**TCL Interview Questions & Answers**

1) **Abstract class and interface.**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

**2) final** ->can be used for all levels .:-

A) The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

3) **polymoriphism**

A) Polymorphism in Java is **a concept by which we can perform a single action in different ways**. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java:

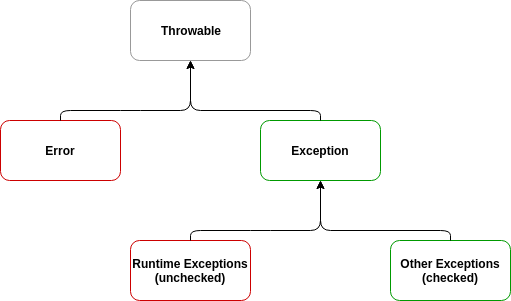
1. **Compile-time/static/overloading polymorphism**

* Overloading occurs when two or more methods in one class have the same method name but different parameters
* Static, final, private, protected can be overload
* Real time – overloading reset password method in our class, overloading the calculate premium method

1. **Runtime/dynamic/overriding polymorphism**

* Overriding occurs when two methods have the same method name and parameters.
* Static, final, private can’t be override. But protected can override
* Real time – overriding createHttpRequest method in our class by extending from rest template, overrding the accountStatus method

4) **Exception** :



1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, FileNotFoundException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

5) **Stream**

A) Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

Two operation on Streams

1)Intermediate Operation

a) map

b) filter

c) sorted

2) Terminal Operation

a) collect

b) forEach

c)reduce

6) **Comparable vs Comparator**

A) **Comparable should be used when you compare instances of the same class**. Comparator can be used to compare instances of different classes. Comparable is implemented by the class which needs to define a natural ordering for its objects.

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

6) **what kind of exception Stream API will throw?**

1. Stream API will throw Runtime Exception

Ex:

List<List<String>> fileContents = paths.stream().map(path -> Paths.get(path))

.map(path -> {

**try** {

**return** Files.readAllLines(path);

} **catch** (IOException e) {

**throw** **new** **RuntimeException**(e);

}

}).collect(Collectors.toList());

7) **HashSet** -> experience .

A) Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

8) **class file** ->assuming your the program : JVM ->class loader

A) First whatever java code we have written in IDE/ notepad first JVM get our java file along with all necessary library files and then compile it using compiler then it produce .class file in the form of bytecode where the class file can use in any machine whenever we want to run the program the class file will convert to binary code using interpreter

Compiler 🡪 scans all the program at a time to convert it to machine code takes lot of time  
interpreter 🡪 scans line by line of the program to convert it to machine code takes less time

10) **multiple inheritance**

A) multiple inheritance does not support in java. Because Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with the same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

11) **Array list**

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We cannot create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:

1. ArrayList<**int**> al = ArrayList<**int**>(); // does not work
2. ArrayList<Integer> al = **new** ArrayList<Integer>(); // works fine

* Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

12) **functional interface in java 1.8**

A) A **functional interface** is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. ***Runnable***, ***ActionListener***,***Comparable*** are some of the examples of functional interfaces.

Functional Interface is additionally recognized as **Single Abstract Method Interfaces**. In short, they are also known as **SAM interfaces**.

13) one custom class you can extend the array list and actually it should not allow duplicates?

14) **final class**

A) The main purpose of using a class being declared as final is to prevent the class from being subclassed. If a class is marked as final then no class can inherit any feature from the final class.

You cannot extend a final class. If you try it gives you a compile time error.

15) **hash table and has map**, can we have a null value for both key and value? one null keys,

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

16) **socket program [server.socket, port no,]**

**A)**

17) **TCP [Reliable protocol ],UDP**

**A)** Transmission Control Protocol (**TCP**) is connection-oriented, meaning once a connection has been established, data can be transmitted in two directions. TCP has built-in systems to check for errors and to guarantee data will be delivered in the order it was sent, making it the perfect protocol for transferring information like still images, data files, and web pages.

