```
*****************************
*****************************
Core Java :
*****************************
*****************************
==============================
Q) OOPS Concepts
A) Abstraction
Polymorphism (Method over loading & method overriding)
Inheritence
Encapsulation
==============================
Q) String handling
A)      String s1="hello";
        String s2=new String("Hello");
        String s3=s1;
        String s4=s2;
        System.out.println(s1==s2); // Comparing references. so
FALSE
        System.out.println(s1.equalsIgnoreCase(s2)); // Comparing
values. so TRUE
        System.out.println(s1==s3); // s1 directly assigned to s3 so
both references are same. so TRUE
        System.out.println(s2==s4);// s4 directly assigned to s2 so
both references are same. so TRUE
        System.out.println(s1.equalsIgnoreCase(s3)); // s1 directly
assigned to s3 so both values are same. so TRUE
        System.out.println(s2.equalsIgnoreCase(s4)); // s2 directly
assigned to s4 so both values are same. so TRUE
==============================
Q) String& StringBuffer & StringBuilder & StringTockenizer
A) The String class is an immutable class whereas StringBuffer and
StringBuilder classes are mutable. ... StringBuffer is synchronized
i.e. thread safe. It means two threads can't call the methods of
StringBuffer simultaneously. StringBuilder is non-synchronized i.e.
not thread safe.

The java.util.StringTokenizer class allows you to break a string
into tokens. It is simple way to break string. The string tokenizer
class allows an application to break a string into tokens. The
tokenization method is much simpler than the one used by the
StreamTokenizer class. The StringTokenizer methods do not
distinguish among identifiers, numbers, and quoted strings, nor do
they recognize and skip comments.

 i] String is immutable whereas StringBuffer and StringBuilder are
mutable classes.
ii] StringBuffer is thread-safe and synchronized whereas
StringBuilder is not. That’s why StringBuilder is faster than
StringBuffer.
iii] String concatenation operator (+) internally uses StringBuffer
```

or StringBuilder class.
iV] For String manipulations in a non-multi threaded environment, we should use StringBuilder else use StringBuffer class.
==========
Q) Why strings are immutable in java? How can we develop our own immutable classes?
A) In the String constant pool, a String object is likely to have one or many references. If several references point to same String without even knowing it, it would be bad if one of the references modified that String value. That's why String objects are immutable.

i] Declare the class as final so it can't be extended.
ii] Make all fields private so that direct access is not allowed.
iii] Don't provide setter methods for variables.
iv] Make all mutable fields final so that its value can be assigned only once.
v] Initialize all the fields via a constructor performing deep copy.
vi] Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.
(Ref : https://www.journaldev.com/129/how-to-create-immutable-class-in-java)
==========
Q) Polymorphism Concept
A) Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
There are two types of polymorphism in Java: compile-time polymorphism or static polymorphism (Method overloading) and runtime polymorphism or dynamic polymorphish(Method Overriding). We can perform polymorphism in java by method overloading and method overriding.
If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.
(Ref : https://www.geeksforgeeks.org/polymorphism-in-java/)
==========
Q) Collection Hierarchy
Iterable => Collections
List -> ArrayList, LinkedList, DoubleLinkedList, Vector, Stack...
set -> SortedSet, TreeSet, HashSet, LinkedHashSet..
Queue -> PriorotiQueue, Deuqe, ArrayDeque...
++ Methods in one dimension array collections
* public boolean add(E e) - It is used to insert an element in this collection.
* public boolean addAll(Collection<? extends E> c) - It is used to insert the specified collection elements in the invoking collection.
* public boolean remove(Object element) - It is used to delete an element from the collection.
* public boolean removeAll(Collection<?> c) - It is used to delete

all the elements of the specified collection from the invoking collection.
* default boolean removeIf(Predicate<? super E> filter) - It is used to delete all the elements of the collection that satisfy the specified predicate.
* public boolean retainAll(Collection<?> c) - It is used to delete all the elements of invoking collection except the specified collection.
* public int size() - It returns the total number of elements in the collection.
* public void clear() - It removes the total number of elements from the collection.
* public boolean contains(Object element) - It is used to search an element.
* public boolean containsAll(Collection<?> c) - It is used to search the specified collection in the collection.
* public Iterator iterator() - It returns an iterator.
* public Object[] toArray() - It converts collection into array.
* public <T> T[] toArray(T[] a) - It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
* public boolean isEmpty() - It checks if collection is empty.
* default Stream<E> parallelStream() - It returns a possibly parallel Stream with the collection as its source.
* default Stream<E> stream() - It returns a sequential Stream with the collection as its source.
* default Spliterator<E> spliterator() - It generates a Spliterator over the specified elements in the collection.
* public boolean equals(Object element) - It matches two collections.
* public int hashCode() - It returns the hash code number of the collection.

++ Two ways to iterate collection - Iterator & ListIterator & Enumeration

++ Iterator Methods:
* public boolean hasNext() - It returns true if the iterator has more elements otherwise it returns false.
* public Object next() - It returns the element and moves the cursor pointer to the next element.
* public void remove() - It removes the last elements returned by the iterator. It is less used.

++ Methods of ListIterator
The ListIterator interface provides methods that can be used to perform various operations on the elements of a list.
* hasNext() - returns true if there exists an element in the list
* next() - returns the next element of the list
* nextIndex() returns the index of the element that the next() method will return

* previous() - returns the previous element of the list
* previousIndex() - returns the index of the element that the previous() method will return
* remove() - removes the element returned by either next() or previous()
* set() - replaces the element returned by either next() or previous() with the specified element

++ Enumeration Methods

* boolean hasMoreElements( ) - When implemented, it must return true while there are still more elements to extract, and false when all the elements have been enumerated.
* Object nextElement( ) - This returns the next object in the enumeration as a generic Object reference.
(Ref : https://www.tutorialspoint.com/java/java_enumeration_interface.htm)

++ iterator vs enumeration in java

* Basic - In Iterator, we can read and remove element while traversing element in the collections. :: Using Enumeration, we can only read element during traversing element in the collections.
* Access - It can be used with any class of the collection framework. :: It can be used only with legacy class of the collection framework such as a Vector and HashTable.
* Fail-Fast and Fail -Safe - Any changes in the collection, such as removing element from the collection during a thread is iterating collection then it throw concurrent modification exception. :: Enumeration is Fail safe in nature. It doesn’t throw concurrent modification exception
* Limitation - Only forward direction iterating is possible :: Remove operations can not be performed using Enumeration.
* Methods - Iterator has following methods – hasNext() & next() & remove() :: Enumeration has following methods – hasMoreElements() & nextElement()

++ Iterator vs ListIterator

The Iterator and ListIterator are the two among the three cursors of Java. Both Iterator and ListIterator are defined by Collection Framework in Java.Util package. ListIterator is the child interface of Iterator interface.
The major difference between Iterator and ListIterator is that Iterator can traverse the elements in the collection only in forward direction whereas, the ListIterator can traverse the elements in a collection in both the forward as well as the backwards direction.

