***\*\*\* Common Asked Java Interview Question \*\*\****

1. What is SOLID principle?

Ans. SOLID is combination of 5 terms-

S- Single responsibility principle

O- Open closed principle

L- Liskov substitution principle

I- Interface Segregation principle

D- Dependency Inversion principle

2. What is OOP's concept?

Ans. There is mainly 4 main oops concept-

1. Inheritance (EX- parent child relation)

2. Polymorphism (Ex – payment with UPI, Debit card, credit card, paytm multiple option)

3. Encapsulation (Ex-company and their employee, employee data bind with company class )

4. Abstraction (Ex- ATM machine)

3. Difference between comparable and comparator in java?

Ans. For natural shorting use - comparable interface (compareTo(object1)) java.util pkg

Collections.sort(list obj);

For custom shorting use - comparator interface (compare(Obj1, obj2)) java.lang pkg

Collections.sort(list obj, new Class());

4. Difference between Abstract and Interface in java?

Ans. \* An abstract class can extend another Java class and implement multiple Java interfaces.

\* An abstract class can be extended using keyword "extends".

\* A Java abstract class can have class members like private, protected, etc.

\* Abstract class can have abstract and non-abstract methods.

\* An interface can extend another Java interface only.

\* An interface can be implemented using keyword "implements".

\* Members of a Java interface are **public static final** by default.

\*Interface can have only abstract methods. Since Java 8, it can have **default** and **static** methods also.

\* Since Java 9, it can have **private** methods also.

5. Difference between Fail Fast and Fail Safe iterator in java?

Ans.

\* Fail Fast throw **ConcurrentModificationException** while modifying/changing list object when iterating list

\* Fail Fast iterator work on original collection.

\* It requires low memory

\* Example - HashMap, ArrayList, Vector, HashSet, etc

\* Fail Safe does not throw any ConcurrentModificationException while modifying list object when iterating list

\* Fail Safe iterator works on a **cloned** **collection** .

\* It requires high memory

\* Example - **CopyOnWriteArrayList**, ConcurrentHashMap, etc.

6. Can we override static method in java?

Ans- This process called **method hiding**.

7. Can we override private method in java?

Ans - No because **visibility** of method **low** when use private access modifier.

8. What is an Association, Composition and Aggregation in Java?

Ans - Association is relation between two separate classes with no ownership between classes. It can be OTO, OTM, MTM etc.

Composition and Aggregation are the two forms of association.

Aggregation is In Aggregation, both the entries can survive individually which means ending one entity will not effect the other entity (**weak** relationship)

Composition is When there is a composition between two entities, the composed object cannot exist without the other

entity.(**strong** relationship)

Aggregation relation is 'has-a' and composition is 'part-of' relation.

9. When do we use Aggregation?

Ans- **Code reuse** is best achieved by aggregation.

10. equals() and hashCode() in java ?

Ans- hashCode returns the hashCode value as an Integer. hashCode value is mostly used in hashing based collections

like HashMap, HashSet, HashTable….etc. This method must be overridden in every class which overrides equals() method.

In java equals() method is used to **compare equality of two Objects**. The equality can be compared in two ways:

1.Shallow comparison 2.Deep Comparison

When we use hash-based collections like hashMap, HashSet, HashTable need to override these 2 method.

If we do not override hashCode methed get always wrong index value .

11. How substring method works internally in Java?

Ans- Every time you call substring() method in Java, it will return a new String because String is immutable in Java.

fix memory leak issue Other option is, call intern() method on substring, which will then fetch an existing string from pool or

add it if necessary.

ex - String substr = obj.substring(0, 2).intern();

Fix for substring() in JDK 7

Sun Microsystems has changed the implementation of substring() from JdK 7. When we invoke substring() in JDK 7,

instead of referring char[] array from original String, jvm creates new String objects with its own char[] array.

String (int offset, int count, char value []);

12. What is try with resources in java 7?

Ans- try-with-resources statement is a try statement that declares one or more resources. The resource is as

an object that must be closed after finishing the program.

You can pass any object that implements java.lang.AutoCloseable, which includes all objects which

implement java.io.Closeable.

Mainly use with File , networking and connection(eg. Database) etc

try-with-resources: A resource is closed only if it initialized to a non-null value. This can actually be useful,

when a resource might present sometimes, and absent others. ... But if the reference is null, no attempt is made

to call close() on it, no NullPointerException is thrown, and the code still works.

13. What is **Classloader** and types of Classloader in java?

Ans- Classloader is program that loads byte code program(getClassLoader() method is used) into memory when we want

access any class.

A Java Classloader is of three types:

\* BootStrap ClassLoader - load class/file from rt.jar.

\* Extension ClassLoader - load class/file from jre/lib/ext diretory

\* System ClassLoader - load class from CLASSPATH

14. What is new features in java 8(Long term support LTS)?

Ans- There is major features are -

\* Lambda expressions(A lambda expression is an anonymous function eg. (x, y) -> x + y) function without name called lambda function and provide implementation for single abstract method

\* Method references (Method reference is a shorthand notation of a lambda expression to call a method eg. System.out::println)

\* Functional interfaces (@FunctionalInterface An interface with only single abstract method is called functional interface)

\* Stream API(sequential stream() and parallelStream())

\* Default methods and Static methods in interface

\* Base64 Encode Decode API

\* Optional class for dealing with nullPointerEXCEPTION

\* forEach() method

\* spliterator() use for parallel stream collection processing

\* JDBC-ODBC Bridge has been removed.

\* PermGen memory space has been removed and **metaspace** introduced

\* Java 8 Date/Time API Changes (The new classes intended to replace Date class are LocalDate, LocalTime and LocalDateTime)

\* StringJoiner class

eg. StringJoiner(can not join 2 list) mystring = new StringJoiner("-");

mystring.add("Krishna");

mystring.add("Ram");

output : Krishna-Ram

/\* Passing comma(,) as delimiter and opening bracket

\* "(" as prefix and closing bracket ")" as suffix

\*/

StringJoiner mystring = new StringJoiner(",", "(", ")");

\* Arrays Parallel Sort in jdk8

eg. int numbers[] = {22, 89, 1, 32, 19, 5};

//Parallel Sort method for sorting int array

Arrays.parallelSort(numbers);

15. What is **Stream in jdk8**?

Ans- java.util.stream pkg

using streams we can perform various aggregate operations on collection, array and i/o operation etc.

Stream API is used to process collections of objects

enable functional-style operations

Stream lazy in nature

Stream use 2 kind of operation - 1. Intermediate Operation(lazy) 2. Terminal Operation(Eager)

1. Intermediate Operation -

\* map()- List number = Arrays.asList(2,3,4,5);

List square = number.stream().map(x->x\*x).collect(Collectors.toList());

\* filter() - List names = Arrays.asList("Krishna","Ram","Hari");

List result = names.stream().filter(s->s.startsWith("R")).collect(Collectors.toList());

\* sorted() - List names = Arrays.asList("Krishna","Ram","Hari");

List result = names.stream().sorted().collect(Collectors.toList()); // alphabatically sort

\* distinct() - List<String> list = Arrays.asList("A", "B", "C", "D", "A", "B", "C");

List<String> distinctElements = list.stream().distinct().collect(Collectors.toList()); // unique result

\* limit() - Stream<Integer> evenNumInfiniteStream = Stream.iterate(0, n -> n + 2);

List<Integer> newList = evenNumInfiniteStream.limit(10).collect(Collectors.toList());

\* skip() - Stream<Integer> evenNumInfiniteStream = Stream.iterate(0, n -> n + 2);

List<Integer> newList = evenNumInfiniteStream.skip(5).limit(10).collect(Collectors.toList());

2. **Terminal Operation**

\* collect()

\* forEach() - List number = Arrays.asList(2,3,4,5);

number.stream().map(x->x\*x).forEach(y->System.out.println(y));

***Intermediate Operation Terminal Operation***

filter(), forEach() , findAny()

map(), collect(),findFirst()

distinct(), min(),noneMatch()

sorted(), max(),allMatch()

limit(), count(),anyMatch()

skip(),

16. ***Consumer functional interface*** example ?

Ans - **public** **static** **void** main(String[] args) {

List<Integer> list=**new** ArrayList<>();

list.add(10); list.add(55);

list.add(3); list.add(12);

list.add(15); list.add(23);

list.add(77); list.add(8);

// way 1 java7

**for**(Integer res : list) {

System.***out***.println(res);

}

// way 2 java8

list.forEach(s->System.***out***.println(s));

// way 3 alternative of way 2

Consumer<Integer> con=**new** // this @FuntionalInterface so have 1 abstract method

Consumer<Integer>() {

**public** **void** accept(Integer t) {

System.***out***.println(t);

};

};

list.forEach(con);

}

17. What is Lombok project ?

Ans- Below annotation **provide by Lombok**

* @Data (if use this all annotation will cover)
* @Getter
* @Setter
* @NoArgsConstructor
* @RequiredArgsConstructor
* @ToString
* @EqualsAndHashCode
* @Delegate
* @Synchronized
* @Slf4j
* @Cleanup

18. How to create **custom annotation** in java ?

Ans-

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

**public @interface** MyCustomAnnotation

{

**int** krishnaDOB();

}

There are **three** **types** of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation

19. How can you prevent a child class from being serialized when it's parent class implements Serializable interface?

Ans - In order to prevent subclass from serialization we need to **implement writeObject() and readObject() methods which are executed by JVM** during serialization and deserialization also NotSerializableException is made to be thrown from these methods.

Ex -

Class A implements Serializable{

}

class B extends A {

private void writeObject(ObjectOutputStream oos) throws IOException {

throw new NotSerializableException();

}

}

20. What is **Cohesion** and **coupling** in java ?

Ans – **Cohesion** –  focuses on a single thing so always be good high cohesion.

**Coupling** -  focuses on a relationship so always be good low coupling.

Ques 21. What is Difference between Final, Finally and Finalize?

Ans- Final – is keywards

Finally – block

Finalize- method use or cleanup operation call by jvm.

**\*\*\*\*\*\* Thread question**

1. What is the **different way to implement thread** in java ?

Ans - Using 2 way - (A) - By extending Thread class and (B) - By implementing Runnable interface.

2. What is difference between **runnable**() and **callable**() ?

Ans - Callable has call() method

Callable has call method which returns value

call method can throw checked exception

Runnable has run() method

Runnable has run method which doesn't return any value.

run method can't throw checked exception.

3. What is **join()** in thread ?

Ans- Join() method - ensure all threads that started from main must end in order in which

they started(so maintain thread execution order ) and also main should end in last.

4. What is **life-cycle of thread** ?

Ans - 1. New state

2. runnable

3. running - (blocked state and yield(), wait(), sleep() and suspend())

4. terminate

🡪> **sleep() vs wait()** in java

Ans – sleep() – related to the Thread class.

Sleep () method does not release the lock on the object

Wait()-related to the Object class.

wait () method not mark any lock on object

5. What is **yield()** method in thread ?

Ans - yield(): yield method is used to pause the execution of currently running process so that other

waiting thread with the same priority will get CPU to execute. Threads with lower priority

will not be executed on yield.

6. What will happen if we direct **call run() method instead of start() method()** ?

7. What is thread priority and default thread priority in java ?

8. What is **daemon thread** and what is the use of daemon thread ?

Ans - Daemon thread is a low-priority thread that runs in the background to perform tasks such as clean-up operation.