But while TCP is instinctively reliable, its feedback mechanisms also result in a larger overhead, translating to greater use of the available bandwidth on your network.

User Datagram Protocol (**UDP**) is a simpler, connectionless Internet protocol wherein error-checking and recovery services are not required. With UDP, there is no overhead for opening a connection, maintaining a connection, or terminating a connection; data is continuously sent to the recipient, whether or not they receive it.

Although UDP isn’t ideal for sending an email, viewing a webpage, or downloading a file, it is largely preferred for real-time communications like broadcast or multitask network transmission.

|  |  |  |
| --- | --- | --- |
| **Feature** | **TCP** | **UDP** |
| **Connection status** | Requires an established connection to transmit data (connection should be closed once transmission is complete) | Connectionless protocol with no requirements for opening, maintaining, or terminating a connection |
| **Data sequencing** | Able to sequence | Unable to sequence |
| **Guaranteed delivery** | Can guarantee delivery of data to the destination router | Cannot guarantee delivery of data to the destination |
| **Retransmission of data** | Retransmission of lost packets is possible | No retransmission of lost packets |
| **Error checking** | Extensive error checking and acknowledgment of data | Basic error checking mechanism using checksums |
| **Method of transfer** | Data is read as a byte stream; messages are transmitted to segment boundaries | UDP packets with defined boundaries; sent individually and checked for integrity on arrival |
| **Speed** | Slower than UDP | Faster than TCP |
| **Broadcasting** | Does not support Broadcasting | Does support Broadcasting |
| **Optimal use** | Used by HTTPS, HTTP, SMTP, POP, FTP, etc | Video conferencing, streaming, DNS, VoIP, etc |

18) **plain JDBC.connection?**

A) Steps:

1. Import the database
2. Load the drivers using the *forName() method*
3. Register the drivers *using DriverManager*
4. Establish a connection*using the Connection class object*
5. Create a statement
6. Execute the query
7. Close the connections

19)**what is marker interface ? what is the use of it?**

A) An empty interface in Java is known as a marker interface i.e. it does not contain any methods or fields by implementing these interfaces a class will exhibit a special behavior with respect to the interface implemented. **java.lang.Cloneable** and **java.io.Serializable** are examples of marker interfaces.

## **Uses of Marker Interface**

Marker interface is used as a tag that inform the Java compiler by a message so that it can add some special behavior to the class implementing it. Java marker interface are useful if we have information about the class and that information never changes, in such cases, we use marker interface represent to represent the same. Implementing an empty interface tells the compiler to do some operations.It is used to logically divide the code and a good way to categorize code. It is more useful for developing API and in frameworks like Spring.

20) **what is race condition in java**?

A) A condition in which the critical section (a part of the program where shared memory is accessed) is concurrently executed by two or more threads. It leads to incorrect behavior of a program. Because the same resource may be accessed by multiple threads at the same time and may change the data. We can say that race condition is a **concurrency bug**. It is closely related to **deadlock in Java**. We can resolve the **race condition in Java by using synchronized block**.

21)**Thread and process? any specific difference?**

A)

| SNO | Process | Thread |
| --- | --- | --- |
| 1. | Process means any program is in execution. | Thread means segment of a process. |
| 2. | Process takes more time to terminate. | Thread takes less time to terminate. |
| 3. | It takes more time for creation. | It takes less time for creation. |
| 4. | It also takes more time for context switching. | It takes less time for context switching. |
| 5. | Process is less efficient in term of communication. | Thread is more efficient in term of communication. |
| 6. | Multi programming holds the concepts of multi process. | We don’t need multi programs in action for multiple threads because a single process consists of multiple threads. |
| 7. | Process is isolated. | Threads share memory. |
| 8. | Process is called heavy weight process. | A Thread is lightweight as each thread in a process shares code, data and resources. |
| 9. | Process switching uses interface in operating system. | Thread switching does not require to call a operating system and cause an interrupt to the kernel. |
| 10. | If one process is blocked then it will not effect the execution of other process | Second thread in the same task could not run, while one server thread is blocked. |
| 11. | Process has its own Process Control Block, Stack and Address Space. | Thread has Parents’ PCB, its own Thread Control Block and Stack and common Address space. |
| 12. | If one process is blocked, then no other process can execute until the first process is unblocked. | While one thread is blocked and waiting, a second thread in the same task can run. |
| 13. | Changes to the parent process does not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process. |