++ Iterable Interface
* The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable

interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e., - Iterator<T> iterator() - It returns the iterator over the elements of type T.
(Ref : https://www.javatpoint.com/collections-in-java)

Map => SortedMap -> TreeMap, HashMap -> LinkedHashMap
++ Methods of two dimension collection
* V put(Object key, Object value) - It is used to insert an entry in the map.
* void putAll(Map map) - It is used to insert the specified map in the map.
* V putIfAbsent(K key, V value) - It inserts the specified value with the specified key in the map only if it is not already specified.
* V remove(Object key) - It is used to delete an entry for the specified key.
* boolean remove(Object key, Object value) - It removes the specified values with the associated specified keys from the map.
* Set keySet() - It returns the Set view containing all the keys.
* Set<Map.Entry<K,V>> entrySet() - It returns the Set view containing all the keys and values.
* void clear() - It is used to reset the map.
* V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) - It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
* V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction) - It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
* V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) - It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
* boolean containsValue(Object value) - This method returns true if some value equal to the value exists within the map, else return false.
* boolean containsKey(Object key) - This method returns true if some key equal to the key exists within the map, else return false.
* boolean equals(Object o) - It is used to compare the specified Object with the Map.
* void forEach(BiConsumer<? super K,? super V> action) - It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
* V get(Object key) - This method returns the object that contains the value associated with the key.
* V getOrDefault(Object key, V defaultValue) - It returns the value

to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
* int hashCode() - It returns the hash code value for the Map
* boolean isEmpty() - This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction) - If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
* V replace(K key, V value) - It replaces the specified value for a specified key.
* boolean replace(K key, V oldValue, V newValue) - It replaces the old value with the new value for a specified key.
* void replaceAll(BiFunction<? super K,? super V,? extends V> function) - It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
* Collection values() - It returns a collection view of the values contained in the map.
* int size() - This method returns the number of entries in the map.

++ Map.Entry Interface
Entry is the subinterface of Map. So we will be accessed it by Map.Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

++ Methods of Map.Entry Interface

* K getKey() - It is used to obtain a key.
* V getValue() - It is used to obtain value.
* int hashCode() - It is used to obtain hashCode.
* V setValue(V value) - It is used to replace the value corresponding to this entry with the specified value.
* boolean equals(Object o) - It is used to compare the specified object with the other existing objects.
* static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> comparingByKey() - It returns a comparator that compare the objects in natural order on key.
* static <K,V> Comparator<Map.Entry<K,V>> comparingByKey(Comparator<? super K> cmp) - It returns a comparator that compare the objects by key using the given Comparator.
* static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> comparingByValue() - It returns a comparator that compare the objects in natural order on value.
* static <K,V> Comparator<Map.Entry<K,V>> comparingByValue(Comparator<? super V> cmp) - It returns a comparator that compare the objects by value using the given Comparator.
(Ref : https://www.javatpoint.com/java-map)

++ How ConcurrentHashMap Works Internally in Java
ConcurrentHashMap: It allows concurrent access to the map. Part of the map called Segment (internal data structure) is only getting locked while adding or updating the map. So ConcurrentHashMap allows concurrent threads to read the value without locking at all. This data structure was introduced to improve performance.
The ConcurrentHashMap is very similar to the HashMap class, except that ConcurrentHashMap offers internally maintained concurrency. It means you do not need to have synchronized blocks when accessing ConcurrentHashMap in multithreaded application.
(Ref :
https://howtodoinjava.com/java/multi-threading/best-practices-for-using-concurrenthashmap/)

Java 8 Related Collections Questions:

Q) How to filter list values :
A) List<Integer> list=Arrays.asList(1,2,3,4,5,6,7,8,9,10);
list=list.stream().filter(ll -> ll >5).collect(Collectors.toList());
// FIlter the values of greater than 5 and add those to same list object.

Q) How to filter Map collection:
A) Map<Integer,String> map=new HashMap<Integer, String>();
map.put(1, "Chair");
map.put(2, "Table");
map.put(3, "TV");
map.put(4, "Sofa");
map=map.entrySet().stream().filter(l -> l.getKey() > 3).collect(Collectors.toMap(l -> l.getKey(), l -> l.getValue())); // FIlter the key of greater than 3 and add those to same map object.

Q) How to sort List of Object:
A) List<Employee> emp=new ArrayList<Employee>();
emp.add(new Employee(1,"Bhanu",9652147892));
emp.add(new Employee(2,"Rukku",7854219630));
emp.add(new Employee(3,"Bolt",9652147892));
emp.add(new Employee(4,"Potti",7854219630));
emp.sort((Employee e1, Employee e2) -> (e1.getName()).compareTo(e2.getName()));

Q) How to convert List to Map?
A) List<Employee> emp=new ArrayList<Employee>();
emp.add(new Employee(1,"Bhanu",9652147892));
emp.add(new Employee(2,"Rukku",7854219630));
Map<String,String> m=emp.stream().collect(Collectors.toMap(Employee::getId, Employee::getName));

Q) Hashing Cillision

A) Underlying working of all these Maps(HashMap, LinkedHashMap, Hashtable, WeakHashMap, IdentityHashMap, ConcurrentHashMap, TreeMap, EnumMap.) is pretty much the same as discussed in How does HashMap internally works in Java(Ref : https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#axzz6rzqC0Ql3), except some minor differences in their specific behaviors. Since the hash table data structure is subject to collision all these implementations are required to handle the collision. A collision occurs when a hash function returns the same bucket location for two different keys. Since all hash-based Map class e.g. HashMap uses equals() and hashCode() (Ref : https://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html#axzz6rzqC0Ql3) contract to find the bucket. HashMap calls the hashCode() method to compute the hash value which is used to find the bucket location as shown in the below code snippet from the HashMap class.
(Ref : https://javarevisited.blogspot.com/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html#axzz6rzqC0Ql3)

Q) How collections sort data internal process
A) Sorting data means arranging it in a certain order, often in an array-like data structure. You can use various ordering criteria, common ones being sorting numbers from least to greatest or vice-versa, or sorting strings lexicographically. You can even define your own criteria, and we'll go into practical ways of doing that by the end of this article. If you're interested in how sorting works, we'll cover various algorithms, from inefficient but intuitive solutions, to efficient algorithms which are actually implemented in Java and other languages. There are various sorting algorithms, and they're not all equally efficient. We'll be analyzing their time complexity in order to compare them and see which ones perform the best.
The list of algorithms you'll learn here is by no means exhaustive, but we have compiled some of the most common and most efficient ones to help you get started:
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Heapsort
Quicksort
Sorting in Java
(Ref : https://stackabuse.com/sorting-algorithms-in-java/)
==========
Q) Threading concept
A) MULTITHREADING in Java is a process of executing two or more threads simultaneously to maximum utilization of CPU. Multithreaded applications execute two or more threads run concurrently. Hence, it is also known as Concurrency in Java. Each thread runs parallel to each other.