9. What is the way to **create threads pool in java** ?

Ans- ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads

10. What is **thread-safety** and give some example from collection ?

11. Why **wait and notify** methods are called from the synchronized block?

Ans - The main reason for calling the wait and notify method from either synchronized block or method

is that it is made mandatory by Java API. If you don't call them from a synchronized context, your

code will throw **IllegalMonitorStateException**. A more subtle reason is to avoid the race condition

between wait and notify calls.

\*\*\*🡪>> All Java programs have at least one thread, known as the main thread which is created by JVM

at the program start when the main() method is invoked.

**\*\*\* Hibernate question**

1. What is dialect in **hibernate.cfg.xml** file ?

2. What **design pattern hibernate** use ?

3. What kind of **relationship hibernate** have and its sub-type ?

4. What is different **ID generation strategies** in hibernate ?

5. What are main interfaces in Hibernate ?

6. SessionFactory vs session

7. 1st level **cache** vs 2nd level cache

8. How many instances of Session and SessionFactory are usually created ?

9. What is **object state in hibernate** ?

10. What is a **cascadeType and fetchType** in hibernate ?

11. What is the difference between **get() and load()** ?

12. What is a difference between **Hibernate and JDBC**?

13. What is minimum requiremnet for class to make a entity ?

14. What is a difference between **openSession**() and **getCurrentSession**()?

15. How to **reattach detached object again in hibernate** ?

16. What is the meaning of **'hibernate.hbm2ddl.auto'** in .cfg.xml file ?

17. Criteria vs HQL

18. What is the difference between **save() and persist()** method in Hibernate?

19. When do you use **merge() and update**() in Hibernate?

20. What is default FetchType for mapping ?

Ans - LAZY loading - @OneToMany , @ManyToMany

EAGER loading - @OneToOne , @ManyToOne

**\*\*\* Spring boot question**

1. What is difference between Spring and Spring Boot ?

2. What is internal working of @**SpringBootApplication** ?

3. @**Controller** vs @**RestController**

4. What is **Swagger** in Spring Boot ?

5. What is **Actuator** in Spring Boot ?

6. What is **Profiling** & **Dev Tool** in Spring Boot ?

7. What is **Project Lombok** & benefits ?

8. CrudRepository  vs  JpaRepository  vs PagingAndSortingRepository(bottom is – Repository -> CurdRepository -> PagingAndSortingRepository -> JpaRepository )

9. What is **Idempotent** & which method is Idempotent ?

10. What is @**Qualifier** annotation and give example ?

**What is JUnit ?**

It is basically a testing framework that is used to write test cases. It is used to check

the flow of the code or to check whether our code is working or not. JUnit framework is mainly

used to perform unit testing of our code.

Once you are done with code, you should execute all tests, and it check should pass(green color)

or fail test(red color)

@**Test**, @**Assert**, @**After**, @**Before**, @BeforeClass , @AfterClass

**assertEquals**(‘source’, ‘target’)

-🡪>>>

1. What is the difference between **arrayList and linkedList** and when do you use them ?

Ans- arrayList(internally use dynamic array) - duplicate element allow and use when search operation required

linkedList(internally use doubly-linked) - duplicate element allow and use when insertion/deletion operation required

2. What is the difference between **arrayList and vector** and when do you use them ?

Ans- arrayList - not synchronized, fail-fast, iterator, when reach on thresold point grow 50%(10 then 15)

vector - synchronized, fail-safe, iterator and listIterator, when reach on thresold point grow 100%(10 then 20)

3. What is **fail-fast and 'ConcurrentModificationException'** in collection ?

Ans- Any structural modification while iterating collection throw 'ConcurrentModificationException' called fail-fast.

4. How to sort a **list in ascending or descending order** using java 8 ?

Ans- userList.stream().sorted(Comparator.**naturalOrder**()).forEach(s->System.out.println(s));

userList.stream().sorted(Comparator.**reverseOrder**()).forEach(s->System.out.println(s));

2nd way for comparing list is

userList.stream().sorted((a,b)->b.compareTo(a)).forEach(s->System.***out***.println(s));

5. What is set **interface and its implementation classes** ?

Ans- set interface provide some classes in collection framework which does **not allow duplicate value**.

**HashSet** – insertion order is not maintained

**LinkedHashSet** – insertion order is maintained and 1 null value allow

**TreeSet** - treeSet(internally use- red-black tree(self balancing tree)) - not allow any null value, by default all value sort in ascending order

6.

Ans-

-🡪>>API

Q- What are **idempotent** ?

Ans- There are some **HTTP methods** — like GET/PUT — that produce the same response no matter how many times you use them,

sending multiple GET request to the same URI will result in same response without any side-effect. Hence,

this is known as idempotent.

On the other hand, the POST is non-idempotent, because if you send multiple POST request, it will result

in multiple resource creation on the server.

**REST** (REpresentational state transfer protocol(HTTP protocol(stateless produce)))

**Resource** - how data is represented in the REST architecture

**RestTemplate** class is an implementation of the Template method pattern in the Spring framework.

Q- What is **Swagger2**?

Ans- Swagger is used to describing the structure of APIs. Swagger 2 is an open-source service provided

in Spring Boot which makes it easier for the machines to find out the structure of APIs like RESTful Web services.

hot-swapping in Spring Boot? it achieve by using **devTools** = auto-restart.

**CORS**(Cross-Origin Resource Sharing) - authorize cross-domain requests

Cross-Site Request Forgery attack(**CSRF**)- forces other users to execute malicious commands on the application.

Spring Boot starters = set of dependency im pom.xlm

Spring Actuator = health check of application

hibernate = database independent

JDBC = high performance

Q- What is the difference between **JPA and Hibernate**?

Ans- JPA is the interface, Hibernate is one implementation of that interface. JPA is a specification

for accessing, persisting and managing the data between Java objects and the relational database. As

the definition says its API, it is only the specification. Hibernate is a JPA provider.

Q- Session.**get**() vs Session.**load**() ?

Ans- get() - return null(if data not found), eager loading, use when not sure about data available or not

load() - proxy(no direct hitting db so fast), throw objectNotFoundException, lazy loading, use when sure

data must be in db with identifire otherwise thow exception.

**dialect** = make database independent, tell database type (eg. - MySQL5Dialect, PostgreSQLDialect)

**hibernate Object States** are - 1. Transient 2. Persistent 3. Detached

**Session.evict()** : Removes the **single objec**t from the session ,After detaching the object from the session, any change to object will not be persisted - cascade=”evict”

save() vs persist() -

**save** - this method use outside a transaction, returns the generated id immediately ,

no takes care of any cascaded objects so primary entity will save

**persist** - use inside transaction , doesn’t return anything,

takes care of any cascaded objects(2 or more entity relationalship)

update() vs merge() -

**update**() - update method cannot be used when the same object exists in the session.

update can fail if an instance of the object is already in the session and throw 'NonUniqueObjectException'

**merge**() - The merge method will work fine when the same object exists in the session.

Update and Merge both work on detached instances, ie instances which have a corresponding entry in the

database but which are currently not attached to a Session. Both methods are used to convert these object

into persistence state from detached state.

**Criteria** **API** - helps us build criteria query objects dynamically.(Criteria API, HQL, native SQL queries(session.createSQLQuery))

**JDBCTransactionFactory** - is the default transaction factory in hibernate.

**evict**() - evicts a single object from the session

**clear**() - evicts all the objects(all object relate to session) in the session

**flush**() - (i.e. to write changes to the database). By default, Hibernate will flush changes automatically for you

SELECT **initcap(column\_name)** as test from table\_name group by project\_name -- input- KRISHNA output - Krishna

------------------------Map sort based on key and value ---------------------------

List<Entry<Integer, String>> sh = **map**.entrySet().stream().**sorted**(Map.Entry.**comparingByKey**()).collect(Collectors.toList());

sh.forEach(s->System.out.println(s));

System.out.println("-----");

List<Entry<Integer,String>> sh1 = **map**.entrySet().stream().**sorted**(Map.Entry.**comparingByKey**(Comparator.**reverseOrder**())).collect(Collectors.toList());

sh1.forEach(s->System.out.println(s));

// **get avg, sum, count, min,max**

**IntSummaryStatistics** result = **list**.stream().**mapToInt**(o->o).**summaryStatistics**();

System.out.println("-- "+result.getCount());

// **frequency of count**

Map<Integer,Long> noOfMaleAndFemaleEmployees = list.stream().collect(Collectors.groupingBy (Function.identity(), Collectors.counting()));

**--- list sort with java 8**

list.sort(Comparator.naturalOrder());

**servlet** - (i) have 5 method so need to implement all. init(), service(), destroy(),getServletConfig(),getServletInfo()

**genericservlet** - (Abstract c) have 3 method init(), service(), destroy()

**httpservlet** - (Abstract c) have 1 method service()

What is **functional programming** example?

Functional programming languages are specially designed to **handle symbolic computation** and list processing applications. Functional programming is **based on mathematical functions**.

Q- what is **var-args** in java ?(from java 5 onward)

Ans- Varargs is a short name for variable arguments. It represent method can **accept any number of arguments**/parameter.

ex - public void display(String... values){

System.out.println("display method invoked ");

}

Q- What is **try-with-resources** in java ?

Ans - try-with-resources statement **ensures that each resource is closed at the end of the statement execution**. so no need to take care all these by developer.

Closeable(connect with IO calling close() method ) extends AutoCloseable

Q- What is **covariant-return type** ?

Ans - It allows to narrow down return type of an overridden method **without any need to cast the type or check the return type** .

String address1 = **InetAddress**.getLocalHost().getHostAddress();

System.out.println("address1: "+address1); // for finding **local system ip address**

insert into test2 select id,dob from test; // **1 table data copy into another table**

CREATE TABLE test3 AS SELECT \* FROM test WHERE 1=2 // **only structure copy**

**openSession** - always create new session. explicitly flush and close session objects

**getCurrentSession** - It creates a new Session if not exists. hibernate close session object

Q- Can you declare Entity class as final in hibernate?

Ans- yes, but not good practice(because hibernate uses proxy pattern for lazy initialisation)

**1st level Cache** - Session , by default enable,

**2nd level Cache** - SessionFactory , not enable by default(eg. EHCache )

**Lazy loading** - means when you load parent, child objects won’t get loaded until requested

(so improves performance)

**LazyInitializationException**- when you use lazy loading and child objects are accessed after closing session.

<property name="hibernate.cache.use\_query\_cache">true</property>

and in query use - .setCacheable(true)(use when same query hit again again so improves performance)

**Cascade Operation** - ALL | DETACH | MERGE | PERSIST | REFRESH | REMOVE

**Spring IoC container**(Application context) - creating the objects, managing them (with dependency injection (DI)),

wiring them together, configuring them, as also managing their complete lifecycle

(BeanFactory is the heart of any spring-based application)

**beans lifecycle** methods - @PostConstruct((init-method) and @PreDestroy(destroy-method)

**DispatcherServlet**(front-controller) - to handle http request- response

**Java Persistence API (JPA)-** is a Java application programming interface specification/rules that describes

the management of relational data.

