to break out the sleep or wait state of a thread.
We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

Q66)What is static synchronization?
If you make any static method as synchronized, the lock will be on the class not on the object.
If we use the synchronized keyword before a method so it will lock the object (one thread can access an object at a ti me)
but if we use static synchronized so it will lock a class (one thread can access a class at a time).

Q67)How is the safety of a thread achieved?
If a method or class object can be used by multiple threads at a time without any race condition, then the class is thre ad-safe.
Thread safety is used to make a program safe to use in multithreaded programming. It can be achieved by the follow ing ways:

Synchronization
Using Volatile keyword
Using a lock based mechanism
Use of atomic wrapper classes

Q68)What is the volatile keyword in java?
Volatile keyword is used in multithreaded programming to achieve the thread safety, as a change in one volatile variable is
visible to all other threads so one variable can be used by one thread at a time.

Q69)What is the difference between Process and Thread?
A process is a self contained execution environment and it can be seen as a program or application
whereas Thread is a single task of execution within the process.

Q70)What is Thread Scheduler and Time Slicing?
Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads.
Once we create and start a thread, it's execution depends on the implementation of Thread Scheduler.

Time Slicing is the process to divide the available CPU time to the available runnable threads.

Q71)Why thread communication methods wait(), notify() and notifyAll() are in Object class?
In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads
that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object,
that's why it's part of Object class.

Q72)What is volatile keyword in Java
When we use volatile keyword with a variable, all the threads read it's value directly from the memory and don't cache it.
This makes sure that the value read is the same as in the memory.

Q73)Which is more preferred – Synchronized method or Synchronized block?
Synchronized block is more preferred way because it doesn't lock the Object,
synchronized methods lock the Object and if there are multiple synchronization blocks in the class,
even though they are not related, it will stop them from execution and put them in wait state to get the lock on Object.

Q74)How to create daemon thread in Java?
Thread class setDaemon(true) can be used to create daemon thread in java.
We need to call this method before calling start() method else it will throw IllegalThreadStateException.

Q75)What is Thread Pool? How can we create Thread Pool in Java?
A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.
A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.

Q76)What is atomic operation? What are atomic classes in Java Concurrency API?
Atomic operations are performed in a single unit of task without interference from other operations.
Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.
int++ is not an atomic operation. So by the time one thread read its value and increment it by one,
another thread has read the older value leading to the wrong result.
To solve this issue, we will have to make sure that increment operation on count is atomic

COLLECTIONS IN JAVA
Q77 What is the benefit of Generics in Collections Framework?
Generics allow us to provide the type of Object that a collection can contain,
so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation.

Q78 What are the basic interfaces of Java Collections Framework?
Set is a collection that cannot contain duplicate elements.
List is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.
A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Q79 What is an Iterator?
The Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection

using iterator() method.
Iterator takes the place of Enumeration in the Java Collections Framework.

Q80 What is difference between Enumeration and Iterator interface?
Enumeration is twice as fast as Iterator and uses very little memory.
Enumeration is very basic and fits basic needs.
But the Iterator is much safer as compared to Enumeration because it always denies other threads to modify the colle ction object which is being iterated by it.

Q81 What is different between Iterator and ListIterator?
We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.

Q82 What are different ways to iterate over a list?
We can iterate over a list in two different ways – using iterator and using for-each loop.

```
List<String> strList = new ArrayList<>();
//using for-each loop
for(String obj : strList){
System.out.println(obj);
}
//using iterator
Iterator<String> it = strList.iterator();
while(it.hasNext()){
String obj = it.next();
System.out.println(obj);
}
```

Using iterator is more thread-safe because it makes sure that if underlying list elements are modified, it will throw C oncurrentModificationException.

Q83 What do you understand by iterator fail-fast property?
Iterator fail-fast property checks for any modification in the structure of the underlying collection everytime we try t o get the next element.
If there are any modifications found, it throws ConcurrentModificationException.
All the implementations of Iterator in Collection classes are fail-fast by design
except the concurrent collection classes like ConcurrentHashMap and CopyOnWriteArrayList.

Q84 What is difference between fail-fast and fail-safe?
Iterator fail-safe property work with the clone of underlying collection, hence it's not affected by any modification i n the collection.
By design, all the collection classes in java.util package are fail-fast whereas collection classes in java.util.concurren t are fail-safe.
Fail-fast iterators throw ConcurrentModificationException whereas fail-safe iterator never throws ConcurrentModifi

cationException.

Q85 How HashMap works in Java?
Node can represent a class having following objects :
int hash
K key
V value
Node next
HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.
hash code, which will usually be an int
The use of a linked list is important because of collisions:
you could have two different keys with the same hash code or two different hash codes that map to the same index.
When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the ind ex to store the key-value pair.
The Entry is stored in the LinkedList, so if there is an already existing entry, it uses equals() method to check if the p assed key already exists,
if yes it overwrites the value else it creates a new entry and stores this key-value Entry.
When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use eq uals() method to find the correct Entry and return its value.
HashMap initial default capacity is 16 and load factor is 0.75.
Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached.
Which means HashMap starts with storing Entry objects in linked list but after the number of items in a hash becom es larger than a certain threshold,
the hash will change from using a linked list to a balanced tree, which will improve the worst case performance from O(n) to O(log n).
In Java 8, HashMap replaces linked list with a binary tree when the number of elements in a bucket reaches certain t hreshold.
While converting the list to binary tree, hashcode is used as a branching variable.
If there are two different hashcodes in the same bucket, one is considered bigger and goes to the right of the tree and other one to the left.
But when both the hashcodes are equal, HashMap assumes that the keys are comparable,
and compares the key to determine the direction so that some order can be maintained.
It is a good practice to make the keys of HashMap comparable.This JDK 8 change applies only to HashMap, Linked HashMap and ConcurrentHashMap.

Q86 What is difference between HashMap and Hashtable?
HASHMAP:
A HashMap contains values based on the key.
It contains only unique elements.
It may have one null key and multiple null values.
It maintains no order.
HashMap is not synchronized so HashMap is fast.

Hashtable:
A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hash code() method.
A Hashtable contains values based on the key.
It contains only unique elements.
It does have not have any null key or value.
It is synchronized so Hashtable is slow.

Q87 What are similarities and difference between ArrayList and Vector?
Both maintains the order of insertion and we can get the elements in the order of insertion.
ArrayList and Vector both allows null values and random access to element using index number.

Vector is synchronized whereas ArrayList is not synchronized.
ArrayList is faster than Vector because it doesn't have any overhead because of synchronization.

Q88 What is difference between Array and ArrayList?
Arrays are fixed-size whereas ArrayList size is dynamic.
Arrays don't provide a lot of features like ArrayList, such as addAll, removeAll, iterator, etc.

Q89 What is difference between ArrayList and LinkedList?
ArrayList is an index based data structure so it provides random access to its elements
LinkedList stores data as list of nodes where every node is linked to its previous and next node.
So even though there is a method to get the element using index,
internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.
Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList
because there is no concept of resizing array or updating index when element is added in middle.
LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous a nd next elements.

Q90 Which collection classes provide random access of its elements?
ArrayList, HashMap, TreeMap, Hashtable, and Vector classes provide random access to its elements.

Q91 Which collection classes are thread-safe?
Vector, Hashtable, Properties and Stack are synchronized classes, so they are thread-safe and can be used in multi-th readed environment.

Q92 What is BlockingQueue?
BlockingQueue is an interface Queue that supports operations that wait for the queue to become non-empty when ret rieving and removing an element,
and wait for space to become available in the queue when adding an element.

Q93 What is Queue and Stack, list their differences?
Queue allows retrieval of element in First-In-First-Out (FIFO) order but it's not always the case.
There is also Deque interface that allows elements to be retrieved from both end of the queue.
The stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.
Stack is a class that extends Vector whereas Queue is an interface.

Q94 What is Comparable and Comparator interface?
The comparable interface has a compareTo(T obj) method which is used by sorting methods
Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argumen t,
it should be implemented in such a way that it returns negative int if the first argument is less than the second one and returns zero if they are equal and positive int if the first argument is greater than the second one.

Q95 What is difference between Comparable and Comparator interface?
Comparable and Comparator interfaces are used to sort collection or array of objects.
Comparable interface can be used to provide single way of sorting whereas Comparator interface is used to provide different ways of sorting.
For using Comparable, Class needs to implement it whereas for using Comparator we don't need to make any chang e in the class.
Comparable interface is in java.lang package whereas Comparator interface is present in java.util package.

Q96 Difference between HashMap and HashSet?
HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements.

Q97 Difference between Iterator and Enumeration?
Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() me thod.
You can not add/remove elements from a collection when using enumerator.
Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.
Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.nex t() etc.

Q98 Difference between Vector and ArrayList?
All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.
Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it i s re-sized.

Q99 Difference between Set and List?
Set is unordered collection where List is ordered collection based on zero based index.
List allow duplicate elements but Set does not allow duplicates.
List does not prevent inserting null elements (as many you like), but Set will allow only one null element.

Q100 what is concurrentHashmap??
ConcurrentHashMap is the class that is similar to HashMap but works fine when you try to modify your map at runti me.