Q) Thread Life cycle
A)
1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

(Ref : https://www.javatpoint.com/life-cycle-of-a-thread)

Q) Methods in thread class
A) String getName() - Retrieves the name of running thread in the current context in String format
void start() - This method will start a new thread of execution by calling run() method of Thread/runnable object.
void run() - This method is the entry point of the thread. Execution of thread starts from this method.
void sleep(int sleeptime) - This method suspend the thread for mentioned time duration in argument (sleeptime in ms)
void yield() - By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
void join() - This method used to queue up a thread in execution. Once called on thread, current thread will wait till calling thread completes its execution
boolean isAlive() - This method will check if thread is alive or dead
(Ref : https://www.w3resource.com/java-tutorial/java-threadclass-methods-and-threadstates.php)

Q) Thread vs Runnable intrface
1. Basic :: Thread is a class. It is used to create a thread - Runnable is a functional interface which is used to create a thread
2. Methods :: It has multiple methods including start() and run() - It has only abstract method run()
3. Each thread creates a unique object and gets associated with it - Multiple threads share the same objects.
4. Memory :: More memory required - Less memory required
5. Limitation :: Multiple Inheritance is not allowed in java hence after a class extends Thread class, it can not extend any other class - If a class is implementing the runnable interface then your class can extend another class.

(Ref : https://www.tutorialspoint.com/difference-between-thread-and-runnable-in-java)

Q) Wait()-notify()-notifyAll() ::
wait::
Object wait methods has three variance, one which waits indefinitely for any other thread to call notify or notifyAll method on the object to wake up the current thread. Other two variances puts the

current thread in wait for specific amount of time before they wake up.
notify::
notify method wakes up only one thread waiting on the object and that thread starts execution. So if there are multiple threads waiting for an object, this method will wake up only one of them. The choice of the thread to wake depends on the OS implementation of thread management.
notifyAll::
notifyAll method wakes up all the threads waiting on the object, although which one will process first depends on the OS implementation.
(Ref : https://www.journaldev.com/1037/java-thread-wait-notify-and-notifyall-example)

Q) Sleep()
A) Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds. The argument value for milliseconds can’t be negative, else it throws IllegalArgumentException.
There is another overloaded method sleep(long millis, int nanos) that can be used to pause the execution of current thread for specified milliseconds and nanoseconds. The allowed nano second value is between 0 and 999999.
(Ref : https://www.journaldev.com/1020/thread-sleep-java)

Q) Wait (Vs) Sleep
A) Wait() method belongs to Object class. - Sleep() method belongs to Thread class.
Wait() method releases lock during Synchronization. -Sleep() method does not release the lock on object during Synchronization.
Wait() should be called only from Synchronized context. - There is no need to call sleep() from Synchronized context.
Wait() is not a static method. -Sleep() is a static method.
Sleep() Has Two Overloaded Methods:
sleep(long millis)millis: milliseconds
sleep(long millis,int nanos) nanos: Nanoseconds
Wait() Has Three Overloaded Methods:
wait()
wait(long timeout)
wait(long timeout, int nanos)
public final void wait(long timeout)   - public static void sleep(long millis) throws Interrupted_Execption
(Ref : https://www.geeksforgeeks.org/difference-between-wait-and-sleep-in-java/)

Q) Join()
A) java.lang.Thread class provides the join() method which allows one thread to wait until another thread completes its execution. If

t is a Thread object whose thread is currently executing, then t.join() will make sure that t is terminated before the next instruction is executed by the program.
If there are multiple threads calling the join() methods that means overloading on join allows the programmer to specify a waiting period. However, as with sleep, join is dependent on the OS for timing, so you should not assume that join will wait exactly as long as you specify.
There are three overloaded join functions.
join(): It will put the current thread on wait until the thread on which it is called is dead. If thread is interrupted then it will throw InterruptedException.
Syntax:
public final void join()
join(long millis) :It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds).
Syntax:
public final synchronized void join(long millis)
join(long millis, int nanos): It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds + nanos).
Syntax:
public final synchronized void join(long millis, int nanos)
(Ref : https://www.geeksforgeeks.org/joining-threads-in-java/)

Q) Dead lock?
A) Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order. A Java multithreaded program may suffer from the deadlock condition because the synchronized keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object. Here is an example.
(Ref : https://www.tutorialspoint.com/java/java_thread_deadlock.htm)

Q) How can overcome Deadlock?
A) Avoid Nested Locks: A deadlock mainly happens when we give locks to multiple threads. Avoid giving a lock to multiple threads if we already have given to one.
Avoid Unnecessary Locks: We can have a lock only those members which are required. Having a lock unnecessarily can lead to a deadlock.
Using Thread.join(): A deadlock condition appears when one thread is waiting other to finish. If this condition occurs we can use Thread.join() with the maximum time the execution will take.
(Ref :
https://www.tutorialspoint.com/how-can-we-avoid-a-deadlock-in-java)

Q) Executor Framework
A) The Executor Framework contains a bunch of components that are used to efficiently manage worker threads. The Executor API

de-couples the execution of task from the actual task to be executed via Executors. This design is one of the implementations of the Producer-Consumer pattern.
The java.util.concurrent.Executors provide factory methods which are be used to create ThreadPools of worker threads.
To use the Executor Framework we need to create one such thread pool and submit the task to it for execution. It is the job of the Executor Framework to schedule and execute the submitted tasks and return the results from the thread pool.
A basic question that comes to mind is why do we need such thread pools when we can create objects of java.lang.Thread or implement Runnable/Callable interfaces to achieve parallelism?
The answer comes down to two basic facts:
Creating a new thread for a new task leads to overhead of thread creation and tear-down. Managing this thread life-cycle significantly adds to the execution time.
Adding a new thread for each process without any throttling leads to the creation of a large number of threads. These threads occupy memory and cause wastage of resources. The CPU starts to spend too much time switching contexts when each thread is swapped out and another thread comes in for execution.
All these factors reduce the throughput of the system. Thread pools overcome this issue by keeping the threads alive and reusing the threads. Any excess tasks flowing in than the threads in the pool can handle are held in a Queue. Once any of threads get free, they pick up the next task from this queue. This task queue is essentially unbounded for the out-of-box executors provided by the JDK.
(Ref : https://stackabuse.com/concurrency-in-java-the-executor-framework/#:~:text=The%20Executor%20Framework%20contains%20a,The%20java.)