2 types of **IoC containers** in Spring - 1. ApplicationContext(advance) 2.BeanFactory(basic)

**5 type Bean scopes** - Singleton, Prototype, Request, Session, Global session

used **in-memory databases are H2** database(default provide by spring env. and default db name is 'testdb')

logging.level.org.springframework.web: DEBUG

logging.level.org.hibernate: ERROR

JSON Tree Model (JsonNode)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**URI** - Uniform Resource Identifier

**SOAP** - Simple Object Access Protocol

**REST** - Representational State Transfer

**AJAX** - Asynchronous javascript and XML

**JSON** - JavaScript Object Notation

partial update - they can use HTTP **PATCH**

replace an existing Resource entirely - they can use **PUT**.

check the current version in postgresql - **SELECT version();**

**Lambda Expressions** - provide implementation for **Anonymous Function**(functional interface) that allows users

to pass methods as arguments.

In lambda expression, we don't need to define the **method** again for providing the implementation. Here, we

just write the **implementation code**.

lambda expression is treated as a function.

helps removes a lot of **boilerplate code**

Lambda **syntax define 3 way** -

1. no parameter 2. 1 parameter 3. 2 parameter

(p1) -> {

//Body of single parameter lambda

}

**Stream API** - is used to process collections of objects and **perform computational operation(mathematical operation**)

using intermediate(filter(),sorted(),map(),distinct() etc) and terminal(count(),forEach(),min() max())

operation those are **lazy in nature** so cpu core optimization high.

Stream is 2 type - 1. **sequential stream 2. parallel stream**

Q- Difference between **Intermediate vs Terminal operations** ?

Ans -

**Intermediate operations are lazily loaded.**

do not produce result

return stream so can be use further operation

**Terminal operations are eagerly loaded**.

produce result

return non-stream so can not use again

**\*\* Read file in java**

1. **FileReader** - get file with path

2. **BufferedReader** br - ready to read available file

3. br.**readLine**() - read line by line till EOF(But first check br is not null)

  \*\*\* spring.jpa.properties.hibernate.enable\_lazy\_load\_no\_trans=true

While **using lazy loading in Hibernate**, we might face exceptions, saying there is no session.

that each fetch of a lazy entity will open a temporary session and run inside a separate transaction.

When retrieving lazily-loaded data, there are two steps in the process. First, there's populating the main object,

and second, retrieving the data within its proxies. Loading data always requires an open Session in Hibernate.

The problem arises when the second step happens after the transaction has closed, which leads to a **LazyInitializationException**.

\*\* **For cunsuming api**

1. KrishnaStudent quote = resTemplate.**getForObject**("https://api.agify.io/?name="+name, KrishnaStudent.class);

     (direct access object)

2. ResponseEntity<String> list = resTemplate.**exchange**("https://api.agify.io/?name="+name, HttpMethod.GET, null,String.class);

     (list.getBody() // so you will get actual value)

then ObjectMapper objectMapper - read value with bind class(like-Student.class)

\*\*\*\* normal key value make

and convert into json using -

Gson gson = new **GsonBuilder().setPrettyPrinting().create();**

        String json = gson.**toJson**(ob);

// check convert json is valid format or not

JSONObject b = new JSONObject(new **JSONTokener**(json));

\*\* **for preventing csrf we use in all request form**

<input type="hidden" name="${\_csrf.**parameterName**}" value="${\_csrf.**token**}">

and add 'org.springframework.security' in pom.xml

and use **WebSecurityConfigurerAdapter** class for manual configuration(protected void configure(**HttpSecurity** http){} ye method hota hai)

Q- Can we execute a static block **without declaring main() method**?

Answer: **No** since JDK 1.7 it is not possible to execute any java class without main() method. But it was one of the ways till JDK 1.6.

(Error: Main method not found in class yourclassname)

\*\*\*\*\*\*\*\*\*\*\* **api response code**

1xx Informational

2xx Success

3xx Redirection

4xx Client Error

5xx Server Error

@**Scheduled**(cron = "0 0 19 \* \* \*") - use for **enabling scheduling**

(use above the method)

**\*\*\* Problem caused by substring() in JDK 6**

This method works well with small Strings. But when it comes with taking substring() from a

String with more characters, it leads to serious memory issues if you are using JDK 6 or below.

So This issue should be handled by developers in jdk6.

**Fix for substring() in JDK 7**

Sun Microsystems has changed the implementation of substring() from JdK 7. When we invoke substring() in JDK 7,

instead of referring char[] array from original String, jvm creates new String objects with its own char[] array.

thus making original string eligible for garbage collection.

Q- How to **read values from properties file** in spring boot?

Ans - Using 2 Way -

1. Using @**Value**("${key}")

String password;

2. Using @**Autowired**

private Environment env;

env.getProperty("key")

Q- How **to enable swagger-2** in spring boot ?

Ans - add dependency in pom.xml

 1. springfox-swagger2

2. springfox-swagger-ui

Then use blow according to your project-

@Configuration

@EnableSwagger2

public class SwaggerConfig extends WebMvcConfigurationSupport {

@Bean

   public Docket api() {

       return new Docket(DocumentationType.SWAGGER\_2)

         .select()

         .apis(RequestHandlerSelectors.any())

         .paths(PathSelectors.any())

         .build();

   }

  @Override

    protected void addResourceHandlers(ResourceHandlerRegistry registry) {

        registry.addResourceHandler("swagger-ui.html").addResourceLocations("classpath:/META-INF/resources/");

        registry.addResourceHandler("/webjars/\*\*").addResourceLocations("classpath:/META-INF/resources/webjars/");

    }}

and what url you will hit -   **http://localhost:8082/swagger-ui.html**  (so you will get your all controller details here and get list of operation)

Q- What is **an actuator in spring boot**(health/metrics check in application) ?

Ans - add dependency in pom - spring-boot-starter-actuator

http://localhost:8082/actuator

http://localhost:8082/actuator/health

http://localhost:8082/actuator/beans

http://localhost:8082/actuator/info

 Q- How to **enable security in spring boot** ?

Ans - add dependency in pom.xml - spring-boot-starter-security

And use beleow code according to your project-

@Configuration

@EnableWebSecurity

public class SpringSecurityConfig extends **WebSecurityConfigurerAdapter** {

@Override

protected void **configure**(**HttpSecurity** http) throws Exception {

http.csrf().disable().authorizeRequests()

.antMatchers(HttpMethod.POST, "/login").permitAll()

.antMatchers("/nitiaayog/\*\*").permitAll()

.antMatchers("/aebas/\*\*").permitAll()

.anyRequest().authenticated()

.and().httpBasic()

.authenticationEntryPoint(authEntryPoint);

}

@Bean

public **PasswordEncoder** passwordEncoder(){

PasswordEncoder encoder = new **BCryptPasswordEncoder**();

return encoder;

}

}

clean code principles (**SOLID, DRY, KISS**)

Aggregate functions are often used with the **GROUP BY and HAVING clauses** of the SELECT statement. Following are the widely used **SQL aggregate functions**:

AVG() |COUNT()|MIN()|MAX() |SUM() |FIRST() |LAST()

Q-1 What is difference **between include() vs forward in servlet** ?

Ans -

**Include**() will include another file in the current file

leaves the output stream open

  RequestDispatcher rd=request.getRequestDispatcher("/index.html");

       rd.include(request, response);

**Forward**() - (erver side redirection to 1 page to another page) will forward the current request to the forwarding page.

close the output stream after it has been invoked

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

        rd.forward(request, response);

Q-2 What is **sendRedirect**() ?

Ans - used to redirect response to another resource, inside or outside the server .

ex- response.sendRedirect( "www.facebook.com );

Q-3 what is **load on startup in servlet** ?

Ans - The load-on-startup element of web-app loads the servlet at the time of project deployment or server start if value is

positive. It is also known as pre initialization.

As you know well, servlet is loaded at first request. That means it consumes more time at first request.

Q-4 What **is @WebServlet**("/Simple") and why we use it ?

Ans- If you use annotation, deployment descriptor (web.xml file) is not required. But you should have tomcat7.

@WebServlet annotation is used to map the servlet with the specified name.

Q-5 How to **print output in servlet** ?

Ans- PrintWriter pw = ServletRequest req.getWriter();

    pw.println("you output message");

**inversion of control (IoC)** - IOC is a concept where the flow of application is inverted(Developer side to spring container side)

**Dependency Injection (DI)** - injects objects into other objects or dependencies(for loose coupling of components)

dependencies might be inject by using 1. Constructor-Based Injection(when all mandatory) 2. Setter Injection(when needed optional)

3. property-based Injection

IoC can be called the design principle and DI is its implementation.

Spring boot uses below design pattern -

MVC D. Pattern

Proxy D. Pattern - (Especially in Spring AOP)

Factory D. Pattern - load beans using BeanFactory and Application context.

Singleton D. Pattern - in singleton bean creation

DI/IOC Pattern -

FrontController Pattern - design pattern using DispatcherServlet, to dispatch incoming requests to the correct controllers.

When to use the **volatile variable in Java**?

when you need to instruct the JVM that a variable can be modified by multiple threads and

give hint to JVM that does not cache its value(read value from main memory).

How does **\*\*ConcurrentHashMap\*\* achieve scalability**?

by dividing the map into segments and only locking during the write operation.

What is the **load factor of \*\*HashMap\*\* means**?

The threshold that triggers the re-sizing of HashMap is generally 0.75, which means

HashMap resize itself if it's 75 percent full.

Difference between **\*\*PATH\*\* and \*\*Classpath\*\*** in Java?

PATH is used by the OS while Classpath is used by JVM to locate Java binary

What is **\*\*IdentityHashMap**\*\* in Java?

A Map, which uses the == equality operator to check equality instead of the equals() method.

**Why wait and notify method are declared in \*\*Object\*\* class in Java**?

because they require lock which is only available to an object.

Spring **Bean Life Cycle** – @PostConstruct, @PreDestroy

Below is a simple class that will be configured as spring bean and for post-init and pre-destroy methods, we are using

@PostConstruct and @PreDestroy annotations.

import javax.annotation.PostConstruct/PreDestroy;

public class MyService {

@PostConstruct

public void init(){

System.out.println("MyService init method called");

}

public MyService(){

System.out.println("MyService no-args constructor called");

}

@PreDestroy

public void destory(){

System.out.println("MyService destroy method called");

}}

Spring provides different types of ApplicationContext containers suitable for different requirements. These are implementations of the ApplicationContext interface.

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationcontext/user-bean-config.xml");

ApplicationContext context = new AnnotationConfigApplicationContext(AccountConfig.class);

XmlWebApplicationContext context = new XmlWebApplicationContext();

FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext(path);

BeanFactory is the root interface of Spring IoC container. ApplicationContext is the child interface of BeanFactory interface

that provide Spring AOP features, i18n etc.

Some of the useful child-interfaces of ApplicationContext are ConfigurableApplicationContext and WebApplicationContext

\*\*\*\*\*custom exception example \*\*\*\*\*\*\*\*

//public class FinalClassExample extends RuntimeException{   // runtime exception

public class FinalClassExample extends Exception{   // compile  exception

// class make final so no other class extend/inherit it

// value provide by initializing the constructor

// no setter only getter

//When exposing methods which modify the state of the class, you must always return a new instance of the class.

//If the class holds a mutable object:  \*\*\* Make sure to always return a clone copy of the field and never return the

real object instance.

public FinalClassExample(String msg) {

super(msg);

}}

\*\*\*\*\*start  jsp

**Include Directive** - Includes content at page translation time

Preferred in Static Pages

Page size can be increased in the run-time

**Include Action** - Includes content at page request time

Preferred in Dynamic Pages

Page size is fixed

**ServletContext** is designed to provide information about the Container

**PageContext** is designed to provide information about the Request

\*\*\*\*\*end  jsp

4. Explain the **Servlet API**.

A servlet does not have a main() method, unlike a regular Java program, and just like an applet. It has some methods of a

servlet that are called upon by the server for the purpose of handling requests. It invokes the servlet’s service() method,

every time the server sends a request to a servlet.