Q101 explain hashmap, LinkedHashMap,TreeMap??
HASHMAP:
A HashMap contains values based on the key.
It contains only unique elements.
It may have one null key and multiple null values.
It maintains no order.
HashMap is fast.
LinkedHashMap:
A LinkedHashMap contains values based on the key.
It contains only unique elements.
It may have one null key and multiple null values.
It is same as HashMap instead maintains insertion order.
TreeMap
A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap c

lass.
It contains only unique elements.
It cannot have null key but can have multiple null values.
It is same as HashMap instead maintains ascending order.

Q102 Explain HashSet,LinkedHashSet,TreeSet?
HashSet :
HashSet stores the elements by using a mechanism called hashing.
HashSet contains unique elements only.
ex: HashSet<String> set=new HashSet<String>();
LinkedHashSet:
Contains unique elements only like HashSet.
Provides all optional set operations, and permits null elements.
Maintains insertion order.
ex: LinkedHashSet<String> al=new LinkedHashSet<String>();
HashSet and LinkedHashSet both allow Null elements to be added.

TreeSet:
Contains unique elements only like HashSet.
Access and retrieval times are quiet fast.
Maintains ascending order.
ex: TreeSet<String> al=new TreeSet<String>();

Q103 How to convert ArrayList to Array and Array to ArrayList?
We can convert an Array to ArrayList by using the asList() method of Arrays class.
Arrays.asList(item)
We can convert an ArrayList to Array using toArray() method of the ArrayList class.
List_object.toArray(new String[List_object.size()])

Q104 How to make Java ArrayList Read-Only?
java ArrayList Read-only by calling the Collections.unmodifiableCollection() method.
When we define an ArrayList as Read-only then we cannot perform any modification in the collection through add() , remove() or set() method.

Q105 what happens if we try to add null in treeset?
TreeSet adds elements to it according to their natural order. This internally compares the elements with each other us ing the compareTo (or compare) method.
If you try to compare any object with a null value using one of these methods, a NullPointerException will be throw n

Q106 What is Try with Resources?
Support for try-with-resources – introduced in Java 7 – allows us to declare resources to be used in a try block with t he assurance
that the resources will be closed when after the execution of that block. The resources declared must implement the AutoCloseable interface
Execution with try block

```
Scanner scanner = null;
try {
scanner = new Scanner(new File("test.txt"));
while (scanner.hasNext()) {
System.out.println(scanner.nextLine());
}
} catch (FileNotFoundException e) {
e.printStackTrace();
} finally {
if (scanner != null) {
scanner.close();
}
}
```

super succinct solution using try-with-resources:

```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
while (scanner.hasNext()) {
System.out.println(scanner.nextLine());
}
} catch (FileNotFoundException fnfe) {
fnfe.printStackTrace();
}
```

Q107 Why Map interface does not extend Collection interface?
Collection has a method add(Object o). Map can not have such method because it need key-value pair. So they are i ncompatible

Q108 Why Collection doesn't extend the Cloneable and Serializable interfaces?
The Collection interface in Java specifies a group of objects called elements.
The maintainability and ordering of elements is completely dependent on the concrete implementations provided by each of the Collection.
Thus, there is no use of extending the Cloneable and Serializable interfaces.

Q109 List down the major advantages of the Generic Collection.
Provides stronger type checks at the time of compilation
Eliminates the need for typecasting
Enables the implementation of generic algorithms which makes the code customizable, type-safe and easier to read

Q110 What is Java Priority Queue ?
The PriorityQueue is an unbounded queue, based on a priority heap
A PriorityQueue doesn't allow null values,

JAVA 8
Q111 What are the main features of java 8??
New features in java 8.
1. Lambda Expression
2. Functional Interface
3. Default Methods
4. Streams
5. Date/Time API Changes
6. forEach loop

Q112 what is Lambda Expression?
Lambda expression (or function) is just an anonymous function,
i.e., a function with no name and without being bounded to an identifier.
They are written exactly in the place where it's needed.
example: (x, y) -> x + y //This function takes two parameters and return their sum.

Q113 what is Functional Interface?
Functional interfaces are also called Single Abstract Method interfaces (SAM Interfaces).
They permit exactly one abstract method inside them. Java 8 introduces an annotation i.e.
@FunctionalInterface which can be used for compiler level errors
example:

```
@FunctionalInterface
public interface MyFirstFunctionalInterface {
public void firstWork();
}
```

Functional interface is valid even if the @FunctionalInterface annotation would be omitted.
It is only for informing the compiler to enforce single abstract method inside interface

Q114 What is Default Methods?
Java 8 allows you to add non-abstract methods in interfaces. These methods must be declared default methods.
Example:

```
public interface Moveable {
default void move(){
System.out.println("I am moving");
}
}
public class Animal implements Moveable{
public static void main(String[] args){
```

```
Animal tiger = new Animal();
tiger.move();
}
}
```
Output: I am moving

Q115 what are Streams??
Java 8 Streams API, which provides a mechanism for processing a set of data in various ways that can include filteri
ng, transformation,
or any other way that may be useful to an application.
example:
```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Yana");
memberNames.add("Lokesh");
memberNames.stream().filter((s) -> s.startsWith("A"))
.forEach(System.out::println);
```
Output:
Amitabh
Aman

Q116 what are java 8 Date/Time API Changes?
In java 8 we have 3 classes
The LocalDate class represents a date. There is no representation of a time or time-zone.
The LocalTime class represents a time. There is no representation of a date or time-zone.
The LocalDateTime class represents a date-time. There is no representation of a time-zone.
For setting up zone we have Zone-id provided in java 8
example:
```
LocalDate localDate = LocalDate.now();
LocalTime localTime = LocalTime.of(12, 20);
LocalDateTime localDateTime = LocalDateTime.now();
OffsetDateTime offsetDateTime = OffsetDateTime.now();
ZonedDateTime zonedDateTime = ZonedDateTime.now(ZoneId.of("Europe/Paris"));
```
Timestamp and Duration:
The Instant class represents an instant in time to an accuracy of nanoseconds.
```
Instant instant = Instant.now();
Instant instant1 = instant.plus(Duration.ofMillis(5000));
Instant instant2 = instant.minus(Duration.ofMillis(5000));
Instant instant3 = instant.minusSeconds(10);
```
Duration class represents the time difference between two time stamps.
```
Duration duration = Duration.ofMillis(5000);
duration = Duration.ofSeconds(60);
duration = Duration.ofMinutes(10);
```
Duration deals with small unit of time such as milliseconds, seconds, minutes and hour.

Q117 What is forEach loop used for in java 8??
The Java forEach is a utility method to iterate over a collection or stream and perform a certain action on each eleme

nt of it.

example:

```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Yana");
memberNames.add("Lokesh");
memberNames.stream().filter((s) -> s.startsWith("A"))
.forEach(System.out::println);
```

Output:
Amitabh
Aman

Q118 how u convert streams to collections?
Stream.collect(Collectors.toList()) is used to convert streams to collection
example:

```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Yana");
memberNames.add("Lokesh");
List<String> memNamesInUppercase = memberNames.stream().sorted()
.map(String::toUpperCase)
.collect(Collectors.toList());
System.out.print(memNamesInUppercase);
```

Outpout: [AMAN, AMITABH, LOKESH, RAHUL, SALMAN, SHAHRUKH, SHEKHAR, YANA]

Q119 how u convert stream to Array?
Stream.toArray(EntryType[]::new) is used to convert streams to array
example:

```
public class StreamBuilders {
public static void main(String[] args){
List<Integer> list = new ArrayList<Integer>();
for(int i = 1; i< 10; i++){
list.add(i);
}
Stream<Integer> stream = list.stream();
Integer[] evenNumbersArr = stream.filter(i -> i%2 == 0).toArray(Integer[]::new);
System.out.print(evenNumbersArr);
}
}
```

Q120 explain different Core stream operations?
let's explain all the operations with example:

```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
```

```
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Yana");
memberNames.add("Lokesh")
```

1) Stream.filter(): Filter accepts a predicate to filter all elements of the stream.

```
memberNames.stream().filter((s) -> s.startsWith("A"))
.forEach(System.out::println);
```

Output:

Amitabh

Aman

2) Stream.map(): The intermediate operation map converts each element into another object via the given function. The following example converts each string into an upper-cased string.

```
memberNames.stream().filter((s) -> s.startsWith("A"))
.map(String::toUpperCase)
.forEach(System.out::println);
```

Output:

AMITABH

AMAN

3) stream.sorted()

Sorted is an intermediate operation which returns a sorted view of the stream. The elements are sorted in natural ord er unless you pass a custom Comparator.

```
memberNames.stream().sorted()
.map(String::toUpperCase)
.forEach(System.out::println);
```

Output:

AMAN

AMITABH

LOKESH

RAHUL

SALMAN

SHAHRUKH

SHEKHAR

YANA

4) Stream.forEach()

This method helps in iterating over all elements of a stream and perform some operation on each of them. The operation is passed as lambda expression parameter.