Q) Synchronization concept
A) Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.
Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.
(Ref : https://www.geeksforgeeks.org/synchronized-in-java/)
==========
Q)Exception handling
A) An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the

normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.
An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
A user has entered an invalid data.
A file that needs to be opened cannot be found.
A network connection has been lost in the middle of communications or the JVM has run out of memory.
Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.
Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.
Checked exceptions – A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.
For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception.
Unchecked exceptions – An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptionexception occurs.
(Ref : https://www.tutorialspoint.com/java/java_exceptions.htm)

Q)Resource Management
Q) Java provides a feature to make the code more robust and to cut down the lines of code. This feature is known as Automatic Resource Management(ARM) using try-with-resources from Java 7 onwards. The try-with-resources statement is a try statement that declares one or more resources.
This statement ensures that each resource is closed at the end of the statement which eases working with external resources that need to be disposed or closed in case of errors or successful completion of a code block.
What is a resource?
A resource is an object that must be closed after the program is finished using it. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.
(Ref : https://www.geeksforgeeks.org/automatic-resource-management-java/)
==========

Q) Memory Leak
A) The standard definition of a memory leak is a scenario that occurs when objects are no longer being used by the application, but the Garbage Collector is unable to remove them from working memory – because they're still being referenced. As a result, the application consumes more and more resources – which eventually leads to a fatal OutOfMemoryError.
(Ref : https://stackify.com/memory-leaks-java/#:~:text=What%20is%20a%20Memory%20Leak,they're%20still%20being%20referenced.)
==========
Q) Class Loader Concept
A) The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. The Java run time system does not need to know about files and file systems because of classloaders. Java classes aren't loaded into memory all at once, but when required by an application. At this point, the Java ClassLoader is called by the JRE and these ClassLoaders load classes into memory dynamically.
(Ref : https://www.geeksforgeeks.org/classloader-in-java/)
==========
9. Serilization & Deserilization concept
A) Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.
serialize-deserialize-java
The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.
To make a Java object serializable we implement the java.io.Serializable interface.
The ObjectOutputStream class contains writeObject() method for serializing an Object.

```
public final void writeObject(Object obj)
                        throws IOException
```

The ObjectInputStream class contains readObject() method for deserializing an object.

```
public final Object readObject()
                  throws IOException,
               ClassNotFoundException
```

Advantages of Serialization
1. To save/persist state of an object.
2. To travel an object across a network.

Only the objects of those classes can be serialized which are implementing java.io.Serializable interface.
Serializable is a marker interface (has no data member and method).

It is used to “mark” java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are:- Cloneable and Remote.

Points to remember
1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process.So, if you don’t want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

(Ref : https://www.geeksforgeeks.org/serialization-in-java/)
==========
Q) Singleton class implimentation
A) Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.
 -- Implementation
We're going to create a SingleObject class. SingleObject class have its constructor as private and have a static instance of itself. SingleObject class provides a static method to get its static instance to outside world. SingletonPatternDemo, our demo class will use SingleObject class to get a SingleObject object.
(Ref : https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm)
How to prevent Singleton Pattern from Reflection, Serialization and Cloning? :: (Ref : https://www.geeksforgeeks.org/prevent-singleton-pattern-reflection-serialization-cloning/)
==========
Q) Java Data Structures
A) The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as we have already seen one of the data structures, i.e., array in C language. Array is a collection of memory elements in which data is stored sequentially, i.e., one after another. In other words, we can say that array stores the elements in a continuous manner. This organization of data is done with the help of an array of data structures. There are also other ways to organize the data in memory. Let's see the different types of data structures.
The data structure is not any programming language like C, C++, java, etc. It is a set of algorithms that we can use in any

programming language to structure the data in the memory.
To structure the data in memory, 'n' number of algorithms were proposed, and all these algorithms are known as Abstract data types. These abstract data types are the set of rules.
(Ref : https://www.javatpoint.com/data-structure-tutorial#:~:text=The%20data%20structure%20is%20not,known%20as%20Abstract%20data%20types.)
==========
Q) Sorting employee object using Comparable & Comparator by Collections?
A) (Ref : https://howtodoinjava.com/java/sort/sort-arraylist-objects-comparable-comparator/)
==========
Q) Sorting algorithms?
A) (Ref : https://stackabuse.com/sorting-algorithms-in-java/)
==========
Q) Core programs (Letter frequency in a String & Word frequency in a sentence & remove duplicates from an array & sort elements in an array & n'th highest integer from an array)
A) (Ref : https://www.journaldev.com/370/java-programming-interview-questions)
==========
Q) Ways to create an object?
A) (Ref : https://www.tutorialspoint.com/5-different-ways-to-create-objects-in-Java)

Q) Methods in Object class?
A) public final Class getClass() - returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode() -returns the hashcode number for this object.
public boolean equals(Object obj) compares the given object to this object.
protected Object clone() throws CloneNotSupportedException - creates and returns the exact copy (clone) of this object.
public String toString() - returns the string representation of this object.
public final void notify() -wakes up single thread, waiting on this object's monitor.
public final void notifyAll() - wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException - causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException - causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread

notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException - causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable - is invoked by the garbage collector before object is being garbage collected.
(Ref : https://www.javatpoint.com/object-class)
==========
Q) Java 8 new features -
https://www.journaldev.com/2389/java-8-features-with-examples
==========
Q) Difference between Abstract Class and Interface?
A)
1) Abstract class can have abstract and non-abstract methods. - Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance. - Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables. - Interface has only static and final variables.
4) Abstract class can provide the implementation of interface. - Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class. - The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces. - An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends". - An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc. - Members of a Java interface are public by default.
9)Example Abstract Class:
public abstract class Shape{
public abstract void draw();
}