To handle requests that are appropriate for the servlet, a typical servlet must override its service() method. The service()

method allows 2 parameters: these are the request object and the response object. The request object is used to inform the

servlet about the request, whereas the response object is used to then give a response.

As opposed to this, an HTTP servlet typically does not override the service() method. However, it actually overrides the

doGet() to handle the GET requests and the doPost() to handle POST requests. Depending on the type of requests it needs to

handle, an HTTP servlet can override either or both of these methods.

**web server** - responsibility is to handler HTTP requests from client browsers (ex- apache tomcat)

**application server** - provide additional features such as Enterprise JavaBeans support, JMS Messaging support, Transaction

Management, etc(Ex- JBOSS/Glassfish)

**idempotent** - An HTTP method is said to be idempotent if it returns the same result every time(ex- GET/PUT/DELETE)

\*\*\*Idempotent methods are also known as safe methods and we don’t care about the repetitive request from the client for

safe methods.

**non-idempotent** - not return the same result (ex - POST)

Most of the times, web applications are accessed using HTTP protocol and that’s why we mostly extend HttpServlet class.

**ServletConfig** - is a unique object per servlet (getServletConfig() method)

**ServletContext** - is a unique object for complete application.(getServletContext() method )

\*\*\*\* both use for show output in servlet

**PrintWriter** - is a character-stream class

**ServletOutputStream** - is a byte-stream class.

Q- Do we need to **override service() method**?

An - When servlet container receives client request, it invokes the service() method which in turn invokes the doGet(),

doPost() methods based on the HTTP method of request. I don’t see any use case where we would like to override the service()

method. The whole purpose of service() method is to forward to request to corresponding HTTP method implementations.

If we have to do some pre-processing of request, we can always use servlet filters and listeners.

**inter-servlet communication**  - use forward() and include() method

Q- What are different methods **of session management in servlets**?

Ans - The session is a conversational state between client and server and it can consist of multiple request and response

between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some

unique information about the session (session-id) is passed between server and client in every request and response.

Q- How does **Cookies work in Servlets**?

Ans - Cookies are used a lot in web client-server communication, it’s not something specific to java. Cookies are text data

sent by server to the client and it gets saved at the client local machine.

Q- How to notify an object in **session when session is invalidated or timed-out**?

Ans - If we have to make sure an object gets notified when session is destroyed, the object should implement

javax.servlet.http.HttpSessionBindingListener interface. This interface defines two callback methods – valueBound() and

valueUnbound() that we can define to implement processing logic when the object is added as attribute to the session and

when session is destroyed.

Q- What is the effective way to make sure **all the servlets are accessible only when the user has a valid session**?

Ans- We know that servlet filters can be used to intercept request between a servlet container and servlet, we can utilize

it to create an authentication filter and check if the request contains a valid session or not.

application has a single entry point - user login

Q- How to make sure a servlet **is loaded at the application startup**?

Ans- Usually, servlet container loads a servlet on the first client request. Sometimes the servlet is heavy and takes time to

loads, we might want to load it on application startup. We can use a load-on-startup element with servlet configuration in

the web.xml file or use WebServlet annotation loadOnStartup variable to tell the container to load the servlet on system

startup.

The load-on-startup value should be int, if it’s a negative integer then servlet container will load the servlet based on

client requests and requirement but if it’s 0 or positive, then the container will load it on application startup.

If there are multiple servlets with load-on-startup value such as 0,1,2,3 then lower integer value servlet will be loaded first.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

spring - Rod Johnson. In 2003

hibernate - 2001 by Gavin King

java - James Gosling 1991

spring security - Ben Alex 2004

struts - Craig McClanaha 2000

javascript - Brendan Eich in 1995

jquery - John Resig 2006

RDBBMS - Edgar F. Codd

**Boxing and Unboxing** - primitive data type convert into wrapper and vice versa.

Type Conversion -  is 2 type

1. **Implicit** (Automatic) Type Conversion

2. **Explicit** Type Conversion

(**BSILFD**) narrow -> wide and vice versa

Q- **Reflection** in Java?

Ans - Reflection in Java is an API(Application Programming Interface) that is used at runtime to analyze or change classes,

methods, and interfaces.

It is a process of examining or modifying the run time behavior of a class at run time.

\*\*\* -->>>**accessing private method using reflection api**

public class Private\_Method {

private void getMsg(String s) {

System.out.println("welcome to site "+s);

}}

public static void **main**(String[] args) throws Exception {

Class c = Class.forName("com.reflection.Private\_Method"); // provide complete class path

Object obj = c.newInstance();

     //Method method = c.getDeclaredMethod("getMsg", null);

     Method method=c.getDeclaredMethod("getMsg",new Class[]{String.class}); // if you want pass any value/parameter

     method.setAccessible(true);

     //method.invoke(obj, null);

     method.invoke(obj, "Krishna");// if you want pass any value/parameter

}

**static** block -> **Initialization** block -> and finally **Constructor**.

**static** block -> This static block will be get executed only once when the control come to the class.(JVM Load this class)

**Initialization** block -> This Initialization block will be get executed whenever a new object Created for the Class (It will

be executed from second statement of the Constructor then following constructor statements- remember First statement of the

**Constructor** will be Super()/this())

Constructor -> This will be get whenever a new object is created.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**how to read js file in java (using Nashorn engine)**

(and java and js file in same folder)

**jsfunctions.js**

function display (name) {

print("Welcome to : ",name);

}

**Test.java**

public static void main(String[] args) {

try {

ScriptEngineManager manager = new ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("JavaScript");

// read script file

engine.eval(Files.newBufferedReader(

Paths.get("F:\\sts\_new\_workspace\_testing\_purpose\\Java\_8\_Functional\_Interface\\src\\com\\md5\\jsfunctions.js")));

Invocable inv = (Invocable) engine;

// call function from script file

inv.invokeFunction("display", "krishna");

}

catch (Exception e){

   e.printStackTrace();

  }}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SHA-256**(Secure Hash Algorithm ) - generates an almost-unique 256-bit (32-byte)

signature for a text.

It is deterministic – i.e. if you apply the algorithm for the same data again, you will get the same hash value.

(it is a 'one-way' cryptographic algorithm)

**MD5**(message digest - 128 bits long) - it is one way cryptographic hashing algorithm.

MD5 is most commonly used to verify the integrity of files.

\*\*\*\*\*\*\*\*\*\*\*

**1st level cache** - by default enable, session level ,

**2nd level cache** - need to enable, sessionFactory level , ex is EH Cache

for enabling 2nd level cache we use-

hibernate.cfg.xml - <property name="cache.use\_second\_level\_cache">true</property>

<property name="cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

in pom.xml - <dependency>

    <groupId>net.sf.ehcache</groupId>

    <artifactId>ehcache</artifactId>

    <version>2.10.3</version>

</dependency>

**advantages** -  improves application performance

**Session.evict(): to remove the cached/stored entity.**

cache is stored in the RAM only. And because of that, it gives faster access to data rather than databases.

1) You can use the Session.evict() to remove the loaded entity from the first level cache, can use the refresh()

method to refresh the cache, and can use the clear() method to remove all entities in cache.

\*\*\*\*\*\*\*\*\*\*\*\*\* //**alt+shift+r == to rename multiple variable** name same type

Spring boot cache annotations  --

1 - **@EnableCaching** with class annotated with @SpringBootApplication

2 - **@Cacheable**  - It is used on the method level to let spring know that the response of the method are cacheable.

**Var js** - has a function Scope.

**let js** - has a Block Scope.

**==**  operator - Type Converting Operator, compares Value, do not compare type

=== operator - Strict Equality Operator, compares both value and type

‘**null’** - in ‘null’ we will define a variable and assign the ‘null’ value to the variable

‘**undefined’** - we will define a variable, but we won’t assign a value to that variable

Ex - type of (null)

**data type in javascript** -

Boolean , Null , Undefined, Number, BigInt, String, Symbol, ,

// System.out.println(" -- "+1/0);   **//AE**

      //  System.out.println(" -- "+0/1);  **// 0**

     //   System.out.println(" -- "+1.0/0.0);  **// Infinity**

// System.out.println(" -- "+0.0/0.0); **//Nan**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Loose coupling** - in Java means that the classes/Objects are dependent on each other.

**tight coupling** is usually not good because it reduces the flexibility and re-usability of the code

how to achieve loose coupling - using abstract classes or interface

@EnableAsync(apply on main class @SpringBootApplication) - Spring's ability to run @Async methods in a background thread pool.

pring will execute it in a separate thread and the caller of the method will not wait till the method is completed execution.

use '@Async' on method level

**// \*\*\* start db query part**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

select **initcap**(user\_design),\* from mst\_user where login\_id = 'your value'; // input- KRISHNA SINGH  output - Krishna Singh

select **upper**(user\_design),\* from mst\_user where login\_id = 'your value'; // input- krishna   output - KRISHNA

select **lower**(user\_design),\* from mst\_user where login\_id = 'your value'; // input- KRISHNA   output - krishna

SELECT COUNT(\*)

FROM INFORMATION\_SCHEMA.columns

WHERE  table\_name = 'mst\_user' ;     -- get number for how many columns in table ex. 3 columns(id.name,address)

SELECT \* FROM information\_schema.sequences;  -- get list of all available sequences

select **last\_value**  from bank\_seq; // tell value increase from this(ex 318 so next value increase as 319)

select **nextval**('bank\_seq'); // increment by 1 in sequence

select **currval**('bank\_seq');  // check current sequences of table

select **CAST**(level\_cd AS Integer) alias\_coloumn\_name,\* from mst\_user limit 10;  // change 1 value data type into another data

type

SELECT **COALESCE**(NULL, 11); // if null get then return 0 but no return null

SELECT now(); -- output 2022-03-09 10:58:00.934423+05:30

select cast(now() as date) ;  -- output  2022-03-09

select cast(**now**() as time) ;  -- output 10:58:33.526311

SELECT TO\_CHAR(NOW(), 'dd/MM/YYYY'); -- as user formatted - output 09/03/2022

SELECT TO\_**CHAR**(NOW(), 'hh : mm : ss AM') as current\_time; -- for time formated - output 10 : 03 : 06 AM

SELECT **sqrt**(25); -- o/p - 5

SELECT **TRIM**('     Welcome to site!     ') AS TrimmedString; -- o/p - Welcome to site!

SELECT **REVERSE**('krishna'); -- output - anhsirk

SELECT **replace**('mrishna', 'm', 'K'); -- output - Krishna

SELECT **length**('krishna'); -- o/p - 7

**// \*\*\* end db query part**

Q- How **to enable scheduling in spring boot** ?