example:

```
memberNames.forEach(System.out::println);
```

5) Stream.collect()

collect() method used to receive elements from a steam and store them in a collection

```
List<String> memNamesInUppercase = memberNames.stream().sorted()
.map(String::toUpperCase)
.collect(Collectors.toList());
System.out.print(memNamesInUppercase);
```

Outpout: [AMAN, AMITABH, LOKESH, RAHUL, SALMAN, SHAHRUKH, SHEKHAR, YANA]

6) Stream.match()

Various matching operations can be used to check whether a certain predicate matches the stream. All of those opera

tions are terminal and return a boolean result.

```
boolean matchedResult = memberNames.stream()
.anyMatch((s) -> s.startsWith("A"));
System.out.println(matchedResult);
```

```
matchedResult = memberNames.stream()
.allMatch((s) -> s.startsWith("A"));
System.out.println(matchedResult);
matchedResult = memberNames.stream()
.noneMatch((s) -> s.startsWith("A"));
System.out.println(matchedResult);
```

Output:
true
false
false

7) Stream.count()

Count is a terminal operation returning the number of elements in the stream as a long.

```
long totalMatched = memberNames.stream()
.filter((s) -> s.startsWith("A"))
.count();
System.out.println(totalMatched);
```

Output: 2

8) Stream.findFirst()

It will return first element from stream and then will not process any more element.

```
String firstMatchedName = memberNames.stream()
.filter((s) -> s.startsWith("L"))
.findFirst().get();
System.out.println(firstMatchedName);
```

Output: Lokesh

The stream functions are:

Intermediate Operations
filter()
map()
flatMap()
distinct()
sorted()
peek()
limit()
skip()

Terminal Operations
forEach()
forEachOrdered()
toArray()
reduce()
collect()
min()
max()
count()
anyMatch()
allMatch()
noneMatch()
findFirst()
findAny()

Q121 what is Java Boxed Stream?

In Java 8, if you want to convert stream of objects to collection, then you can use one of the static methods in the Co

llectors class.
//It works perfect !!
List<String> strings = Stream.of("how", "to", "do", "in", "java")
.collect(Collectors.toList());
The same process doesn't work on streams of primitives, however.
//Compilation Error !!
IntStream.of(1,2,3,4,5)
.collect(Collectors.toList());
To convert a stream of primitives, you must first box the elements in their wrapper class and then collect them. This type of stream in called boxed stream.
example:
convert int stream to List of Integers.
//Get the collection and later convert to stream to process elements
List<Integer> ints = IntStream.of(1,2,3,4,5)
.boxed()
.collect(Collectors.toList());
System.out.println(ints);
//Stream operations directly
Optional<Integer> max = IntStream.of(1,2,3,4,5)
.boxed()
.max(Integer::compareTo);
System.out.println(max);
Program Output:
[1, 2, 3, 4, 5]
5
Example to convert long stream to List of Longs.
List<Long> longs = LongStream.of(1l,2l,3l,4l,5l)
.boxed()
.collect(Collectors.toList());
System.out.println(longs);
Output:
[1, 2, 3, 4, 5]
List<Double> doubles = DoubleStream.of(1d,2d,3d,4d,5d)
.boxed()
.collect(Collectors.toList());
System.out.println(doubles);
Output:
[1.0, 2.0, 3.0, 4.0, 5.0]

Q122 Explain lambda expression in depth?
Lambda expression (or function) is just an anonymous function, i.e., a function with no name and without being bou nded to an identifier.
features of lambda expression:
A lambda expression can have zero, one or more parameters.
The type of the parameters can be explicitly declared or it can be inferred from the context.
Multiple parameters are enclosed in mandatory parentheses and separated by commas. Empty parentheses are used t o represent an empty set of parameters.
When there is a single parameter, if its type is inferred, it is not mandatory to use parentheses. e.g. a -> return a*a.
The body of the lambda expressions can contain zero, one or more statements.
If body of lambda expression has single statement curly brackets are not mandatory and the
return type of the anonymous function is the same as that of the body expression.
When there is more than one statement in body than these must be enclosed in curly brackets.
example:

```
List<String> memberNames = new ArrayList<>();
memberNames.add("Amitabh");
memberNames.add("Shekhar");
memberNames.add("Aman");
memberNames.add("Rahul");
memberNames.add("Shahrukh");
memberNames.add("Salman");
memberNames.add("Yana");
memberNames.add("Lokesh");
memberNames.stream().filter((s) -> s.startsWith("A"))
.forEach(System.out::println);
```

Output:
Amitabh
Aman

Q123 explain Functional Interfaces in depth?
Functional interfaces are new additions in java 8 which permit exactly one abstract method inside them.
These interfaces are also called Single Abstract Method interfaces (SAM Interfaces).
Java 8 introduces an annotation i.e. @FunctionalInterface too, which can be used for compiler level errors when the interface you have annotated violates the contracts of exactly one abstract method.

```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
public void firstWork();
}
```

Let’s try to add another abstract method:

```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
public void firstWork();
public void doSomeMoreWork(); //error
}
```

Above will result into compiler error as given below:
Console
Unexpected @FunctionalInterface annotation
@FunctionalInterface ^ MyFirstFunctionalInterface is not a functional interface
multiple non-overriding abstract methods found in interface MyFirstFunctionalInterface
Functional-Interface-Error
A functional interface is valid even if the @FunctionalInterface annotation would be omitted.
It is only for informing the compiler to enforce single abstract method inside interface.
default methods have an implementation, they are not abstract.
Since default methods are not abstract you’re free to add default methods to your functional interface as many as yo u like.
Below is valid functional interface:

```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
public void firstWork();
default void doSomeMoreWork1(){
//Method body
}
default void doSomeMoreWork2(){
//Method body
```

```
}
}
```
If an interface declares an abstract method overriding one of the public methods of java.lang.Object, that also does not count toward the interface's abstract method count since any implementation of the interface will have an implementation from java.lang.Object or elsewhere.
Below interface is a valid functional interface.
```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
public void firstWork();
@Override
public String toString(); //Overridden from Object class
@Override
public boolean equals(Object obj); //Overridden from Object class
}
```


Q124 explain Java 8 method reference?
In Java 8, we can refer a method from class or object using class::methodName type syntax.
Java 8 allows four types of method references.
static method: example Math::max equivalent to Math.max(x,y)
instance method from instance: example System.out::println equivalent to System.out.println(x)
instance method from class type: example String::length equivalent to str.length()
Reference to constructor: example ArrayList::new equivalent to new ArrayList()
1) Class::staticMethodName
An example to use Math.max() which is static method.
```
List<Integer> integers = Arrays.asList(1,12,433,5);
Optional<Integer> max = integers.stream().reduce( Math::max );
max.ifPresent(value -> System.out.println(value));
```
Output:
433
2) ClassInstance::instanceMethodName
In above example, we use System.out.println(value) to print the max value found. We can use System.out::println to print the value.
```
List<Integer> integers = Arrays.asList(1,12,433,5);
Optional<Integer> max = integers.stream().reduce( Math::max );
max.ifPresent( System.out::println );
```
Output:
433
3) Class::instanceMethodName
In this example, s1.compareTo(s2) is referred as String::compareTo.
```
List<String> strings = Arrays
.asList("how", "to", "do", "in", "java", "dot", "com");
List<String> sorted = strings
.stream()
.sorted((s1, s2) -> s1.compareTo(s2))
.collect(Collectors.toList());
System.out.println(sorted);
List<String> sortedAlt = strings
.stream()
.sorted(String::compareTo)
.collect(Collectors.toList());
System.out.println(sortedAlt);
```
Output:

[com, do, dot, how, in, java, to]
[com, do, dot, how, in, java, to]
4) Reference to constructor – Class::new
The first method can be updated to create a list of integers from 1 to 100. Using lambda expression is rather easy.
To create a new instance of ArrayList, we have use ArrayList::new.

```
List<Integer> integers = IntStream
.range(1, 100)
.boxed()
.collect(Collectors.toCollection( ArrayList::new ));
Optional<Integer> max = integers.stream().reduce(Math::max);
max.ifPresent(System.out::println);
```

Output:
99

Q125 What are default methods in java 8? explain
we can add non-abstarct methods in our interface with help of default methods.
example:

```
public interface Moveable {
default void move(){
System.out.println("I am moving");
}
}
```

Moveable interface defines a method move(); and provided a default implementation as well.
If any class implements this interface then it need not to implement it's own version of move() method.
It can directly call instance.move();

```
public class Animal implements Moveable{
public static void main(String[] args){
Animal tiger = new Animal();
tiger.move();
}
}
```

Output: I am moving
And if class willingly wants to customize the behavior then it can provide it's own custom implementation and overr

ide the method.
Now it's own custom method will be called.

```
public class Animal implements Moveable{
public void move(){
System.out.println("I am running");
}
public static void main(String[] args){
Animal tiger = new Animal();
tiger.move();
}
}
```

Output: I am running

Q126 Why default methods were needed in java 8?
To enable the functionality of lambda expression in java.
Before java 8, if you had to iterate on a java collection then your would get an iterator instance and call it's next met hod until hasNext() returns false.
Now to iterate and perform some simple operation on every item in list, all you need to do is:

```
import java.util.ArrayList;
import java.util.List;
public class Animal implements Moveable{
public static void main(String[] args){
List<Animal> list = new ArrayList();
list.add(new Animal());
list.add(new Animal());
list.add(new Animal());
//Iterator code reduced to one line
list.forEach((Moveable p) -> p.move());
}
}
```


Q127 how to Parse String to Date in java 8?