Example Interface :
public interface Drawable{
void draw();
}
(Ref :
https://www.javatpoint.com/difference-between-abstract-class-and-interface)
==========
Q) How to iterate HshMap -
https://www.geeksforgeeks.org/iterate-map-java/
==========
Q) Asked Scenario based question on Template Design Pattern (there is one interface hving 4 methods. you have to implement in a way such that all methods should execute one after another.
A) (Ref :

https://www.tutorialspoint.com/design_pattern/template_pattern.htm)
==========
Q) Asked Scenario based question on Strategic Design Pattern
A) (Ref :
https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm)
==========
Q) What are the design patterns you have worked on?
A) You can say Template Design Pattern & Strategic Design Pattern. Then if they are asking anything about these, then you can proceed with above references.
==========
Q) Roles and Responsibilities of all the projects?
A) Clear cut explaination of your roles and responsibilities in your current project.
==========
Q) How to achieve thread safety for a class?
A) There are four ways to achieve Thread Safety in Java. These are:
Using Synchronization.
Using Volatile Keyword.
Using Atomic Variable.
Using Final Keyword.
(Ref :
https://www.geeksforgeeks.org/thread-safety-and-how-to-achieve-it-in-java/)
==========
Q) what is class level and object level locking?
A) Object-level lock :
Every object in java has a unique lock. Whenever we are using a synchronized keyword, then only the lock concept will come into the picture. If a thread wants to execute then synchronized method on the given object. First, it has to get a lock-in that object. Once the thread got the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock. Acquiring and release lock internally is taken care of by JVM and the programmer is not responsible for these activities.

Class level lock:
Every class in Java has a unique lock which is nothing but a class level lock. If a thread wants to execute a static synchronized method, then the thread requires a class level lock. Once a thread got the class level lock, then it is allowed to execute any static synchronized method of that class. Once method execution completes automatically thread releases the lock.
(Ref :
https://www.geeksforgeeks.org/object-level-class-level-lock-java/)
==========
Q) Parent and child class scopes related question?
A) (Ref :
https://www.geeksforgeeks.org/parent-and-child-classes-having-same-data-member-in-java/)

==========
Q) Generics advantage?
A) 1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects.
2) Type casting is not required: There is no need to typecast the object.
3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.Syntax
Syntax: ClassOrInterface<Type>
Example to use Generics in java : ArrayList<String>
==========
****
****
JSP & Servlets :
****
****
==========
Q) Servlet lifecycle
A) Servlet class is loaded.
Servlet instance is created.
init method is invoked.
service method is invoked.
destroy method is invoked.
(Ref : javatpoint.com/life-cycle-of-a-servlet)
==========
Q) Servlet context vs Servlet Config
A) ServletConfig and ServletContext, both are objects created at the time of servlet initialization and used to provide some initial parameters or configuration information to the servlet. But, the difference lies in the fact that information shared by ServletConfig is for a specific servlet, while information shared by ServletContext is available for all servlets in the web application.
ServletConfig:
ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.
ServletConfig is for a particular servlet, that means one should store servlet specific information in web.xml and retrieve them using this object.
Example:
Suppose, one is building a job portal and desires to share different email ids (which may get change over time) to recruiter and job applicant.
So, he decides to write two servlets one for handling recruiter’s request and another one for the job applicant.
Where to store email-ids?
Put email-id as a name-value pair for different servlet inside web.xml which can further be retrieved using getServletConfig().getInitParameter(“name”) in the servlet.
ServletContext:

ServletContext is the object created by Servlet Container to share initial parameters or configuration information to the whole application.
Example:
Suppose, the name of one's job portal is "NewWebsite.tg". Showing the website name at the top of webpages delivered by different servlets, one needs to store the website name in every servlet inviting redundancy. Since the information shared by ServletContext can be accessed by every Servlet, it is better to go with ServletContext and retrieve the website name using getServletContext.getInitParameter("Name") whenever required.
(Ref : https://www.geeksforgeeks.org/difference-between-servletconfig-and-servletcontext-in-java-servlet/#:~:text=ServletConfig%20is%20an%20object%20containing,retrieve%20them%20using%20this%20object.)
==========
Q) More Servlet Interview Questions :
A) https://www.javatpoint.com/servletinterview
https://www.journaldev.com/2015/servlet-interview-questions-and-answers
==========
Q) JSP Life cycle methods
A) Following steps are involved in JSP life cycle:
Translation of JSP page to Servlet
Compilation of JSP page(Compilation of JSP into test.java)
Classloading (test.java to test.class)
Instantiation(Object of the generated Servlet is created)
Initialization(jspInit() method is invoked by the container)
Request processing(_jspService()is invoked by the container)
JSP Cleanup (jspDestroy() method is invoked by the container)
(Ref : https://www.geeksforgeeks.org/life-cycle-of-jsp/)
==========
Q) JSP implicit objects
A) request - This is the HttpServletRequest object associated with the request.
response - This is the HttpServletResponse object associated with the response to the client.
out - This is the PrintWriter object used to send output to the client.
session - This is the HttpSession object associated with the request.
application - This is the ServletContext object associated with the application context.
config - This is the ServletConfig object associated with the page.
pageContext - This encapsulates use of server-specific features like higher performance JspWriters.
page - This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
Exception - The Exception object allows the exception data to be accessed by designated JSP.
(Ref : https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm)

==========
Q) JSP Tags
A) The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:
scriptlet tag
expression tag
declaration tag
(Ref : https://www.javatpoint.com/jsp-scriptlet-tag & https://www.tutorialspoint.com/jsp/jsp_custom_tags.htm)
==========
Q) More JSP interview Questions
A) https://www.javatpoint.com/jspinterview
https://www.journaldev.com/2110/jsp-interview-questions-and-answers
https://www.tutorialspoint.com/jsp/jsp_interview_questions.htm
==========
****
****
JDBC :
****
****
==========
Q) Statement Vs PrepareStatement Vs CallableStatement Vs BatchStatement

Statement :: From the connection interface, you can create the object for this interface. It is generally used for general-purpose access to databases and is useful while using static SQL statements at runtime.
Syntax:Statement statement = connection.createStatement();
Prepared Statement :: represents a recompiled SQL statement, that can be executed many times. This accepts parametrized SQL queries. In this, "?" is used instead of the parameter, one can pass the parameter dynamically by using the methods of PREPARED STATEMENT at run time.
Callable Statement :: are stored procedures which are a group of statements that we compile in the database for some task, they are beneficial when we are dealing with multiple tables with complex scenario & rather than sending multiple queries to the database, we can send the required data to the stored procedure & lower the logic executed in the database server itself. The Callable Statement interface provided by JDBC API helps in executing stored procedures.
Ref : https://www.geeksforgeeks.org/types-of-statements-in-jdbc/
https://javaconceptoftheday.com/statement-vs-preparedstatement-vs-callablestatement-in-java/
Batch Statement :
https://www.programcreek.com/java-api-examples/?api=com.datastax.driver.core.BatchStatement
==========
Q) SQL Injection means? Drawback? How overcome it?
A) The SQL Injection is a code penetration technique that might cause loss to our database. It is one of the most practiced web

hacking techniques to place malicious code in SQL statements, via webpage input. SQL injection can be used to manipulate the application's web server by malicious users.