Ans -   use 2 annotation -

@**EnableScheduling** (in main class with @SpringBootApplication)

and //@**Scheduled**(cron = "0 \* \* \* \* \*")  (above the method with pattern those want to trigger)

Struts 1.x - class should be extends  Action (or DispatchAction) class, return of method ActionForward type

Struts1.x - form beans class must extend ActionForm or ValidatorForm

Struts 2.x -  no need to extends , return of method can be simple string type

Struts 2.x - class might be simple pojo

**What is new when using MS**

- high cohesion - well focused on single task

- low coupling - loosely coupled each service via service discovery(API gateway)

- fault tolerance - if 1 service down no impact on other service(whole project will not down)

(easily handle using fault tolerance design patterns - hystrix circuit breakers)

- use different technology together(ex- java, php, python)

- use different database for different service(ex- rdbms, nosql etc)

- each service has their own security(ex- JWT, simple oauth2.0 , simple password based, etc)

- Ease of Understanding of the Codebase when new member join the project in team

**cons of microservices**

- welled experience developer need

- new in market so might be Community issue

- when project grow ,which service depend on which service this issue you will face

- network latency(service to service call via - rest API(Feign(synchronous ) is a declarative web service )), fault tolerance,

handle asynchronous operation issue

- hard to maintain logs/metrics for different service

- There is a higher chance of failure during communication between different services

- The external API communication in microservice architecture leads to an increased risk of attacks.

- Testing/Debugging is Difficult.

monolithic applications (SOA - Service Oriented Application)

microservice architecture -

Developers must implement the inter-service communication mechanism.

Microservice is costly, as you always need to maintain various server space for different business tasks

SOA is focused on application service reusability while Microservices are more focused on decoupling.

SOA involves sharing data storage between services while in Microservices, each service can have independent data storage.

**@Order Annotations** -  (as well as the Ordered interface) implies a specific order, in which the beans will be loaded or

prioritized by Spring.

ex- @Order(1)

**zipkin** - is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in service

architectures monitors and manages the Spring logs of your application.

**sleuth** - Spring Cloud Sleuth is used to generate and attach the trace id, span id to the logs so that these can then be used

by tools like Zipkin.

**Prometheus** - is a open source monitoring/alert tool for microservices. It helps us keep track of system metrics over a given

time period and can be used to determine the health of a software system. Metrics include memory usage and CPU consumption.

(embadded time series database)

(record real-time metrics in time series)

**Grafana** - is an open source data visualization tool. It can help you build dashboards to visualize the metrics being provided

by Prometheus data source.

**resilience4j/Hystrix** - is a lightweight, easy-to-use fault tolerance/circuit breaker library inspired by. Netflix Hystrix,

but designed for Java 8 and functional programming.

It helps stop cascading failures and allows you to fail fast and rapidly recover, or fallback and gracefully degrade

**Ribbon** -  is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients.

**Feign client**- is a declarative web service client. It makes writing web service clients easier.

by which microservices can use to communicate with each other without writing detailed REST client code.

**Eureka server**- Eureka Server is an application that holds the information about all client-service applications. Every Micro

service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP

address. Eureka Server is also known as Discovery Server;

**Api gateway**(Zuul Server)- Zuul is built to enable dynamic routing, monitoring

**Application Programming Interface (API)** - A software interface that allows users to access data & interact with other programs.

(allows two applications to talk to each other)

(an API is the messenger that delivers your request to the provider that you're requesting it from and then delivers the

response back to you)

**Web service** - A Web service is a method of communication between two application/machine/software over a network.

Q- What is **Projection in hibernate**?

1)Fetching only certain columns of the table

2)Performing aggregations like count,sum,max,min,avg.

3)Fetching only the columns which are needed improves the performance of the query.

**zxing library** (QR Code generation)

**highChart library** - for chart and graph generation in java

@**CrossOrigin**(origins = "\*") - (Cross-Origin Resource Sharing) - allow cross domain requests

- is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from

which the first resource was served.

Q- What is the use of **c3p0 in Hibernate**?

c3p0 is a Java library that provides a convenient way for managing database connections. In short, it achieves this by

creating a pool of connections.

database connection pooling capability

A **filter** - is an object that is invoked at the preprocessing and postprocessing of a request.

call **garbage collector** -- System.gc() or Runtime.gc() method.

**'StackOverflowError'** generates errors in java(Stack memory)

eg. public static void main(String[] args) {

main(new String[1]);

}

**'OutOfMemoryError'** generates errors in java(Heap memory)

eg. Long a[] = new Long[100000 \* 1000000];

System.out.println(a);

\*\*\***using PK-FK**

@OneToMany(mappedBy="emp",cascade = CascadeType.ALL)

private Set<Vehicles> vehicles = new HashSet<>();

@ManyToOne

@JoinColumn(name="vehicles\_detail\_id", nullable=false)

private Employee emp;

**\*\*\* using table**

@OneToMany(mappedBy="emp",cascade = CascadeType.ALL)

private Set<Vehicles> vehicles = new HashSet<>();

@ManyToOne

@JoinTable(name="employee\_vehicles\_table", joinColumns={@JoinColumn(name="vehicles\_id", referencedColumnName="vehicleId")}

, inverseJoinColumns={@JoinColumn(name="employee\_id", referencedColumnName="empId")})

private Employee emp;

**JAX-B** is an abbreviation for Java Architecture for XML Binding. This is used to convert XML to Java object and Java object to

XML.

using JAX-RS to create RESTful web services.

JAX-RS is an specification (just a definition) and Jersey is a JAX-RS implementation.

JAX-RS is a specification for implementing REST web services in Java.

**Jackson** - a library for data binding. It supports both XML and JSON as document formats, and implements the JAXB API

\*\*\*\*\*\*\* **Default FetchType of mapping**

OTO - EAGER

MTO - EAGER

OTM - LAZY

MTM - LAZY

Q- What is the difference between **poll() and remove()** method?

Answer: Both poll() and remove() take out the object from the Queue but if poll() fails, then it returns null. However, if

remove() fails, it throws exception.

Q-Is it possible for **two unequal objects to have the same hashcode?**

Answer: Yes, two unequal objects can have the same hashcode. This is why collision can occur in hashmap. The equal hashcode

contract only says that two equal objects must have the identical hashcode, but there is no indication to say anything about

the unequal object.

Q- What is the difference between **java.util.Date and java.sql.Date** in Java?

java.util.Date - represents a specific instant in time with millisecond precision. It represents both date and time

information without timezone.

The java.util.Date class implements Serializable, Cloneable and Comparable interface

java.sql.Date - it stores years, months and days while hour, minute, second and millisecond are ignored.

represents date without time information and it should be used only when dealing with databases.

Q- What’s the difference **between unit, integration and functional testing**?

Answer: A unit **tests** tests a small isolated piece of code such as a method. It doesn’t interact with any other systems such

as a database or another application.

An **integration** test tests how your code plays with other systems, such as a database, a cache or a third-party application.

A **functional** test is the testing of the complete functionality of an application. This may involve the use of an automated

tool to carry out more complex user interactions with your system. It tests certain flows/use cases of your application.

**anagrams** string ex - army and mary

**LoD(Law of Demeter)** violation - defines, how objects interact with each other. It says that you should talk only to the

object that you directly know. Sometimes you can hear about it is 'talk to friends and not stranger' rule.

**\*\*\* using MTM as bidirectional relation**

@ManyToMany(cascade = CascadeType.ALL)

@JoinTable(name = "student\_subject",

joinColumns = @JoinColumn(name = "student\_id", referencedColumnName = "studentId"),

inverseJoinColumns = @JoinColumn(name = "subject\_id", referencedColumnName = "subjectId"))

private Set<Subject> sub;

@ManyToMany(mappedBy = "sub")

private Set<Student> stu = new HashSet<>() ;

(so new 3rd table will create for maintaining relation)

**Hibernate framework** - is java based ORM tool that provides mapping java application objects to the relational database

tables and vice versa

Java Persistence API (JPA) - provides specification for managing the relational data in applications

(javax.persistence package) Using JPA annotation helps us in writing implementation independent code.

**Hibernate benefits -**

1. application makes database independent and java object direct map into database.

2. lazy initialization using proxy objects and perform actual database queries only when it’s required.

3. cache(1st level(by default enable) and 2nd level) helps us in getting better performance

4. Hibernate provides a powerful query language (HQL) that is is fully object-oriented and provide criteria

API(many pre-define method).

5. Hibernate create table structure so do not create manually.

6. Hibernate provide transaction(commit,rollback) support internally so don't worry developer about it.

7. developer does not take care about checked exception.(handle by framework)

8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection

pool(for code re-usability).

The internal state of a SessionFactory is immutable. Once it is created this internal state is set. This internal

state includes all of the metadata about Object/Relational Mapping.

SessionFactory also provide methods to get the Class metadata and Statistics instance to get the stats of query

executions, second level cache details etc.

Q - Hibernate **SessionFactory** is thread safe?

Internal state of SessionFactory is immutable, so it’s thread safe. Multiple threads can access it simultaneously

to get Session instances.

**handle cyclic redundancy/Infinite Recursion in mapping(OTM, MTM)**

@**JsonManagedReference**(parent side), @**JsonBackReference**(child side)(those want to ignore)

or

@**JsonIgnore** annotation to simply ignore one of the sides of the relationship

**@CrossOrigin** - CORS stands for Cross-Origin Resource Sharing. CORS is a mechanism that allows restricted resources on

a web page to be requested from another domain, outside the domain from which the resource originated

**Hibernate Session is thread safe?**

Hibernate Session object is not thread safe, every thread should get it’s own session instance and close it after it’s

work is finished.

**session.merge() method** - is used when we want to change a detached entity into the persistent state again, and it will

automatically update the database.

Q- What is difference between **Hibernate save(), saveOrUpdate() and persist()** methods?

Ans- Hibernate **save** can be used to save entity to database. Problem with save() is that it can be invoked without a

transaction and if we have mapping entities, then only the primary object gets saved causing data inconsistencies. Also save

returns the generated id immediately.

Hibernate **persist** is similar to save with transaction. I feel it’s better than save because we can’t use it outside the

boundary of transaction, so all the object mappings are preserved. Also persist doesn’t return the generated id immediately,

so data persistence happens when needed.

Hibernate **saveOrUpdate** results into insert or update queries based on the provided data. If the data is present in the

database, update query is executed. We can use saveOrUpdate() without transaction also, but again you will face the issues

with mapped objects not getting saved if session is not flushed. For example usage of these methods.

Q- What will happen if we don’t have **no-args constructor in Entity bean**?

Ans - Hibernate uses Reflection API to create instance of Entity beans, usually when you call get() or load() methods.

The method Class.newInstance() is used for this and it requires no-args constructor. So if you won’t have no-args

constructor in entity beans, hibernate will fail to instantiate it and you will get HibernateException.

Q- What is difference between **sorted collection and ordered collection**, which one is better?

Ans- Ordered list is better than sorted list because the actual sorting is done at database level, that is fast and doesn’t

cause memory issues.

Q- Why we should **not make Entity Class final**?

Ans - Hibernate use proxy classes for lazy loading of data, only when it’s needed. This is done by extending the entity bean,

if the entity bean will be final then lazy loading will not be possible, hence low performance.

Q- What is **Named SQL Query?**

Ans - Hibernate provides Named Query that we can define at a central location and use them anywhere in the code. We can

created named queries for both HQL and Native SQL.

Hibernate Named Queries can be defined in Hibernate mapping files or through the use of JPA annotations @NamedQuery

and @NamedNativeQuery.

Hibernate Named Query is global, means once defined it can be used throughout the application.

However one of the major disadvantage of Named query is that it’s hard to debug, because we need to find out the location

where it’s defined.

Q- What is the **benefit of Hibernate Criteria API?**

Ans- Hibernate provides Criteria API that is more object oriented for querying the database and getting results. We can’t

use Criteria to run update or delete queries or any DDL statements. It’s only used to fetch the results from the database

using more object oriented approach.