1) If you have any string which represent date in ISO8601 format then you can use LocalDate.parse() or LocalDateTi

me.parse() methods directly.

```
String armisticeDate = "2016-04-04";
LocalDate aLD = LocalDate.parse(armisticeDate);
System.out.println("Date: " + aLD);
String armisticeDateTime = "2016-04-04T11:50";
LocalDateTime aLDT = LocalDateTime.parse(armisticeDateTime);
System.out.println("Date/Time: " + aLDT);
```

Output:

Date: 2016-04-04

Date/Time: 2016-04-04T11:50

2) Convert string to date in custom formats

If you have dates in some custom format, then you need to put additional logic to handle formatting as well using Da

teTimeFormatter.ofPattern().

```
String anotherDate = "04 Apr 2016";
DateTimeFormatter df = DateTimeFormatter.ofPattern("dd MMM yyyy");
LocalDate random = LocalDate.parse(anotherDate, df);
System.out.println(anotherDate + " parses as " + random);
```

Q128 What are SOLID Principles in Java??

The following five concepts make up our SOLID principles:

1) Single Responsibility: A class should only have one responsibility.

2) Open/Closed: classes should be open for extension but closed for modification.

In doing so, we stop ourselves from modifying existing code and causing potential new bugs in an otherwise happy application.

Of course, the one exception to the rule is when fixing bugs in existing code.

3) Liskov Substitution: if class A is a subtype of class B, we should be able to replace B with A without disrupting t he behavior of our program

4) Interface Segregation: larger interfaces should be split into smaller ones.

By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of intere st to them.

5) Dependency Inversion: The principle of dependency inversion refers to the decoupling of software modules.

This way, instead of high-level modules depending on low-level modules, both will depend on abstractions

Q129) How Would u get millions of data at the same time in Java??

1) Use Pagination

2) DataBase Partition

Q130) How would u avoid OutOfMemoryError in Production environment in Java??

You should not handle it in code. OutOfMemory should not be caught and handled. Instead start your JVM with a bi gger heapspace

java -Xmx512M

Another way is to use visual vm to avoid space issue

Q131) What are Optionals in Java 8??

Optional is a container object used to contain not-null objects.
This class has various utility methods to facilitate code to handle values as 'available' or 'not available' instead of checking null values. It is introduced in Java 8 and is similar to what Optional is in Guava.

Q132) How would u enable Redis Cache in Java??

1) use @EnableCaching In the Main Class
2) use @Cacheable(key="#p0",value="ResponseDto") in the controller methods which we want to cache
1st time it will get the data from DB and 2nd time onwards it will get data from cache

Q133) What are differences between comparable and comparator in Java??

Comparable provides a single sorting sequence.
In other words, we can sort the collection on the basis of a single element such as id, name, and price

The Comparator provides multiple sorting sequences.
In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.

Comparable affects the original class, i.e., the actual class is modified.
Comparator doesn't affect the original class, i.e., the actual class is not modified.

Comparable provides compareTo() method to sort elements.
Comparator provides compare() method to sort elements.

Q134) How to make class immutable??

Declare the class as final so it can’t be extended.
Make all fields private so that direct access is not allowed.
Don’t provide setter methods for variables.
Make all mutable fields final so that its value can be assigned only once

Q135) What is difference between JAR WAR EAR??
The JAR file is a file that has Java class files, related metadata, and resource combined into a single file to execute a Java application
EAR is a Java EE file that packages one or more modules into a single archive to deploy them on to an application s erver
WAR files are the files that contain Servlet, JSP, HTML, JavaScript and other files necessary

Q136) Difference between ClassNotFoundException and classdefinationnotfound exception??

ClassNotFoundException
It is an exception
It occurs when an application tries to load a class at run time which is not updated in the classpath.
It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemC lass().

NoClassDefFoundError
It is an error. It is of type java.lang.Error.
It occurs when java runtime system doesn't find a class definition, which is present at compile time, but missing at r un time.
It is thrown by the Java Runtime System.

Q137) min heap and max heap in java??

A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
Since a heap is a complete binary tree, a heap with N nodes has log N height.
It is useful to remove the highest or lowest priority element. It is typically represented as an array.

There are two types of Heaps in the data structure.

In a Min-Heap the key present at the root node must be less than or equal among the keys present at all of its childre n
In a Min-Heap the minimum key element present at the root.
The same property must be recursively true for all sub-trees in that Binary Tree.

In a Max-Heap the key present at the root node must be greater than or equal among the keys present at all of its chil dren.
The same property must be recursively true for all sub-trees in that Binary Tree.
In a Max-Heap the maximum key element present at the root.

Q138) What is the difference between protected and default access specifiers?

The protected specifier allows access by all subclasses of the class in question,
whatever package they reside in, as well as to other code in the same package.

The default specifier allows access by other code in the same package, but not by code that is in subclasses residing i n different packages

SPRINGBOOT

Q139) Explain the flow of springBoot Projects

Q140) What are Zuul in springBoot??

Q141) what are circuit breakers in Springboot??

Q142) Explain the annotation used in springBoot??

Q143) Difference between Spring and SpringBoot??

Q144) what are actuators in SpringBoot??
Actuator – to get production-ready features such as monitoring

Q145) what are annotations are present in @SpringBootApplication??
@SpringBootConfiguration: enable @Component scan on the package where the application is located
@EnableAutoConfiguration
@ComponentScan(excludeFilters={@Filter(type=CUSTOM, classes={TypeExcludeFilter.class}),
@Filter(type=CUSTOM, classes={AutoConfigurationExcludeFilter.class})})

Q146) what are differences between SpringBootConfiguration and EnableAutoConfiguration??
@SpringBootConfiguration is a class-level annotation that is part of the Spring Boot framework.
It indicates that a class provides application configuration.

@EnableAutoConfiguration is used for auto-configuring beans present in the classpath in Spring Boot applications.

Q147) What Spring Boot Starters Are Available out There?

Q148) How to Disable a Specific Auto-Configuration?
f we want to disable a specific auto-configuration, we can indicate it using the exclude attribute of the @EnableAutoConfiguration annotation. For instance, this code snippet neutralizes DataSourceAutoConfiguration:

```
// other annotations
@EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)
public class MyConfiguration {}
```

Q149) How to Deploy Spring Boot Web Applications as Jar and War Files?

Maven build will create JAR in Target folder

Q150)What Is Spring Boot Actuator Used For?

Essentially, Actuator brings Spring Boot applications to life by enabling production-ready features.
These features allow us to monitor and manage applications when they're running in production.

Integrating Spring Boot Actuator into a project is very simple.
All we need to do is to include the spring-boot-starter-actuator starter in the pom.xml file:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering,
and many more that are automatically applied to your application.

Q151) Which Embedded Servers does Spring Boot Support, and How to Change the Default?
As of date, Spring MVC supports Tomcat, Jetty, and Undertow. Tomcat is the default application server supported b y Spring Boot's web starter.

Q152) Why Do We Need Spring Profiles?

Q153) How do you Configure Log4j for logging?

Q154) What is the difference between @RestController and @Controller in Spring Boot?
@Controller Map of the model object to view or template
and make it human readable but @RestController simply returns the object and object data is directly written in HT

TP response as JSON or XML.

@RestController= @Controller + @ResponseBody

Q155) What is the difference between RequestMapping and GetMapping?
RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation.
Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

Q156) Name two ways to create a new Spring Boot project from scratch? Also, how do you know what spring-boot-starters your project needs?

Spring Boot CLI
Install Plugin from eclipse marketplace or Spring Initializr web application

Q157) Default HTML template engine in Spring Boot?

Thymeleaf . you can also use many others, like Freemarker, Velocity or even JSP

Q158) How can I reload my Spring Boot changes without restarting the server?
This is achievable by Spring Boot Dev Tools module.it's a powerful tool for development.
It helps developers to shorten the development cycle and enable easy deployment and testing during development.
To enable this feature, add the following dependency to Maven POM file.

```
<dependencies>
   <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-devtools</artifactId>
   </dependency>
</dependencies>
```

Q159) How can we override default properties in Spring Boot?
Spring Boot advocate convention over configuration.
Spring Boot externalize application configurations through application.properties file.
These properties work as default values for the Spring Boot application.
To override these default values, Spring Boot provides the following options.

Create an application.properties file in the classpath for overriding specific properties
for Spring Boot. For Maven based project, this file will be under /src/main/resource.

application.yml file in the classpath for overriding specific properties for Spring Boot.
For Maven based project, this file will be under /src/main/resource.