SQL injection generally occurs when we ask a user to input their username/userID. Instead of a name or ID, the user gives us an SQL statement that we will unknowingly run on our database. For Example - we create a SELECT statement by adding a variable "demoUserID" to select a string. The variable will be fetched from user input (getRequestString).
(Ref : https://www.javatpoint.com/sql-injection)
SQL Injection and How to Prevent It? -
https://www.baeldung.com/sql-injection
==========
Q) ResultSet metadata
A) The metadata means data about data i.e. we can get further information from the data.
If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.
-- Methods
public int getColumnCount()throws SQLException - it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException - it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException - it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException - it returns the table name for the specified column index.
==========
Q) JDBC Drivers
A) Type 1: JDBC-ODBC bridge
Type 2: partial Java driver
Type 3: pure Java driver for database middleware
Type 4: pure Java driver for direct-to-database
Type 5: highly-functional drivers with superior performance
(Ref :
https://www.progress.com/faqs/datadirect-jdbc-faqs/what-are-the-types-of-jdbc-drivers)
==========
More JDBC Interview Questions :
https://www.javatpoint.com/jdbc-interview-questions
https://www.journaldev.com/2529/jdbc-interview-questions-and-answers
==========
****
****
Hibernate :
****
****
==========

Q) ORM Main Concept?
A) ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.
(Ref : https://www.tutorialspoint.com/hibernate/orm_overview.htm#:~:text=ORM%20stands%20for%20Object%2DRelational,as%20Java%2C%20C%23%2C%20etc.&text=No%20need%20to%20deal%20with,concepts%20rather%20than%20database%20structure.)

==========

Q) Different States Of Entity beans (Or) Transaction Stages?
A) Given an instance of a class that is mapped to Hibernate, it can be in any one of four different persistence states (known as hibernate entity lifecyle states):
Transient
Persistent
Detached
Removed
(Ref : https://howtodoinjava.com/hibernate/hibernate-entity-persistence-lifecycle-states/)

==========

Q) SessionFactory (Singleton) & Sessin (Not singleton) & Transaction?
A) SessionFactory:
SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
It is immutable and thread-safe in nature.
buildSessionFactory() method gathers the meta-data which is in the cfg Object.
From cfg object it takes the JDBC information and create a JDBC Connection.
SessionFactory factory=cfg.buildSessionFactory();

Session:
Session is an interface which is present in org.hibernate package.
Session object is created based upon SessionFactory object i.e. factory.
It opens the Connection/Session with Database software through Hibernate Framework.
It is a light-weight object and it is not thread-safe.
Session object is used to perform CRUD operations.
Session session=factory.buildSession();

Transaction:
Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.
Transaction tx=session.beginTransaction();

tx.commit();
(Ref :
https://www.geeksforgeeks.org/hibernate-architecture/#:~:text=SessionFactory%20is%20an%20Interface%20which,is%20in%20the%20cfg%20Object.
)
==========
Q) get() vs load()?
A) Basic :: It is used to fetch data from the database for the given identifier - It is also used to fetch data from the database for the given identifier
Null Object :: It object not found for the given identifier then it will return null object - It will throw object not found exception
Lazy or Eager loading :: It returns fully initialized object so this method eager load the object - It always returns proxy object so this method is lazy load the object
Performance :: It is slower than load() because it return fully initialized object which impact the performance of the application - It is slightly faster.
Use Case :: If you are not sure that object exist then use get() method - If you are sure that object exist then use load() method
(Ref :
https://www.tutorialspoint.com/difference-between-get-and-load-in-hibernate)
==========
Q) save() vs persist()?
A) 1)The first difference between save and persist is there return type. Similar to save method, persist also INSERT records into the database, but return type of persist is void while return type of save is Serializable Object. 2) Another difference between persisting and save is that both methods make a transient instance persistent. However, persist() method doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time.
3) One more thing which differentiates persist and save method in Hibernate is that it is their behavior on the outside of transaction boundaries. persist() method guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. save() method does not guarantee the same, it returns an identifier, and if an INSERT has to be executed to get the identifier (like "identity" generator), this INSERT happens immediately, no matter if you are inside or outside of a transaction.
4) The fourth difference between save and persist method in Hibernate is related to previous differences in saving vs. persist. Because of its above behavior of persist method outside transaction boundary, it's useful in long-running conversations with an extended Session context. On the other hand, the save method is not good in a long-running conversation with an extended Session context.
These were some differences between save, saveOrUpdate, and persist method of Hibernate. All three methods are related to saving Objects into a database, but their behavior is quite different. Knowledge of save, persist and saveOrUpdate not only helps to decide better use

of Hibernate API but also help you to do well in Hibernate interviews.
(Ref : https://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html#axzz6rzqC0Ql3)
==========
Q) save() vs saveOrUpdate()?
A)The main difference between save and saveOrUpdate method is that save() generates a new identifier and INSERT record into the database while saveOrUpdate can either INSERT or UPDATE based upon the existence of a record. Clearly, saveOrUpdate is more flexible in terms of use but it involves extra processing to find out whether a record already exists in the table or not.
In summary, the save() method saves records into the database by INSERT SQL query, Generates a new identifier, and returns the Serializable identifier back.
On the other hand saveOrUpdate() method either INSERT or UPDATE based upon the existence of an object in the database. If a persistence object already exists in the database then UPDATE SQL will execute, and if there is no corresponding object in the database, then INSERT will run.
(Ref : https://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html#axzz6rzqC0Ql3)
==========
Q) Dhirty Checking?
A)Dirty checking is an essential concept of Hibernate. The Dirty checking concept is used to keep track of the objects. It automatically detects whether an object is modified (or not) or wants to be updated.
It also allows a developer to avoid time-consuming database write actions. It modifies only those fields which require modifications, and the remaining fields are kept unchanged.
Example of Dirty Checking::
Let’s understand the concept of dirty checking with the help of an example. In this example, we are taking an entity class Student. The Student class contains student id (id), name (Sname), course (Scourse), and rollno (Srno) of the student. It also provides default and a parameterized constructor.
To access dirty checking in the application, we are using the annotation @DynamicUpdate to the entity class Student.
@DynamicUpdate- It is used for updating the objects. This annotation makes the necessary modifications and changes to the required fields.
(Ref : https://www.tutorialandexample.com/dirty-checking-in-hibernate/)
==========
Q) Lazy Loading?
A) Consider one of common Internet web application: the online store. The store maintains a catalog of products. At the crudest level, this can be modeled as a catalog entity managing a series of