Criteria API provides Projection that we can use for aggregate functions such as sum(), min(), max() etc.

Criteria API can be used with ProjectionList to fetch selected columns only.

Criteria API can be used for join queries by joining multiple tables, useful methods are createAlias(), setFetchMode()

and setProjection()

Criteria API can be used for fetching results with conditions, useful methods are add() where we can add Restrictions.

**cascade**.ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. Basically everything

\*\*\*\*\*\*\*\*\*\*

**how to add google reCAPTCHA in java**

reCAPTCHA is 2 types -

1. v2 Checkbox(generally we use this)

2. v3 Checkbox

first you need to go 'https://www.google.com/recaptcha/' site and add your domain name

then you will get 2 key

1 **site** key (this key add in login html page then you will get captcha image(i am not robot and match images))

2 **secret** key (this key need to add in 1 class file and verify through api(all code predefine))

\*\*\*\*\*\*\*\*\*\*

Q- Why thread communication **methods wait(), notify() and notifyAll() are in Object class**?

Ans- 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 so that every class in java has these essential methods for inter

thread communication.

Q- Why **wait(), notify() and notifyAll() methods have to be called from synchronized method** or block?

Ans - When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait

state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves

the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require

Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized

method or block.

Q- What is an **Exception in Java**?

An exception is an error event that can happen during the execution of a program and disrupts its normal flow. The exception

can arise from different kinds of situations such as wrong data entered by the user, hardware failure, network connection

failure, etc.

Whenever any error occurs while executing a java statement, an exception object is created, and then JRE tries to find an

exception handler to handle the exception. If a suitable exception handler is found then the exception object is passed to

the handler code to process the exception, known as catching the exception. If no handler is found then the application

throws the exception to the runtime environment and JRE terminates the program.

Java Exception handling framework is used to handle runtime errors only, compile-time errors are not handled by exception

handling framework.

Exception in thread main java.lang.**UnsupportedClassVersionError**: This exception comes when your java class is compiled from

another JDK version and you are trying to run it from another java version.

Exception in thread main java.lang.**NoSuchMethodError**: main: This exception comes when you are trying to run a class that

doesn’t have the main method.

Q How **to serialize an ArrayList** in java?

In Java, the ArrayList class implements a Serializable interface by default i.e., ArrayList is by default serialized. We can

just use the ObjectOutputStream directly to serialize it.

Q How to **synchronize an ArrayList** in java?

Collections.synchronizedList(List<T>) method

Thread safe means that a method or class instance can be used by multiple threads at the same time without any problems

occurring. Where as **Synchronized means** only one thread can operate at single time.

Collections.**unmodifiableSet**(List) -- for read only

Collections.**synchronizedCollection**(List) -- for thread safe

CopyOnWriteArrayList -- use for fail-safe

**Comparable interface** for collection sorting

single object based sorting

compareTo(obj)

Collections.sort(List)

java.lang

natural sorting order.

**Comparator interface** for collection sorting

two object based sorting

compare(obj1, obj2)

Collections.sort(List, Comparator)

java.util

customized sorting order

Q- Can we **create a non-web application in Spring Boot**?

Ans- bty implements CommandLineRunner interface and override run() method

and in pom.xml file use packaging as jar

Q- What is the **default port of tomcat in spring boot**?

Ans port 8080(By default, Spring Boot uses Tomcat 7 version)

Q- How to **disable a specific auto-configuration class**?

Ans- You can use exclude attribute of @EnableAutoConfiguration if you want auto-configuration not to apply to any specific class.

//use of exclude

@**EnableAutoConfiguration**(exclude={className})

@**RestController** - converts the response to JSON or XML(as http response)

A **protocol** is a set of rules that define how communication occurs in a network.

**HTTPS** is HTTP with encryption. The only difference between the two protocols is that HTTPS uses TLS (SSL) to encrypt normal

HTTP requests and responses. As a result, HTTPS is far more secure than HTTP.

**SSL** is a cryptographic protocol that uses explicit connections to establish secure communication between web server and client.

**TLS** is also a cryptographic protocol that provides secure communication between web server and client via implicit connections.

Transport Layer Security (TLS) is the successor protocol to SSL(Secure Sockets Layer).

\*\*\* use **spliterator** for iterating list in java8

List<String> s= new ArrayList<>();

  s.add("one");  s.add("three");  s.add("five");

   s.stream().**spliterator**().forEachRemaining((n) -> System.out.println(n));

Q- What is **Java Persistence API (JPA)**?

Ans- JPA is not a tool or framework; rather, it defines a set of concepts that can be implemented by any tool or framework.

hibernate Session maintains the life cycle of hibernate entity( Persistence , Transient , Detached )

load() is from Jpa where as get() is hibernate specific.

- The loaded entity can be removed from session using evict() method. The next loading of this entity will again make a database

call if it has been removed using evict() method.

- The whole session cache can be removed using clear() method. It will remove all the entities stored in cache.

When you close the hibernate session or call the evict() method then the object moves to the detached state. In this state,

hibernate doesn't track the object but you can re-attach a detached object to Hibernate session by calling the update() or

saveOrUpdate(), or merge() method. Once reattached, the detached object will move to Persistent state.

save() is hibernate methods where as persist() comes from Jpa.

Ques. Why we **should not make Entity Class final**?

Ans - As hibernate internal uses Proxy design patter to create proxy object in some cases like load() method or in lazy loading

concept and proxy design patter is based on inheritance and final class is opposite of inheritance

so we don't make our final in hibernate at all , other wise hibernate will throw an exception

There is no default cascade type in JPA. By default no operations are cascaded.

**JPA** is a specification for accessing, persisting and managing the data between Java objects and the relational database.

Where as, Hibernate is the actual implementation of JPA guidelines.

session.createQuery(hql);  -- as **HQL** query

session.createSQLQuery(hql); --as **native** query

Q- What is **Breadcrumb in website** ?

Ans -  breadcrumb navigation provide links back to each previous page the user navigated through,

and shows the user's current location in a website.

Home / Pictures / Summer / 15 / Italy

Q- What is **Toast in website** ?

Ans - The toast component is like an alert box that is only shown for a couple of seconds when something happens.

@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)  -- **Table per class** (1 table with extra column)

@Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) -- **Table per concrete class** (2 table with duplicate value)

@Inheritance(strategy=InheritanceType.JOINED) -- **Table per sub class** (2 table with using primary and foreign key so no duplicate value)

@DiscriminatorColumn

**instanceOf** operator - is type comparison operator return true or false

**ClassNotFoundException** - is an exception that occurs when you try to load a class at run time using Class.

forName() or loadClass() methods and mentioned classes are not found in the classpath.

**NoClassDefFoundError** - is an error that occurs when a particular class is present at compile time, but was missing at run time.

**UnsupportedClassVersionError** - class compiled with higher version but running on lower version.

Is it possible to **throw a statement/exception** inside a static block - yes we can

ex -  static String path = "D://sample.txt";

   static {

      FileNotFoundException ex = new FileNotFoundException();

      throw ex;

   }

 Q- what is **exception re-throwing in java** ?

Ans - When an exception is cached in a catch block, you can re-throw it using the throw keyword (which is used to throw the exception objects).

try{

}

catch(ArithmeticException e  yaa ArrayIndexOutOfBoundsException  e) {

   throw e yaa throw new IndexOutOfBoundsException();

}

Q- What is **chained exception in java** ?

Ans - Chained exception helps to relate one exception to other.

using initCause() method and using ae.getCause() for knowing ex

ArithmeticException ae = new ArithmeticException(" ");

        ae.initCause(new IOException("cause"));

throw ae;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q- What is **N 1 problem in Hibernate How will you identify and solve it** ?

The N+1 query problem is said to occur when an ORM, like hibernate, executes 1 query to retrieve the parent entity

and N queries to retrieve the child entities. The N+1 selects problem is a performance anti-pattern.

\*\*\* **disadvantages**

- N+1 problems create more queries to database means more db hitting. This means database will be overloaded.

- More queries to the database impacts the performance of the database as well as the application server.

- Longer processing time means more open connections from connection pools so other will in wait condition and deadlock situation can be occur.

- Network load increase due to multiple round trip so increase n/w latency.

\*\*\* - **solution is**

1. The first solution is to **use join fetch**

ex- entityManager.createQuery("select a from Author a left join fetch a.books", Author.class);

2. Another way is to use **@BatchSize** on the lazy association:

public class Author {

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "author")

    @BatchSize(size = 10)

    private Set<Book> books;

}

3. by using **@Fetch(FetchMode.SUBSELECT)** on the lazy association:

public class Author {

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "author")

    @Fetch(FetchMode.SUBSELECT)

    private Set<Book> books;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q- What is **NumberFormatException** in java ?

Ans - String s= "krishna";

Integer i = Integer.parseInt(s);

System.out.println(s+" -- "+i);

o/p- java.lang.NumberFormatException:

**evict()** - will clear particular object from session in hibernate

**clear()-** will clear all objects from session(used to clear the persistance context.)

**flush()** - Flushing the session forces Hibernate to synchronize the in-memory state of the Session

with the database (i.e. to write changes to the database). By default, Hibernate will flush changes automatically

for you: before some query executions. when a transaction is committed.

**Lazy loading** -loads the child objects on demand.

**map()**

The function you pass to the map() operation returns a single value. ex - .map(Integer::valueOf)

map() is used for transformation only.

**flatmap()**

The function you pass to flatMap() operation returns a Stream of value. ex - .flatMap(x -> x.stream())

flatMap() is a combination of map and flat operation.

flatMap() is used for both transformation and flattening.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Agile methodology with scrum framework**

Agile is a **software/project development methodology** to build a software incrementally using short

iterations of 1 to 4 weeks so that the development process is aligned with the changing

business needs. Instead of a single-pass development of 6 to 18 months where all the

requirements and risks are predicted upfront, Agile adopts a process of frequent feedback

where a workable product is delivered after 1 to 4-week iteration using sprint.

Agile **follow 12 rule Principles for higher productvity**.

\*\*\* **Agile Development Cycle**

Planning -> Requirements analysis -> Design -> coding -> Testing -> Deployment

\*\*\* **Advantages of Agile**

- frequent changes so design improved

- Strong team interaction

- fast & high-quality delivery

- customer intreaction high

- Continuous improvement

for implementing agile methodology we use Scrum and Kanban framework for developing, delivering, and sustaining products in a complex environment.

**Scrum** - is a subset of Agile and one of the most popular process frameworks for implementing Agile.

It is an iterative development model.

Fixed-length iterations, called sprints lasting one to two weeks long, allow the team to ship software on a

regular basices. At the end of each sprint, stakeholders and team members meet to plan next steps.

**advatages of scrum**--

- More transparency and project visibility

- Increased team accountability

- Easy to changes

- Increased cost savings

**Roles in Scrum--**

1. Product Owner(client team)

2. Scrum Master(manager, team lead, BA) - to remove any kind of block for their team.

3. Scrum Team(developer,tester,designer)

**Steps in the Scrum Process---**

- Product Planned meetings

- Backlog grooming

- Daily Scrum stand-up meetings (morning)

- Sprint review meeting (evening)

**Tools in Scrum**

1. Scrum board

2. User stories -  required define(what they want in functionality)

3. Timeboxing

Every agile team should be a self-sufficient **team with 5 to 9 team member**s and an average

experience ranging from of 6 to 10 years. Typically, an agile team comprises of 3 to 4

developers, 1 tester, 1 technical lead, 1 product owner and 1 scrum master.

team size - 3 to 9

sprint duration - 1 to 4 week

Q- What do you know about **impediments** in Scrum?