Q160) What is a shutdown in the actuator?
Spring Boot actuator provides a shutdown endpoint. This endpoint allows the graceful shutdown of the application.
This end point disabled by default and we need to enable this in case we need to use this service.
We can enable this end point using project.properties

management.endpoint.shutdown.enabled=true

Q161) Why do we need spring-boot-maven-plugin?
Spring Boot Maven plugin provides Spring Boot support in the maven.
This plugin provides options to create an executable jar or war files. Here are goals for this plugin.

boot: run runs your Spring Boot application.

spring-boot:repackage repackages your jar/war to be executable.
spring-boot:start and spring-boot:stop to manage the lifecycle of your Spring Boot application (i.e. for integration te sts).
spring-boot:build-info generates build information that can be used by the Actuator.

Q162) What is the use of YAML in Spring Boot?
YAML is a superset of JSON. Spring Boot YAML as an alternative to the application.properties file to define your p roject properties.

Let's take the following example of the application.properties file.

environments.dev.url=https://dev.javadevjournal.com
environments.dev.name=Developer Setup
It can represent the YAML files as follows.


```
environments:
  dev:
    url: https://dev.javadevjournal.com
    name: Developer Setup
```

Q163) How to configure and enable SSL for your Spring Boot application?
Use the server.ssl.* properties in the application.properties or yml file to configure and enable SSL for your Spring Boot application.
Here are typical SSL configurations for your application.

```
server.port=8443 //SSL port
server.ssl.key-store=classpath:keystore.jks //You can also configure it to external location
server.ssl.key-store-password= //password for your key
server.ssl.key-password=//key password
```

Remember, Spring Boot does not support configuration of both HTTP and HTTPS through the property file.

Q164) What are qualifier in springBoot??

The @Qualifier annotation is used to resolve the autowiring conflict, when there are multiple beans of same type.

The @Qualifier annotation can be used on any class annotated with @Component or on method annotated with @B ean.
This annotation can also be applied on constructor arguments or method parameters.

Vehicle.java

```
package com.boraji.tutorial.spring.bean;

public interface Vehicle {

  public void start();

  public void stop();
}
```

Car.java

```
package com.boraji.tutorial.spring.bean;

import org.springframework.stereotype.Component;

@Component
public class Car implements Vehicle {

  @Override
  public void start() {
    System.out.println("Car started");
  }

  @Override
  public void stop() {
    System.out.println("Car stopped");
  }

}
```

Bike.java

```
package com.boraji.tutorial.spring.bean;

import org.springframework.stereotype.Component;

@Component
public class Bike implements Vehicle{

  @Override
  public void start() {
    System.out.println("Bike started");
  }

  @Override
  public void stop() {
    System.out.println("Bike stopped");
  }
}
```

VehicleService.java

```
@Component
public class VehicleService {

  @Autowired
  private Vehicle vehicle;

  public void service() {
    vehicle.start();
    vehicle.stop();
  }
}
```

it gives error since it dont which which start to call

Modify the Bike, Car and VehicleService beans as follow.

Car.java

```
@Component
@Qualifier("carBean")
public class Car implements Vehicle {

   //...
   //...
}
```


Bike.java

```
@Component
@Qualifier("bikeBean")
public class Bike implements Vehicle{

   //...
   //...
}
```

VehicleService.java

```
@Component
public class VehicleService {

   @Autowired
   @Qualifier("carBean")
   private Vehicle vehicle;

   //...
   //...
}
```

Now run the MainApp class again and see output.

Output

```
Car started
Car stopped
```

Q165) If i create object parent p = new child which object will be geting created??
if i call method p.parentmethod and p.childmethod which will be called??

Parent p = new Child();
It is called as up-casting. Object 'p' has limited access. It can only access parent class properties,because p is a refern ce to parent class.

Hence accessing 'p.age' was not possible eventhough p is a child object.

Q166)In which index null key sets in hashmap??

null key actually stored at index 0 in hashmap

Q167) How to stop calling 2nd method from an interface if we have 2 methods??

```
public interface a {
   public void m1();
   public void m2();
   public void m3();
}

public class A implements a {
   public void m3() {
      // implementation code
   }
}
```


I want to avoid implementation for the rest of the method???

```
public abstract class A implements a{

      public void m3(){

         // implementation code

         }

}
```

Declare as abstract class so you will not needed to implement these methods in this class.
But you have to implement those method in concrete class

Q168) what is reflection API??

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.
Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

Drawbacks:

Performance Overhead: Reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code which are called frequently in performance-sensitive applications.

Exposure of Internals: Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.

Q169) parallel streams java8??

Java Parallel Streams is a feature of Java 8 and higher, meant for utilizing multiple cores of the processor.
Normally any java code has one stream of processing, where it is executed sequentially.
Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes.

Therefore, it is advisable to use parallel streams in cases where no matter what is the order of execution,
the result is unaffected and the state of one element does not affect the other as well as the source of the data also re
mains unaffected.

Q170) concurrenthashmap internal working??

ConcurrentHashMap utilizes the same principles of HashMap, but is designed primarily for a multi-threaded applica
tion
and hence it does not require explicit synchronization.

Why we need ConcurrentHashMap when we already had Hashtable ?

Hashtable provides concurrent access to the Map.Entries objects by locking the entire map to perform any sort of op
eration (update,delete,read,create).
Suppose we have a web application , the overhead created by Hashtable (locking the entire map) can be ignored und
er normal load.
But under heavy load , the overhead of locking the entire map may prove fatal and may lead to delay response time a
nd overtaxing of the server.

ConcurrentHashMap class is fully interoperable with Hashtable in programs that rely on its thread safety but not on i
ts synchronization details.
So the main purpose of this class is to provide the same functionality as of Hashtable but with a performance compar
able to HashMap.

The constructor of ConcurrentHashMap looks like this :

public ConcurrentHashMap (int initialCapacity, float loadFactor, int concurrencyLevel)

initialCapacity - the initial capacity. The implementation performs internal sizing to accommodate this many elemen
ts.

concurrencyLevel - the estimated number of concurrently updating threads.
The implementation performs internal sizing to try to accommodate this many threads.

In the ConcurrentHashMap Api , you will find the following constants.

static final int DEFAULT_INITIAL_CAPACITY = 16;
static final int DEFAULT_CONCURRENCY_LEVEL = 16;

initial capacity parameter and concurrency level parameters of ConcurrentHashMap constructor (or Object) are set t
o 16 by default.

Thus, instead of a map wide lock, ConcurrentHashMap maintains a list of 16 locks by default
(number of locks equal to the initial capacity , which is by default 16) each of which is used to lock on a single buc
ket of the Map.
This indicates that 16 threads (number of threads equal to the concurrency level ,
which is by default 16) can modify the collection at the same time , given ,each thread works on different bucket.
So unlike hashtable, we perform any sort of operation (update ,delete ,read ,create) without locking on entire map in
ConcurrentHashMap.

Can two threads update the ConcurrentHashMap simultaneously ?

Yes it is possible that two threads can simultaneously write on the ConcurrentHashMap.
ConcurrentHashMap default implementation allows 16 threads to read and write in parallel.

Why ConcurrentHashMap does not allow null keys and null values ?

The main reason that nulls aren't allowed in ConcurrentMaps (ConcurrentHashMaps, ConcurrentSkipListMaps) is th
at ambiguities
The main one is that if map.get(key) returns null, you can't detect whether the key explicitly maps to null vs the key isn't mapped.

What is the difference between HashMap and ConcurrentHashMap?

The HashMap was not thread safe and therefore could not be utilized in multi-threaded applications.
The ConcurrentHashMap was introduced to overcome this shortcoming and also as an alternative to using HashTabl
e
and synchronized Maps for greater performance and uses the standard Hashing algorithms to generate hash code for storing the key value pairs.

Does ConcurrentHashMap Iterator behaves like fail fast iterator or fail safe Iterator?

ConcurrentHashMap iterator behaves like fail safe iterator. It will not throw ConcurrentModificationException .

Can you write the simple example which proves ConcurrentHashMap class behaves like fail safe iterator?

```
public class ConcurrentHashMapExample
{


   public static void main(String[] args)
   {
      ConcurrentHashMap<String,String> premiumPhone = new ConcurrentHashMap<String,String>();
      premiumPhone.put("Apple", "iPhone6");
      premiumPhone.put("HTC", "HTC one");
      premiumPhone.put("Samsung","S6");

      Iterator iterator = premiumPhone.keySet().iterator();

      while (iterator.hasNext())
      {
         System.out.println(premiumPhone.get(iterator.next()));
         premiumPhone.put("Sony", "Xperia Z");
      }

   }

}
```

Output :
S6
HTC one
iPhone6

Refer https://javahungry.blogspot.com/2015/02/how-concurrenthashmap-works-in-java-internal-implementation.ht

ml

Q171)how treemap works internally in java?? How it arranges data in ascending order??

By default, TreeMap sorts all its entries according to their natural ordering.
For an integer, this would mean ascending order and for strings, alphabetical order.

Why NULL is not allowed in TreeMap?
TreeMap sorts elements in natural order and doesn't allow null keys because compareTo() method throws NullPoint erException if compared with null.