product entities. In a large store, there may be tens of thousands of products grouped into various overlapping categories.
When a customer visits the store, the catalog must be loaded from the database. We probably don't want the implementation to load every single one of the entities representing the tens of thousands of products to be loaded into memory. For a sufficiently large retailer, this might not even be possible, given the amount of physical memory available on the machine.
Even if this was possible, it would probably cripple the performance of the site. Instead, we want only the catalog to load, possibly with the categories as well. Only when the user drills down into the categories should a subset of the products in that category be loaded from the database.
To manage this problem, Hibernate provides a facility called lazy loading. When enabled, an entity's associated entities will be loaded only when they are directly requested.
(Ref :
https://howtodoinjava.com/hibernate/lazy-loading-in-hibernate/)
==========
Q) Lazy Loading (Vs) Eager Loading?
A) Fetching strategy :: In Lazy loading, associated data loads only when we explicitly call getter or size method. - In Eager loading, data loading happens at the time of their parent is fetched
Default Strategy in ORM Layers :: ManyToMany and OneToMany associations used lazy loading strategy by default. - ManyToOne and OneToOne associations used lazy loading strategy by default.
Loading Configuration :: It can be enabled by using the annotation parameter : fetch = FetchType.LAZY - It can be enabled by using the annotation parameter : fetch = FetchType.EAGER
Performance :: Initial load time much smaller than Eager loading - Loading too much unnecessary data might impact performance
(Ref :
https://www.tutorialspoint.com/difference-between-lazy-and-eager-loading-in-hibernate)
==========
Q) 1st & 2nd & 3rd Level caches?
A) First-level Cache
The first-level cache is the Session cache and is a mandatory cache through which all requests must pass. The Session object keeps an object under its own power before committing it to the database.
If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.

Second-level Cache
Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second level cache can be configured on a per-class and per-collection basis and mainly responsible for

caching objects across sessions.
Any third-party cache can be used with Hibernate. An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

Query-level Cache
Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.
This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.
(Ref :
https://www.tutorialspoint.com/hibernate/hibernate_caching.htm#:~:text=Caching%20is%20a%20mechanism%20to,hits%20as%20much%20as%20possible.)
==========
Q) n-1 Problem in Hibernate?
A) The N+1 query problem happens when the data access framework executed N additional SQL statements to fetch the same data that could have been retrieved when executing the primary SQL query.
The larger the value of N, the more queries will be executed, the larger the performance impact. And, unlike the slow query log that can help you find slow running queries, the N+1 issue won't be spot because each individual additional query runs sufficiently fast to not trigger the slow query log.
The problem is executing a large number of additional queries that, overall, take sufficient time to slow down response time.
(Ref :
https://vladmihalcea.com/n-plus-1-query-problem/#:~:text=The%20N%2B1%20query%20problem%20happens%20when%20the%20data%20access,executing%20the%20primary%20SQL%20query.&text=The%20problem%20is%20executing%20a,to%20slow%20down%20response%20time.)
==========
****
****
Spring Boot?
****
****
Q) Struts vs Spring Vs SpringBoot?
A) Spring :: Struts
It is a lightweight framework. - It is a heavyweight framework.
It does not support tag library. - It supports tag library directive.
It has loosely coupled modules. -It has tightly coupled programming modules.
It is integrated with ORM Technologies using which, lesser coding is required after and before the main logic. - It supports manual coding.
It has a layered MVC architecture containing 3 layers for modelling,

viewing and controller. -It does not have a layered architecture.
(Ref : https://www.geeksforgeeks.org/spring-vs-struts-in-java/)

Spring :: SpringBoot
Spring Framework is a widely used Java EE framework for building applications. - Spring Boot Framework is widely used to develop REST APIs.
It aims to simplify Java EE development that makes developers more productive. -It aims to shorten the code length and provide the easiest way to develop Web Applications.
The primary feature of the Spring Framework is dependency injection. - The primary feature of Spring Boot is Autoconfiguration. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop loosely coupled applications. - It helps to create a stand-alone application with less configuration.
The developer writes a lot of code (boilerplate code) to do the minimal task. - It reduces boilerplate code.
To test the Spring project, we need to set up the sever explicitly. - Spring Boot offers embedded server such as Jetty and Tomcat, etc. It does not provide support for an in-memory database. - It offers several plugins for working with an embedded and in-memory database such as H2.
Developers manually define dependencies for the Spring project in pom.xml. - Spring Boot comes with the concept of starter in pom.xml file that internally takes care of downloading the dependencies JARs based on Spring Boot Requirement.
(Ref :
https://www.javatpoint.com/spring-vs-spring-boot-vs-spring-mvc)
==========
Q) IOC Container
A) 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
(Ref : https://www.javatpoint.com/ioc-container)
==========
Q) Dependency Injection
A) Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.
Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

(Ref :
https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring)
==========
Q) Spring bean scopes
A) singleton :: This scopes the bean definition to a single instance per Spring IoC container (default).
prototype :: This scopes a single bean definition to have any number of object instances.
request :: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
session :: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
global-session :: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
(Ref :
https://www.tutorialspoint.com/spring/spring_bean_scopes.htm)
==========
Q) Can we have prototype bean in singleton bean?
A) You cannot dependency-inject a prototype-scoped bean into your singleton bean, because that injection occurs only once, when the Spring container is instantiating the singleton bean and resolving and injecting its dependencies.
(Ref :
https://java2blog.com/injecting-prototype-bean-singleton-bean-spring/)
==========
Q) Spring bean Vs Java Bean?
A) Spring bean is managed by Spring IOC, Java Bean is not.
Java Bean is always serializable, Spring Bean doesn’t need to.
Java Bean must have a default no-arg constructor, Spring Bean doesn’t need to.
A Java object can be a JavaBean, a POJO and a Spring bean all at the same time.
(Ref : https://mossgreen.github.io/Java-Bean-VS-Spring-Bean/)
==========
Q) Stereotype Annotations? (@Component & @Repository& @Service & @Controler)
A) In spring autowiring, @Autowired annotation handles only wiring part. We still have to define the beans so the container is aware of them and can inject them for us.
With @Component, @Repository, @Service and @Controller annotations in place and automatic component scanning enabled, Spring will automatically import the beans into the container and inject to dependencies. These annotations are called Stereotype annotations as well.
Before jumping to example use of these annotations, let’s learn quick facts about these annotations which will help us in making a better decision about when to use which annotation.
(Ref :