Ans - Impediments are the obstacles or issues faced by scrum team which slow down their speed of work.

Q- What do you understand by **Daily Stand-Up**?

Ans-The daily stand-up is an everyday meeting(most preferably held in the morning) in which the whole team meets for

almost 15 minutes to find answer to the following three questions –

What was done yesterday?

What is your plan for today?

Is there any impediment or block that restricts you from completing your task?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**junit**

The most popular testing framework in Java is Junit5.

JUnit is an open source testing framework which is used to write and run repeatable automated tests, so that we can be ensured

that our code works as expected.

**JUnit provides**:

• Assertions for testing expected results.

**Annotaion is**

@Test

@Before

@BeforeClass

@After

@AfterClass

**ex -**

@Test

public void testSum() {

assertEquals("your input", "expected output");

}

fail - red strap (provide count for errors and failures)

pass - green strap

**list of method**

assertEquals(obj1, obj2);

assertSame(obj3, obj4);

assertNotSame(obj2, obj4);

assertNotNull(obj1);

assertNull(obj5);

assertTrue(var1 var2);

assertArrayEquals(arithmetic1, arithmetic2);

Q- What is  **suite tests in junit** ?

Ans - A test suite is a collection of some test cases from different classes that can

be run all together using @RunWith and @Suite annotations.

Ex -

@RunWith(Suite.class)

@Suite.SuiteClasses({ Class1.class, Class2.class(these class have @test method) })

public class SuitTest {

}

\*\*\*\*\*\*

Q- What is **RandomAccess** **interface** and what is work of it ?

Ans - RandomAccess is a **marker interface** implemented in List inteface that help to provide fast random access search to list interface.

\*\*\*\*\*\*\*\*\*\*\* **Agile Methodology**

Only a Product Owner can cancel the sprint.

Time Boxing is called **sprint**.

The work to be done in a sprint is defined by the **velocity**.

Q- How do **you track your progress** in a sprint?

Ans: The progress is tracked by a **'Sprint Burn-Down chart'**(shows the estimated v/s actual effort of the scrum tasks) .

Q-During Review, suppose the product owner or stakeholder **does not agree to the feature you implemented**, what would you do?

Ans:We will first confirm the actual requirement from the stakeholder and update the user story and put it into the backlog.

Based on the priority, we would be pulling the story in the next sprint.

**Advantages of using agile**-

Deliver Frequently

Continuous Testing

Continuous Feedback/changes

Clean Code

Less Documentation

Work Together

Motivated Team

Customer Satisfaction

**\*\*\*Object class method**

wait()

notify()

notifyAll()

clone()

hashcode()

equals()

toString()

getClass()

finalize() -- **deprecated from jdk 9**

note that **spring boot2 uses HikariCP as the default connection pool**. So if you need to change the connection pool, configure the respective properties.

By default, the spring boot application **as .jar file only**.

and for making war use - <packaging>**war**</packaging>

Spring boot supports default configuration for Java Util Logging, Log4J2, and Logback.

The default logging uses a console logger with a log level set to DEBUG, which we can change in the custom logback.xml file.

**By default Logback is used in spring boot**.

What is **open source?**

Open source simply means the source code that anyone can see, inspect, modify, and enhance. If you want you can also contribute to spring-framework.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*@**JsonManagedReferece** - is a forward reference that includes during the serialization process

@**JsonBackReference** - @JsonBackReference annotation is a backreference that omits during the serialization process.(use over the List<Emp> em(like many ))

used to create a JSON structure in a bidirectional way.

these use because 1TOM and MTM relation(bi-directional way) have **Circular Dependencies** so using we remove thise.

A circular dependency occurs when a bean A depends on another bean B, and the bean B depends on bean A as well:

Bean A → Bean B → Bean A

What is a **Design Pattern**?

Design pattern is a general, reusable **solution to a commonly occurring problem** within a given context in software design.

Note:- If we use both constructor & setter injection, IOC container will use the setter injection. This means setter injection overrides the

constructor injection.

Q- What is the **difference between streams and parallel streams**?

Ans- Parallel streams divide the provided task into many and run them in different threads, utilizing multiple cores of the computer.

On the other hand sequential streams work just like for-loop using a single core.

Parallel streams provide high performance

sequential streams provide low performance.

in sequential streams - content of the list is printed in an ordered sequence.

in Parallel streams - is unordered and the sequence changes every time the program is run.

What is **Stream** ?

Stream represents a sequence of objects from a source, which supports aggregate operations.

using stream you can process data in a declarative way similar to SQL statements.

stream is 2 type 1.sequential streams  2.Parallel streams

\*\*\* **artifactId** - is **project name** in spring boot: ex - my\_spring\_demo

@**SpringBootApplication**=

@ComponentScan - to enable component scanning, all the packages and subpackages will be auto-scanned which are

under the root pkg(so you can decide which one should include or exclude)

so that the web controller classes and other components you create will be automatically discovered and

registered as beans in Spring's Application Context

+

@EnableAutoConfiguration  - to enable auto-configuration of the classes bases on the jars added in classpath(dependency added)

+

@Configuration -for java based configuration classes. This is particularly important if you favor Java-based configuration over XML configuration.

 Q- What is **Stereotype means in spring boot** ?

Ans- Spring doesn't care much about the detail of your class, from Spring perspective your Classes are either Repository ,

Service , and Controller , if it doesn't belong to any of that, then its a Component . Spring just making

simplification of your Classes. Hence, the name Stereotype.

Spring will automatically import the beans into the container and inject to dependencies. These annotations are called Stereotype.

--->>> **server port number change from code**

@SpringBootApplication

public class MyApplication {

  public static void main(String[] args) {

     SpringApplication application = new SpringApplication(MyApplication.class);

     Map<String, Object> map = new HashMap<>();

     map.put("SERVER.PORT", "8585");

     application.setDefaultProperties(map);

     application.run(args);

}}

 18. How to **configure Spring Boot properties**?

Ans: Spring Boot loads application.properties and application.yml from classpath by default. If both files are present

in the classpath then both are loaded and merged into environment. Using these files we configure our properties.

**NoUniqueBeanDefinitionException** exception - for resolving use @Qualifier annotation

java.lang.**IllegalStateException**: Ambiguous mapping. Cannot map 'your controller name' method.

**when 2 controller have same uri mapping**

**jackson-databind** by default. Spring Boot REST gives JSON response by default because it detects jackson-databind in its classpath.

**Spring beans** - are Java objects that are managed by the Spring container.

Spring container is responsible for instantiating, configuring, and assembling the Spring beans.

spring **container is 2 type -**

**BeanFactory** Interface (this is basic container, when use - for standalone app., uses lazy loading) - implementation class is -

             i - XmlBeanFactory

**ApplicationContext** Interface(this is advance container with advance support, when use - for mvc app., uses eager loading) - implementation classes are -

             i- ClassPathXmlApplicationContext

             ii- FileSystemXmlApplicationContext

             iii- AnnotationConfigWebApplicationContext

**'Default methods'** -  are also known as defender methods or virtual extension methods.

//using **anonymous class implementation**

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("inside thread.");

}

}).start();

//**using lambda**

new Thread( ()->{

System.out.println("inn thread");

}).start();

1. Lambda expression provides implementation of functional interface(1 abstract method inside interface).

2. Lambda expression is an anonymous function(name less)

3. Fewer Lines of Code so clean code

4. Enables functional programming with the Java Streams API

**functional interface** should contain only one single method without implementation.

**Spring boot 3 exception** -

1. NoSuchBeanDefinitionException -

ex - @Component

public class BeanA {

    @Autowired

    private BeanB dependency; //BeanB is not defined in the Spring Context

    //...

}

2. **NoUniqueBeanDefinitionException** (use @Qualifier)

3. **UnsatisfiedDependencyException**: (if we remove top of annotation form class

ex- @Repositry   // we remove this annotation then run the application

interface Employee extends jpaRepositry{

}

**Hibernate 3 exception** -

1. saving a null value to a NOT NULL column in the database can raise this error.  **ConstraintViolationException**

2. NonUniqueObjectException  -- if same id data you are saving in single session

3. LazyInitializationException

4. **NonUniqueResultException** -- when query return more then 1 result set from db side

**Stereotype annotations** -  are used to create Spring beans automatically in the application context.

ex - @Component, @Service, @Repository and @Controller.

Q- Is **it required to write @Repository annotations in spring data JPA**?

Ans - no , Spring detects that the predefined JpaRepository.

@**NoRepositoryBean** -  say to Spring do not trying to create an instance of MyRepository interface.

it just acts as indermediate between Repository and the actual repository interfaces

avoid creating repository proxies for interfaces.

     @NoRepositoryBean

Ex-  public interface com.foobar.MyInterface<…,…> extends CrudRepository<…,…> {

void foo();

}

**cglib**- (Code Generation Library) library

The bytecode instrumentation allows manipulating or creating classes after the compilation phase of a program.

Hibernate uses cglib for generation of dynamic proxies

Spring used two types of proxy strategy one is JDK dynamic proxy and other one is CGLIB proxy.

GLIB is a third party library which spring used for creating proxy.

**@Bean** - It is used to explicitly declare a single bean, rather than letting Spring do it automatically.

Bean can be created even class is outside the spring container.

@Bean annotation is used for added spring beans for the class which is defined on external jars(3rd party classes ex-RestTemplate, DataSource etc).

@Bean is a method level annotation.

Q- Can we use @**component and @bean in same class**?

Ans- It's not possible. You get a duplicate exception

**BeanFactoryPostProcessor**(I) -  implementation is used to read the configuration metadata and potentially change

it before beans are instantiated by IOC container

If you want to implement some custom logic after the Spring container finishes instantiating, configuring,

and initializing a bean, we can plug in one or more BeanPostProcessor implementations.

Spring BeanFactoryPostProcessor works on the bean definitions or configuration meta data of the bean

before beans are actually created.

@**Transactional** annotation -, which provides broad support for transaction management and allows developers to

concentrate on business logic rather than worrying about data integrity in the event of system failures.

/\*

Integer num1 = 128, num2 = 128;

      System.out.println(num1 == num2);//false

      Integer num3 = 127, num4 = 127;   // integer max range is 127 otherwise show false

      System.out.println(num3 == num4);//true

\*\*/

\*\***Update and Merge both work on detached instances**

Both methods are used to convert these object into persistence state from detached state.

**merge**() - A merge() method is used to update the database. It will also update the database if the object already exists. ,

no throw exception

**v/s**

**update**() - An update() method only saves the data in the database. If the object already exists, no update is performed.,

throw exception if same PK data available in db

we **evict (detach**) the saved entity from the context, change the name field, and then merge the detached entity:

**persist**() - from jpa, return void after save data, save object within transaction boundaries only

**save**() - from hibernate, return serializable id after save data, save object within boundaries and outside boundaries

Q- How the **dispatcher servlet is set up in a spring boot project** ?

Ans- The Spring Boot autoconfiguration registers and configures the DispatcherServlet automatically. Therefore,

we don't need to register the DispatcherServlet manually

@**RestControllerAdvice**- is the combination of both @ControllerAdvice and @ResponseBody (use in REST web services)

@**ControllerAdvice**- annotation for handling exceptions in the RESTful Services but we need to add @ResponseBody

separately.(an use in both MVC and Rest web services)

**ThreadLocal**- class provides thread-local variables. It enables you to create variables that can only be read and write

by the same thread. If two threads are executing the same code and that code has a reference to a ThreadLocal variable

then the two threads can't see the local variable of each othe.