TreeMap uses Red-Black Tree algorithm
The main difference is that TreeMap sorts the key in ascending order.
TreeMap is sorted as the ordering of its keys, or by a Comparator provided at map creation time, depending on whic h constructor is used.

TreeMap does not use hashing for storing key unlike the HashMap and LinkedHashMap use hashing for storing the key.

TreeMap uses a data structure called Red-Black tree. Also, all its elements store in the TreeMap are sorted by key.
TreeMap performs sorting in natural order on its key, it also allows you to use Comparator for custom sorting imple mentation.

TreeMap is based on tree data structure as its name suggested. As we know that, in a tree,
each node has three references its parent, right and left element.

We can provide Comparator at map creation time, depending on which constructor is used. Let’s see the following:

TreeMap(): This default constructor constructs an empty TreeMap that will be sorted by using the natural order of it s keys.
TreeMap(Comparator comp): This is an argument constructor and it takes Comparator object to constructs an empty tree-based map.
It will be sorted by using the Comparator comp.
TreeMap(Map map): It creates a TreeMap with the entries from a map, which will be sorted by using the natural ord er of the keys.
TreeMap(SortedMap sortedMap): It also initializes a TreeMap with the entries from sortedMap, which will be sorte d in the same order as sortedMap.

The left element will always be logically less than the parent element.
The right element will always be logically greater than OR equal to a parent element
The logical comparison of Objects is done by natural order i.e. those object who implement Comparable interface an d override
compareTo(Object obj) method. Based on the return value,

How is TreeMap ordered?

A TreeMap is always sorted based on keys.
The sorting order follows the natural ordering of keys.
You may also provide a custom Comparator to the TreeMap at the time of creation to let it sort the keys using the su pplied Comparator.
A TreeMap cannot contain duplicate keys

Q172) What are different types of beans scope in spring??

In Spring framework, we can create beans in 6 inbuilt spring bean scopes and you can also define your custom bean scope as well
singleton and prototype scopes are available in any type of IOC containers.

1) singleton scope:
singleton is default bean scope in spring container.
It tells the container to create and manage only one instance of bean class, per container.
This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached instance.

Example of singleton scope bean using Java config –

```
@Component
//This statement is redundant - singleton is default scope
@Scope("singleton")  //This statement is redundant
public class BeanClass {

}
```

2) prototype scope :prototype scope results in the creation of a new bean instance every time a request for the bean is made by application code.
Java config example of prototype bean scope –

```
@Component
@Scope("prototype")
public class BeanClass {
}
```

XML config example of prototype bean scope –

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="prototype" />
```

3) Request scope: In request scope, container creates a new instance for each and every HTTP request.
So, if server is currently handling 50 requests, then container can have at most 50 individual instances of bean class.

Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the request is completed.

Java config example of request bean scope –

```
@Component
@Scope("request")
public class BeanClass {
}

//or

@Component
@RequestScope
public class BeanClass {
}
```

XML config example of request bean scope –

<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="request" />

4) Session scope
In session scope, container creates a new instance for each and every HTTP session. So, if server has 20 active sessi ons,
then container can have at most 20 individual instances of bean class.
All HTTP requests within single session lifetime will have access to same single bean instance in that session scope .

Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the session is destroyed/end on server.

Java config example of session bean scope –

```
@Component
@Scope("session")
public class BeanClass {
}

//or

@Component
@SessionScope
public class BeanClass {
}
```

XML config example of session bean scope –

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="session" />
```

5)Application scope
In application scope, container creates one instance per web application runtime. It is almost similar to singleton sco pe, with only two differences i.e.

application scoped bean is singleton per ServletContext, whereas singleton scoped bean is singleton per Application Context. Please note that there can be multiple application contexts for single application.
application scoped bean is visible as a ServletContext attribute.
Java config example of application bean scope –

```
@Component
@Scope("application")
public class BeanClass {
}

//or

@Component
@ApplicationScope
public class BeanClass {
}
```

XML config example of application bean scope –

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="application" />
```

6)websocket scope

The WebSocket Protocol enables two-way communication between a client and a remote host that has opted-in to co mmunication with client.
WebSocket Protocol provides a single TCP connection for traffic in both directions.
This is specially useful for multi-user applications with simultaneous editing and multi-user games.

In this type of web applications, HTTP is used only for the initial handshake.
Server can respond with HTTP status 101 (switching protocols) if it agrees – to handshake request.
If the handshake succeeds, the TCP socket remains open and both client and server can use it to send messages to ea ch other.

Java config example of websocket bean scope –


```
@Component
@Scope("websocket")
public class BeanClass {
}
```

XML config example of websocket bean scope –

```
<bean id="beanId" class="com.howtodoinjava.BeanClass" scope="websocket" />
```

Q173) How to find nth highest data in SQL??

use denserank

```
SELECT Studentname,
      Subject,
      Marks,
      DENSE_RANK() OVER(PARTITION BY StudentName ORDER BY Marks ) Rank
FROM ExamResult
ORDER BY Studentname,
       Rank;
```

Using Offset:

```
SELECT DISTINCT(salary) AS salary
FROM tbl_salary
ORDER BY salary DESC
LIMIT 1 OFFSET (n - 1);
```

Q174) what are Runnable interface??

Runnable is an interface that is to be implemented by a class whose instances are intended to be executed by a thread .
There are two ways to start a new Thread – Subclass Thread and implement Runnable

Q175) how to break sinleton class in java??

It can break if the class is Serializable
It can break if its Clonable

Break by Cloning. If a Singleton class implements java.lang.Cloneable interface then invoking clone() method on its single instance creates a duplicate object.

Deserialization also breaks Singleton.
Reflection can instantiate a Singleton multiple times.

Q176)difference between singleton class and singleton bean

Q177) How to catch exception in overridden method ??

It means that if a method declares to throw a given exception,
the overriding method in a subclass can only declare to throw that exception or its subclass. For example:

```
class A {
   public void foo() throws IOException {..}
}

class B extends A {
   @Override
   public void foo() throws SocketException {..} // allowed

   @Override
   public void foo() throws SQLException {..} // NOT allowed
}
```

SocketException extends IOException, but SQLException does not.

This is because of polymorphism:

```
A a = new B();
try {
    a.foo();
} catch (IOException ex) {
    // forced to catch this by the compiler
}
```

If B had decided to throw SQLException, then the compiler could not force you to catch it,
because you are referring to the instance of B by its superclass - A.
On the other hand, any subclass of IOException will be handled by clauses (catch or throws) that handle IOExceptio
n

Q178) what are covarient return type??

Before Java5, it was not possible to override any method by changing the return type.
But now, since Java5, it is possible to override method by changing the return type
if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type

Q179) what is @primary annotation in springboot??
we use @Primary to give higher preference to a bean when there are multiple beans of the same type.
To access beans with the same type we usually use @Qualifier(“beanName”) annotation.
We apply it at the injection point along with @Autowired.
In our case, we select the beans at the configuration phase so @Qualifier can't be applied here

Q180) How to unit test the private method??
Q181) what are zetkins sleuth??
Q182) what are springboot design pattern??
Q183) what all are immutable in java??
In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable

Q184) what is centralized exception handling??
Q185) How to remove duplicate using sql??
Q186)Why do you need String [] args?
Whenever you run a Java program with command prompt or want to give command line arguments,
then “String[] args” is used. So basically it takes input from you via command lines.
If you don't use command line arguments then it has no purpose to your code. Javac is used for compiling your sourc e code.

187) How to check if brackets are balanced??

{]} --unbalanced
{}[]--balanced

Q188) what is n+1 prblm in hibernate?? how to resolve it??
Q189) what all changes will be required when we pass a class with multiple param as a key in hashmap??
Q190) what are push pop and peak method in stack??
The peek() method is used to look at the object at the top of this stack without removing it from the stack.

Q191) Difference between poll() and remove() method?
Both poll() and remove() take out the object from the Queue but if poll() fails then it returns null but if remove fails i t throws Exception.

Q192) Is it possible for two unequal objects to have the same hashcode?
Yes, two unequal objects can have same hashcode that's why collision happen in a hashmap.
the equal hashcode contract only says that two equal objects must have the same hashcode it doesn't say anything ab out the unequal object.

Q193) Can two equal object have the different hash code?
No, thats not possible according to hash code contract.

Q194) What do the expression 1.0 / 0.0 will return? will it throw Exception? any compile-time error?
The simple answer to this question is that it will not throw ArithmeticExcpetion and return Double.INFINITY.

Q195) If a method throws NullPointerException in the superclass, can we override it with a method that throws Run timeException?
The answer is you can very well throw superclass of RuntimeException in overridden method, but you can not do th e same if it's checked Exception.

Q196) What is the difference between Authentication vs Authorization?
Add to PDF Entry
Answer
Authentication is the process of ascertaining that somebody really is who he claims to be.
Authorization refers to rules that determine who is allowed to do what.
E.g. Adam may be authorized to create and delete databases, while Usama is only authorised to read.
Or in short:
Authentication = login + password (who you are)
Authorization = permissions (what you are allowed to do)
Also:
Authentication = Verification
Authorization = Permissions

Q197) How can you catch an exception thrown by another thread in Java?
This can be done using Thread.UncaughtExceptionHandler.

Q198) different ways to create object in java??
Java new Operator.
Java Class. newInstance() method.
Java newInstance() method of constructor.
Java Object. clone() method.
Java Object Serialization and Deserialization.
Using new Keyword

Q199) What's the base class of all exception classes?
In Java, Java.lang.Throwable is the super class of all exception classes and all exception classes are derived from thi s base class.

Q200) What's meant by anonymous class?
An anonymous class is a class defined without any name in a single line of code using new keyword.

Q201) what is hashing collision??

It’s a situation where two or more key objects produce the same final hash value and hence point to the same bucket location or array index.

Q202) How to solve hash collision in Java?
Separate Chaining. In the method known as separate chaining, each bucket is independent and has some sort of list of entries with the same index.
Open addressing. Double hashing.

Q203) Give example of Custom Exception??