https://howtodoinjava.com/spring-core/stereotype-annotations/)
==========
Q) @SpringBootApplication :: We use the @SpringBootApplication annotation in our Application or Main class to enable a host of features, e.g. Java-based Spring configuration, component scanning, and in particular for enabling Spring Boot's auto-configuration feature.
(Ref : https://dzone.com/articles/the-springbootapplication-annotation-example-in-ja)

@ComponentScan :: Using @ComponentScan in a Spring Application. With Spring, we use the @ComponentScan annotation along with @Configuration annotation to specify the packages that we want to be scanned. @ComponentScan without arguments tells Spring to scan the current package and all of its sub-packages.
(Ref : https://www.baeldung.com/spring-component-scanning#:~:text=Using%20%40ComponentScan%20in%20a%20Spring,all%20of%20its%20sub%2Dpackages.)

@EnableAutoConfiguration :: @EnableAutoConfiguration in spring boot tells how you want to configure spring, based on the jars that you have added in your classpath. For example, if you add spring-boot-starter-web dependency in your classpath, it automatically configures Tomcat and Spring MVC.
(Ref : https://www.javaguides.net/2018/09/spring-boot-enableautoconfiguration-annotation-with-example.html)

@RestController :: Spring RestController annotation is a convenience annotation that is itself annotated with @Controller and @ResponseBody . ... Spring RestController annotation is used to create RESTful web services using Spring MVC. Spring RestController takes care of mapping request data to the defined request handler method.
(Ref : https://www.journaldev.com/21536/spring-restcontroller)
==========
Q) Spring Boot - Application Properties?
A) Application Properties support us to work in different environments. In this chapter, you are going to learn how to configure and specify the properties to a Spring Boot application.
Command Line Properties
Spring Boot application converts the command line properties into Spring Boot Environment properties. Command line properties take precedence over the other property sources. By default, Spring Boot uses the 8080 port number to start the Tomcat. Let us learn how change the port number by using command line properties.
(Ref : https://www.tutorialspoint.com/spring_boot/spring_boot_application_properties.htm)
(Ref :

https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html)
==========
Q) @Autowired :: Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
Autowiring can't be used to inject primitive and string values. It works with reference only.
Advantage of Autowiring ::
It requires the less code because we don't need to write the code to inject the dependency explicitly.
Disadvantage of Autowiring ::
No control of programmer.
It can't be used for primitive and string values.

Autowiring Modes
There are many autowiring modes:
No. Mode Description
1) no It is the default autowiring mode. It means no autowiring bydefault.
2) byName The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
3) byType The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
4) constructor The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.
5) autodetect It is deprecated since Spring 3.
(Ref :
https://www.javatpoint.com/autowiring-in-spring#:~:text=Autowiring%20feature%20of%20spring%20framework,It%20works%20with%20reference%20only.)

@RequestMapping :: @RequestMapping is one of the most common annotation used in Spring Web applications. This annotation maps HTTP requests to handler methods of MVC and REST controllers.
Request Mapping Basics ::
In Spring MVC applications, the RequestDispatcher (Front Controller Below) servlet is responsible for routing incoming HTTP requests to handler methods of controllers.
When configuring Spring MVC, you need to specify the mappings between the requests and handler methods.
Spring MVC Dispatcher Servlet and @RequestMappingTo configure the mapping of web requests, you use the @RequestMapping annotation.
The @RequestMapping annotation can be applied to class-level and/or method-level in a controller.
The class-level annotation maps a specific request path or pattern onto a controller. You can then apply additional method-level annotations to make mappings more specific to handler methods.

(Ref :
https://dzone.com/articles/using-the-spring-requestmapping-annotation)
==========
****
****
Java Design Patterns ::
A design patterns are well-proved solution for solving the specific problem/task.
Now, a question will be arising in your mind what kind of specific problem? Let me explain by taking an example.
Problem Given:
Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.
Solution:
Singleton design pattern is the best solution of above specific problem. So, every design pattern has some specification or set of rules for solving the problems. What are those specifications, you will see later in the types of design patterns.

But remember one-thing, design patterns are programming language independent strategies for solving the common object-oriented design problems. That means, a design pattern represents an idea, not a particular implementation.
By using the design patterns you can make your code more flexible, reusable and maintainable. It is the most important part because java internally follows design patterns.
To become a professional software developer, you must know at least some popular solutions (i.e. design patterns) to the coding problems.

1.Creational Design Pattern
	Factory Pattern
	Abstract Factory Pattern
	Singleton Pattern
	Prototype Pattern
	Builder Pattern.
2. Structural Design Pattern
	Adapter Pattern
	Bridge Pattern
	Composite Pattern
	Decorator Pattern
	Facade Pattern
	Flyweight Pattern
	Proxy Pattern
3. Behavioral Design Pattern
	Chain Of Responsibility Pattern
	Command Pattern
	Interpreter Pattern
	Iterator Pattern

Mediator Pattern
Memento Pattern
Observer Pattern
State Pattern
Strategy Pattern
Template Pattern
Visitor Pattern

(Ref : https://www.javatpoint.com/design-patterns-in-java)
****
****
Database
****
****
==========
Q) Nth Highest Sal
A) Oracle - Select * from (select e.*,ROW_NUMBER over (order by sal desc) rn from employee where rn=N);
MySql - Select * from employee order by sal desc limit N-1,1;
==========
Q) How to swap values in two columns with single query -
A) update table_name set gender=(case when gender='M' then 'F' when gender='F' then 'M' end)
==========
3.Joins - Inner join, outer join, leftjoin, right join, leftouter-join, rightouter-join...
==========
Q) Forced views
A) FORCE keyword is used while creating a view, forcefully. This keyword is used to create a View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated.

Syntax for forced View is,

```
CREATE or REPLACE FORCE VIEW view_name AS
    SELECT column_name(s)
    FROM table_name
    WHERE condition;
```

(Ref : https://www.studytonight.com/dbms/sql-views.php)
==========