ThreadLocal is useful, when you want to have some state that should not be shared amongst different threads.

**SecurityContext and SecurityContextHolder** are two fundamental classes of Spring Security.

Spring Security Context working-  is used to store the details of the currently authenticated user, also known as a principle.

So, if you have to get the username or any other user details, you need to get this SecurityContext first. The SecurityContextHolder

is a helper class, which provides access to the security context.

By default, it uses a ThreadLocal object to store security context, which means that the security context is always available

to methods in the same thread of execution.

Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

String username = ((UserDetails)principal).getUsername();

**A major problem with marker interfaces is** that the interface defines a contract for implementing classes, and that contract is

inherited by all subclasses. This means that we cannot “un-implement” a marker in child classes.

In the example given, if you create a subclass of Data that we do not want to serialize (perhaps because it depends on the

transient state), we must resort to explicitly throwing NotSerializableException.

**Making Child classes as Non Serializable** in java if super class implementing Serializable interface

You can not remove the interface in subclass but you can manage by using -

writeObject(ObjectOutputStream) and readObject(ObjectOutputStream) method in the child class and throw NotSerializableException from those methods.

ex- class Child extends Parent {

private void writeObject(ObjectOutputStream oos) throws IOException {

        throw new NotSerializableException();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**shallow copy** - if you do any changes to the data using the copied object, they are reflected in the original object too.

Shallow copy is preferred if an object has only primitive fields.

In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities

Ex- ClientClass ob  = new ClientClass();  // 1 object created

    ClientClass ob2 = (ClientClass)ob.clone(); // 2 object

and implement Cloneable interface in class.

**deep copy** - if perform you deep copy on an object that contains reference (object), both the original and copied object

refers to different objects and, if you do any changes to the data the copied object they are not reflected in the original object.

Deep copy is preferred if an object has references to other objects as fields.

In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities.

Each entity has its own independent reference

in base class use -

protected Object clone() throws CloneNotSupportedException{

      return super.clone();

   }

and implement Cloneable interface.

and in child class -

 protected Object clone() throws CloneNotSupportedException{

      childclass student = (childclass) super.clone();

      student.contact = (baseclass) contact.clone();

      return student;

   }

**What is Stream ?**

Stream represents a sequence of objects from a source, which supports aggregate operations.

using stream you can process data in a declarative way similar to SQL statements.

stream is 2 type 1.sequential streams  2.Parallel streams

**----**

SELECT \* INTO NewTable FROM kaushal\_pragati\_aadhar; 1 table whole data copied into another table

SELECT \* INTO NewTable FROM kaushal\_pragati\_aadhar where 1=2; 1 table structure copied and make new table

\*\*\* if you want targeted column copied then use -

INSERT INTO newTable(col1,col2) SELECT col1,col2 FROM sourceTable ;

select count(\*) from information\_schema.columns where table\_name='your\_table\_name' // know how many column table have

select version(); // know postgresql version

**----**

Object-level locking is a mechanism when you want to synchronize a non-static method or non-static code block such

that only one thread will be able to execute the code block on a given instance of the class. This should always be

done to make instance-level data thread-safe.

Class-level locking prevents multiple threads to enter in synchronized block in any of all available instances on runtime.

This means if in runtime there are 100 instances of a class, then only one thread will be able to execute a method in any

one instance at a time, and all other instances will be locked for other threads. This should always be done to make

static data thread-safe.

Q- Can **We Assign ‘null‘ to ‘this‘ Reference** Variable?

Ans- NO. We can’t.

Similarly, null can not be assigned to ‘super‘.

- Abstraction is more about ‘What‘ a class can do. [Idea]

– Encapsulation is more about ‘How‘ to achieve that functionality. [Implementation]

Q- Why **wait() and notify() are Declared in Object instead of Thread** ?

Ans -In Java, all object has a monitor. Threads wait on monitors so, to perform a wait, we need 2 parameters:

– a Thread

– a monitor (any object)

In the Java design, the thread can not be specified, it is always the current thread running the code.

If wait() and notify() were on the Thread instead then each thread would have to know the status of every other thread.

How would thread1 know that thread2 was waiting for access to a particular resource?

wait- wait method tells the current thread to give up monitor and go to sleep.

notify- Wakes up a single thread that is waiting on this object's monitor.

if do not call wait() and notify() method from synchronize block then throw exception -

**java.lang.IllegalMonitorStateException**

**transient** keyword -  in Java is used to indicate that a field should not be serialized

Q- What is Inversion of Control (**IoC**) and Dependency Injection (**DI**) ?

Ans- IoC can be called design principle and DI is its implementation. In Both cases responsibility to manage

Objects is transferred to spring container(ApplicationContext ac;) instead so inversion of control.

Q- Difference between **BeanFactory and ApplicationContext**?

Ans- ApplicationContext - advance container, use in enterprise application, mvc application

ApplicationContext preloads all the singleton beans upon start-up.

     BeanFactory - basic container, use when memory consumption is critical so use in lightweight applications like mobile devices.

The BeanFactory by default lazy loads the beans, it creates the bean only when the getBean() method is called.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Bean life cycle in spring boot**

Bean Instantiation -> populate properties -> BeanNameAware(I) (setBeanName()) -> BeanFactoryAware(I) (setBeanFactory())

-> pre-initialization (BeanPostProcessor is interface) -> InitializingBean(I) (afterPropertiesSet() )

-> call custom (int-method) -> post-initialization(BeanPostProcessor) ->ready to use

for destroy - > disposable bean(I) (destroy())

Best Practise standpoint Spring Document preference is to use JSR-250 @PostConstruct and @PreDestroy

annotations and then @Bean init-method and destroy-method tapping options.

Managing the bean life cycle in spring boot we can use 3 ways-

1. using xml

2. using custom programmatic way

3. using annotations(this way is using in best practices because spring container available)

int() method on the startup of a spring container  - @PostConstruct annotation

destroy() method on closing the spring container  - @PreDestroy annotation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q- What is **stateful and stateless bean in Spring**?

Ans - singleton type is stateless

      prototype bean is stateful

Q- **Are Singleton beans thread safe in Spring Framework**?

Ans- No

Q- How to make Spring bean thread safe ?

Ans - change form scope("singleton") scopr to scope("prototype") so you can achieve thread-safety

Q- Different modes **of bean autowiring** ?

Ans  5 type-

1. no means the autowiring is OFF.

2. byName

3. byType

4. constructor

5. ‘Autodetect’ has been deprecated.

in spring boot we use - @Autowired annotation to auto-wire bean.

The default mode of the @Autowired is byType.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **exception rule in overriding**

1. If base class doesn’t throw any exception but child class throws an unchecked exception. - no issue

2. If base class doesn’t throw any exception but child class throws an checked exception - compilation problem

3. When base class and child class both throws a checked exception - no issue

4. When child class method is throwing smaller checked exception compared to the same method of base class - compilation problem

5. If SuperClass declares an exception and SubClass declares without exception.(for checked and unchecked both)  -- no issue

**JSONObject** - is an unordered collection of name/value pairs.

JsonObject is immutable so you cannot modify the object.

ex- JSONObject requestJson = new JSONObject();

requestJson.put("key", value);

**ObjectMapper**- is the main class of Jackson library. ObjectMapper class ObjectMapper provides functionality

for reading and writing JSON, either to and from basic POJOs (Plain Old Java Objects), or to and from a general-purpose JSON Tree Model (JsonNode)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Spring boot 3 exception** -

1. NoSuchBeanDefinitionException -

2. NoUniqueBeanDefinitionException

3. UnsatisfiedDependencyException

**Hibernate 3 exception** -

1.ConstraintViolationException

2. NonUniqueObjectException

3. LazyInitializationException

4. NonUniqueResultException

**@Bean** - It is used to explicitly declare a single bean, rather than letting Spring do it automatically.

Bean can be created even class is outside the spring container.

int() method on the startup of a spring container  - @PostConstruct annotation

destroy() method on closing the spring container  - @PreDestroy annotation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q- What is **stateful and stateless bean** in Spring?

Ans - singleton type is stateless

      prototype bean is stateful

Q- Are Singleton beans thread safe in Spring Framework?

Ans- No

Q- How to make Spring bean thread safe ?

Ans - change form scope("singleton") scopr to scope("prototype") so you can achieve thread-safety

Q- Different modes of **bean autowiring** ?

Ans  5 type-

1. no means the autowiring is OFF.

2. byName

3. byType

4. constructor

5. ‘Autodetect’ has been deprecated.

in spring boot we use - @Autowired annotation to auto-wire bean.

The default mode of the @Autowired is byType.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**circular dependency resolve** in spring

spring.main.allow-circular-references = true

@autowired vs @inject

**jdbcTemplate.queryForObject**("sql query",new UserMapper(), "your search id ?") **return** - rowMapper object

->>>>>>>>>> **1st , 2nd and 3rd ... highest salary get using stream api in java**

//Optional<EmpBean> emp = getEmp().stream().sorted(Comparator.comparingDouble(EmpBean::getSalary).reversed()).findFirst();// 1st highest salary

Optional<EmpBean> emp = getEmp().stream().sorted(Comparator.comparingDouble(EmpBean::getSalary).reversed()).skip(1).findFirst();// 2nd highest salary

Internationalization **- i18n**

Localization -   **i10n**

**Apache Tika -**  is a library that is used for document type detection and content extraction from various file formats.

eg. (such as PPT, XLS, PDF, mage, Mp3, Mp4)

use to detect and extract data from various file formats.Tika - is a content analysis tool, designed and developed by Apache Software Foundation. It is written in Java and used to detect and extract content and metadata from the file.

**// odd-even number print together**

Map<Boolean, List<Integer>> r = list.stream().collect(Collectors.partitioningBy(n -> n % 2 == 0));

System.out.println(r.get(true) +"\n"+r.get(false));

**// print those value start with 1(some specific value)**

List<Integer> myList = Arrays.asList(10,15,8,49,25,98,32);

myList.stream() .filter(s -> String.valueOf(s).startsWith("1")) .forEach(System.out::println);

**Web server**- accepts and fulfills requests from clients for static content (i.e., HTML pages, files, images, and videos)

from a website. Web servers handle HTTP requests and responses only.

web server is meant to serve static pages e.g. HTML and CSS.

eg. nginx , apache server

**Application server-** exposes business logic to the clients, which generates dynamic content.

Application Server is responsible for generating dynamic content by executing server side code e.g. JSP, Servlet or EJB.

eg. Apache tomcat, glassfish,Jboss

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **How to add h2 embedded database**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.path=/h2-console

spring.h2.console.enabled=true

and add dependency in **pom.xml** file

how to **open console** on chrome browser  - http://localhost:8080/h2-console

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Predefined-Functional Interfaces in jdk8**

1. **Predicate**<T> - boolean test(T t);

2. **Consumer**<T> - void accept(T t)

3. **Supplier**<R> - R get()

4. **Function**<T,R> - R apply(T t)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Remove tomcat from spring boot application***

Add in pom.xml

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-web</artifactId>

    <exclusions>

        <exclusion>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-tomcat</artifactId>

        </exclusion>

    </exclusions>

</dependency>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*