```
class InvalidAgeException  extends Exception
{
    public InvalidAgeException (String str)
    {
        // calling the constructor of parent Exception
        super(str);
    }
}

// class that uses custom exception InvalidAgeException
public class TestCustomException1
{

    // method to check the age
    static void validate (int age) throws InvalidAgeException{
      if(age < 18){

        // throw an object of user defined exception
        throw new InvalidAgeException("age is not valid to vote");
    }
      else {
```

```
        System.out.println("welcome to vote");
        }
     }

    // main method
    public static void main(String args[])
    {
        try
        {
            // calling the method
            validate(13);
        }
        catch (InvalidAgeException ex)
        {
            System.out.println("Caught the exception");

            // printing the message from InvalidAgeException object
            System.out.println("Exception occured: " + ex);
        }

        System.out.println("rest of the code...");
    }
}
```

Q204) What are executor framework??

Java provides its own multi-threading framework called the Java Executor Framework.

Thread pools overcome this issue by keeping the threads alive and reusing the threads. Any excess tasks flowing in, that the threads in the pool can't handle are held in a Queue.

Types of Executors:

SingleThreadExecutor: ExecutorService executor = Executors.newSingleThreadExecutor();

A thread pool of single thread can be obtained by calling the static newSingleThreadExecutor() method of the Executors class. It is used to execute tasks sequentially.

FixedThreadPool: ExecutorService fixedPool = Executors.newFixedThreadPool(2);

it is a thread pool of a fixed number of threads. The tasks submitted to the executor are executed by the n threads and if there is more task they are stored on a LinkedBlockingQueue. It uses Blocking Queue

CachedThreadPool: ExecutorService executorService = Executors.newCachedThreadPool();

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available.

ScheduledExecutor:

Scheduled executors are based on the interface ScheduledExecutorService which extends the ExecutorService interface. This executor is used when we have a task that needs to be run at regular intervals or if we wish to delay a certain task.

Q205) what will happen if we remove @repository with @service?

Nothing will happen the code will work as expected

Q206) what is IOC in SpringBoot??

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

to instantiate the application class
to configure the object
to assemble the dependencies between the objects

There are two types of IoC containers. They are:

BeanFactory
ApplicationContext

The org.springframework.beans.factory.BeanFactory and the org.springframework.context.ApplicationContext interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

Q207) what is patch method??
The key idea behind PATCH is to provide a means to apply partial updates to a resource's state.

Q208) Why we need default method in java 8??

Simplest answer is to enable the functionality of lambda expression in java. Lambda expression are essentially of type of functional interface. To support lambda expressions seamlessly, all core classes have to be modified. But these core classes like java.util.List are implemented not only in JDK classes, but also in thousands of client code as well. Any incompatible change in core classes will back fire for sure and will not be accepted at all.

Default methods break this deadlock and allow adding support for functional interface in core classes

Q209) How to make sure method is called once in java?

make use of synchronized keyword in method decleration

```
public class SynchronousCall {

             private boolean methodCalled = false;

          public synchronized void yourMethod() {
              if (!methodCalled) {
                  // Your method logic goes here
                  System.out.println("Method called!");

                  // Set the flag to true to indicate that the method has been called
                  methodCalled = true;
              } else {
                  // Optionally, throw an exception or log a message if the method is called again
                  System.out.println("Method already called. Ignoring additional calls.");
              }
          }

          public static void main(String[] args) {
             SynchronousCall example = new SynchronousCall();

              // Call yourMethod multiple times
              example.yourMethod();
              example.yourMethod();
              example.yourMethod();
          }
      }
```

Q210) how to make hashmap or List immutable?

Map<String, Integer> immutableMap = Collections.unmodifiableMap(new HashMap<>(mutableMap));

Example:

```
public class ImmutableHashmap {
    public static void main(String[] args) {
        // Create a mutable HashMap
        Map<String, Integer> mutableMap = new HashMap<>();
        mutableMap.put("one", 1);
        mutableMap.put("two", 2);
        mutableMap.put("three", 3);

        // Create an immutable view of the HashMap
        Map<String, Integer> immutableMap = Collections.unmodifiableMap(mutableMap);

        // Attempting to modify the immutable map will result in UnsupportedOperationException
        // Uncommenting the line below will cause an exception
         //immutableMap.put("four", 4);

        // Print the original and immutable maps
        System.out.println("Original Map: " + mutableMap);
        System.out.println("Immutable Map: " + immutableMap);
    }
}
```

Q211) will code work if we remove @table annotation from entity file??

The @Table annotation in a JPA (Java Persistence API) entity is used to specify details about the database table to which the entity is mapped. If you remove the @Table annotation, the JPA provider (such as Hibernate) will use default settings and naming conventions to determine the table name. i.e class name will be taken as table name

Q212) how to Pass custom class as key in hashmap??

We need to override hashcode and equals method in class.

Example:

```
import java.util.HashMap;
import java.util.Map;

class MyClass {
    private int id;
    private String name;

    public MyClass(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // Getters, setters, etc.

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        MyClass myClass = (MyClass) o;

        if (id != myClass.id) return false;
        return name != null ? name.equals(myClass.name) : myClass.name == null;
    }

    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (name != null ? name.hashCode() : 0);
        return result;
    }
}
```

```
public class HashMapExample {
    public static void main(String[] args) {
        Map<MyClass, String> myHashMap = new HashMap<>();

        // Adding entries to the HashMap
        myHashMap.put(new MyClass(1, "John"), "Value1");
        myHashMap.put(new MyClass(2, "Jane"), "Value2");
        myHashMap.put(new MyClass(3, "Doe"), "Value3");

        // Retrieving values from the HashMap
        MyClass key = new MyClass(2, "Jane");
        String value = myHashMap.get(key);

        if (value != null) {
            System.out.println("Value for key " + key + ": " + value);
        } else {
            System.out.println("Key " + key + " not found in the HashMap.");
        }
    }
}
```

Output: Value for key MyClass{id=2, name='Jane'}: Value2

Q213) what is the use @builder in lombok?

The @Builder annotation produces complex builder APIs for your classes.

Q214) what are common functional interface present in java8??

1) Consumer

A Consumer is an in-build functional interface in the java.util.function package. we use consumers when we need to consume objects, the consumer takes an input value and returns nothing. The consumer interface has two methods.

```
void accept(T value);
default Consumer<T> andThen(Consumer<? super T> after);
```

```
@Test
public void printCities() {

    List<String> cities = new ArrayList<>();
    cities.add("Delhi");
    cities.add("Mumbai");
```

```
    cities.add("Goa");
    cities.add("Pune");

    Consumer<String> printConsumer= city-> System.out.println(city);
    cities.forEach(printConsumer);
}
```

2) A Predicate is a functional interface, which accepts an argument and returns a boolean. Usually, it is used to apply in a filter for a collection of objects.

```
@Test
public void filterCities() {

   List<String> cities = new ArrayList<>();
   cities.add("Delhi");
   cities.add("Mumbai");
   cities.add("Goa");
   cities.add("Pune");

   Predicate<String> filterCity = city -> city.equals("Mumbai");
   cities.stream().filter(filterCity).forEach(System.out::println);
}
```

3) A Function is another in-build functional interface in java.util.function package, the function takes an input value and returns a value

```
@Test
public void mapCities() {

    List<String> cities = new ArrayList<>();
    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Goa");
    cities.add("Pune");

    Function<String, Character> getFirstCharFunction = city -> {
        return city.charAt(0);
    };
    cities.stream().map(getFirstCharFunction)
                .forEach(System.out::println);
}
```

4) The Supplier Interface is a part of the java.util.function package. It represents a function that does not take in any argument but produces a value of type T. It contains only one method.

```
@Test
public void supplyCities() {

    Supplier<String[]> citySupplier = () -> {
        return new String[]{"Mumbai", "Delhi", "Goa", "Pune"};
    };
    Arrays.asList(citySupplier.get()).forEach(System.out::println);
}
```

Q215) Design patterns in springboot??

Spring Boot, being a popular framework for building Java-based microservices and web applications, encourages the use of various design patterns. Below are some common design patterns used in Spring Boot applications:

1) Singleton Pattern:

Spring Boot by default follows the Singleton pattern, where a single instance of a class is created and shared across the application. This is especially true for Spring-managed beans.

2) Factory Pattern:

Spring Boot uses the Factory pattern extensively, particularly with the Spring Container acting as a factory for creating and managing beans.

3) Dependency Injection (DI):

The Dependency Injection pattern is fundamental to the Spring framework. In Spring Boot, dependencies are injected into components, services, and controllers, promoting loose coupling and making components easily testable.

4) Template Method Pattern:

Spring Boot often uses the Template Method pattern in its data access operations. For example, the JdbcTemplate class provides a set of template methods for executing SQL queries.

5) Strategy Pattern:

In Spring Boot, the Strategy pattern is used in various places, such as the implementation of different @Profiles for different environments or configurations.

6) Builder Pattern:

The Builder pattern is commonly used in Spring Boot for building complex objects. For example, the ResponseEntity class provides a builder pattern for creating HTTP responses.

7) Proxy Pattern:

Spring AOP (Aspect-Oriented Programming) uses the Proxy pattern to add behavior to methods, such as logging, transaction management, etc.

8)Decorator Pattern:

Spring Boot's @Configuration annotation and JavaConfig feature follow the Decorator pattern, allowing you to enhance or modify the behavior of beans.

9) Command Pattern:

Spring's JdbcTemplate uses the Command pattern to encapsulate a request as an object, allowing parameterization of clients with queues, requests, and operations.

10) Chain of Responsibility Pattern:

Spring Boot's filter chain (e.g., Filter and Interceptor) follows the Chain of Responsibility pattern, where each filter can process the request and decide whether to pass it to the next filter.

11) Builder Pattern:

The Builder pattern is used in Spring Boot's @Bean annotation and @Configuration classes to construct complex objects and configurations.

12) Observer Pattern:

Spring's event handling mechanism is based on the Observer pattern. Components can act as observers, listening for events and responding to them.

Q216) what are optionals in java?? Give example?

Optional is a container object used to contain not-null objects.
This class has various utility methods to facilitate code to handle values as 'available' or 'not available'
instead of checking null values. It is introduced in Java 8 and is similar to what Optional is in Guava

```
import java.util.Optional;
public class OptionalExample {
    public static void main(String[] args) {
        String[] str = new String[10];
        Optional<String> checkNull = Optional.ofNullable(str[5]);
        if(checkNull.isPresent()){  // check for value is present or not
```

```
            String lowercaseString = str[5].toLowerCase();
            System.out.print(lowercaseString);
        }else
            System.out.println("string value is not present");
    }
}
```

Q217) how we implement Spring security in springboot?

Spring Security is a powerful and customizable authentication and access control framework for Java applications, particularly in the context of Spring-based applications.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
```

Now, you might want to customize the security configuration. You can create a SecurityConfig class like this:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .antMatchers("/user/**").hasRole("USER")
                .antMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
```

```
        return new BCryptPasswordEncoder();
    }
```

Q218) how we communicate between microservices using feignclient. Give example?

@FeignClient(value = "http://assortmentoptimization",fallback=AssortmentOptimizationAPIImpl.class)

Main class:

```
@EnableFeignClients

@FeignClient(name = "example-service", fallback = FeignClientFallback.class)
public interface MyFeignClientInterface {


       @GetMapping("/api/hello")
        String getHello();
       }

public class FeignClientFallback implements MyFeignClientInterface {

       @Override
    public String getHello() {
        return "Fallback Hello";
    }
}
```

Controller class:

```
  @Autowired private MyFeignClientInterface feignClient;

  @GetMapping("/hello") public String getHello() { return
  feignClient.getHello(); }
```

Q219) Stream flatMap() and stream map example??

The flatMap() operation is used when each element in the stream is transformed into multiple elements, often in the form of another collection or stream. the flatMap() operation is a two-step process i.e. map() + flattening. In a broader sense, it helps convert Collection<Collection<Item>> to Collection<Item>.

```
List<List<Integer>> listOfLists = Arrays.asList(
  Arrays.asList(1, 2, 3),
  Arrays.asList(4, 5),
  Arrays.asList(6, 7, 8)
);

List<Integer> flattenedList = listOfLists.stream()
```

```
        .flatMap(list -> list.stream())
        .toList();

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

The map() operation is used to transform each element of a stream into another object using a given function. It returns a new stream containing the transformed elements in the same order as the original stream.

```
List<String> listOfStrings = Arrays.asList("1", "2", "3", "4", "5");

List<Integer> listOfIntegers = listOfStrings.stream()
                        .map(Integer::valueOf)
                        .toList();

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

Q220) what is @Procedure used in springboot??

A stored procedure is a precompiled collection of one or more SQL statements or procedural logic, which is stored in a relational database management system (RDBMS). Stored procedures are typically written in a procedural language specific to the database system, such as PL/SQL (Oracle)

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    @Procedure(name = "getEmployeesByDepartment")
    List<Employee> getEmployeesByDepartment(@Param("department") String department);
}
```

Q221) Object class in java??

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know

```
Object obj=getObject();
```

Q222) Difference between Wait and Sleep in Java??

wait and **sleep** are the concept of multithreading. Wait and Sleep are the methods used to pause a process

for few seconds and go a thread in the waiting state

Sleep()

The Sleep () method is related to the Thread class that is used to stop the execution of the current Thread for few seconds. The Sleep () method takes the sleeping time in milliseconds

Wait()

The Wait() method is related to the Object class. The Wait() method is responsible for sending the calling thread into the waiting state. The Thread remains in the waiting state until another thread doesn't invoke the notify() or notifyAll() method for that object. The Thread resumes the execution after obtaining the ownership of the monitor.

```
class WaitSleepSimilaritiesExample
{

    //create an instance of the Object
    private static Object obj = new Object();

    //main() method starts with handling InterruptedException
    public static void main(String[] args)throws InterruptedException
    {

        //pause process for two second
        Thread.sleep(2000);

        //print custom statement
        System.out.println( Thread.currentThread().getName() +
        " Thread is woken after two second");
```

```
        //create synchronizec context from which we call Wait() method
        synchronized (obj)
        {
            //use wailt() method to set obj in waiting state for two seconds
            obj.wait(2000);

            System.out.println(obj + " Object is in waiting state and woken after 2 seconds");
        }
    }
}
```

Q223) Write code to find occurance of each character in string using Java 8??

```
 public class CountCharacterOccurrence {

    public static void main(String[] args) {
        String input = "gainjavaknowledge";

        Map<String, Long> output = Arrays.stream(input.split("")).collect(Collectors.groupingBy(Function.identity(),
 Collectors.counting()));
        System.out.println("Output : "+output);

    }
}
```

Q224) Find second largest number without using sort??

```
 public class Del {

 public static void main(String[] args) {

                int temp, size;

            int[] array = {30, 20, 25, 63, 96, 57};

            size = array.length;


            for(int i = 0; i<size; i++ ){

              for(int j = i+1; j<size; j++){


                if(array[i]>array[j]){

                  temp = array[i]; //30

                  array[i] = array[j]; // 20,20...

                  array[j] = temp; //30

                }

              }

            }


            System.out.println("second largest number is:: "+array[array.length-2]);


            System.out.println("second min number is:: "+array[1]);

        }

}
```

Q225) Find Duplicate character in string??

```
 public class DuplicateCharsInString {
    public static void main(String[] args) {
        String input = "pranavvermaz";

        // Using Java 8 streams and collectors to find duplicate characters
        Map<Character, Long> charCountMap = input.chars()
                .mapToObj(c -> (char) c)
                .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

        // Displaying duplicate characters
        System.out.println("Duplicate characters in the string:");

        charCountMap.entrySet().stream()
                .filter(entry -> entry.getValue() > 1)
                .forEach(entry -> System.out.println(entry.getKey() + ": " + entry.getValue()));

 charCountMap.entrySet().stream()
        .filter(entry -> entry.getValue() ==1)
        .forEach(entry -> System.out.println(entry.getKey() + ": " + entry.getValue()));
    }
}
```

Q226) Find 1st occurring character in string using java 8?? { asked in EY}

Q227) Find 1st Non occurring character in string??

Q228) Find which character has appeared most in string using java 8??

Q229) How to sort number in descending order??

```
 List<Integer> distinctSortedNumbers = numbers.stream().distinct()  // Remove duplicates
 .sorted(Collections.reverseOrder())  // Sort in descending order
 .collect(Collectors.toList());
System.out.println("highest" + highest);
```

Note: stream.sorted is used to sort in ascending order

Q230) how to read textfile in java??

```
public class ReadTextFileExample {
    public static void main(String[] args) {
        // Specify the path to the text file
        String filePathString = "D:/textfile.txt";
        Path filePath = Paths.get(filePathString);

        try {
            // Read all lines from the file into a List of Strings
            List<String> lines = Files.readAllLines(filePath, StandardCharsets.UTF_8);

            // Print each line
            for (String line : lines) {
                System.out.println(line);
            }
```

```
        } catch (IOException e) {
            // Handle IOException (file not found, permission issues, etc.)
            e.printStackTrace();
        }
    }
}
```