22) **can we use string in switch case in java? advantages of using switch?**

A) **It is recommended to use String values in a switch statement if the data you are dealing with is also Strings**. The expression in the switch cases must not be null else, a NullPointerException is thrown (Run-time). Comparison of Strings in switch statement is case sensitive

**It allows the best-optimized implementation for faster code execution than the “if-else if” statement**

23) **JDBC**

A) JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and
* Thin Driver

24) **corejava1.7,**

**A) Features**

* Strings in switch statement.
* Binary integer literals.
* Allowing underscores in numeric literals.
* Catching multiple exception types and rethrowing exceptions with improved type checking.
* Automatic resource management in try-statement.
* Improved type inference for generic instance creation, aka the diamond operator <>.

25)JDBC program ->JPA

Spring Out of 10

| **Annotation** | **Meaning** |
| --- | --- |
| @Component | generic stereotype for any Spring-managed component |
| @Repository | stereotype for persistence layer |
| @Service | stereotype for service layer |
| @Controller | stereotype for presentation layer (spring-mvc) |

**26) how will you build the spring boot application ?**

**Step 1: Go to Spring Initializr**

Fill in the details as per the requirements. For this application:

Project: Maven

Language: Java

Spring Boot: 2.2.8

Packaging: JAR

Java: 8

Dependencies: Spring Web

**Step 2:** Click on Generate which will download the starter project

**Step 3:** Extract the zip file. Now open a suitable IDE and then go to File > New > Project from existing sources > Spring-boot-app and select pom.xml. Click on import changes on prompt and wait for the project to sync as pictorially

27) **@Rest controller and @controller annotation** ?

Contoller   
RestContoller 🡪 Controller + Response Body

28) **Post man applicaiton ,** POST

29) Json request body , to get the request body [Pay load ]

30) **Default method in JPA.**

A) CrudRepository and PagingAndSortingRepository offer default methods such as: findAll, findAllById, findById, deleteAll, deleteById, save, saveAll

31) How will create the entity name in class ?

32) final and static variables ? what will be happen ?

A) **The static keyword means the value is the same for every instance of the class.** **The final keyword means once the variable is assigned a value it can never be changed**.

33) What is purpose of finally block?

A) **Java finally block** is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not. The finally block follows the try-catch block.

34) Throw and Throws differnece?

The Java throw keyword is used to throw an exception explicitly. throws is a keyword in Java which is **used in the signature of method to indicate that this method might throw one of the listed type exceptions**

35) how to write API to restrict with JSON[

36) how will define the URL in component class [

37) how will configure the consumer and procdure in spring boot.[Message queue,RabbitMQ]

38) Annonation used in spring .

39) Spring boot how will you wirte the controller classes and annotation .

40)how will you set the media type .[

where you will mediatype ->

default type by mediatype-> where you will be passing the parameter.

41)what is dependecny injection?

A) Dependency Injection is **a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”**. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

42) how will enfroe one mediatype->

43) what is bean scope in springboot? how long the bean will be privlining. [

| **Scope** | **Description** |
| --- | --- |
| [singleton](https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-singleton) | Scopes a single bean definition to a single object instance per Spring IoC container. |
| [prototype](https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-prototype) | Scopes a single bean definition to any number of object instances. |
| [request](https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-request) | Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext. |
| [session](https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-global-session) | Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext. |
| [global session](https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch04s04.html#beans-factory-scopes-global-session) | Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext. |

A) The scope of a bean defines the life cycle and visibility of that bean in

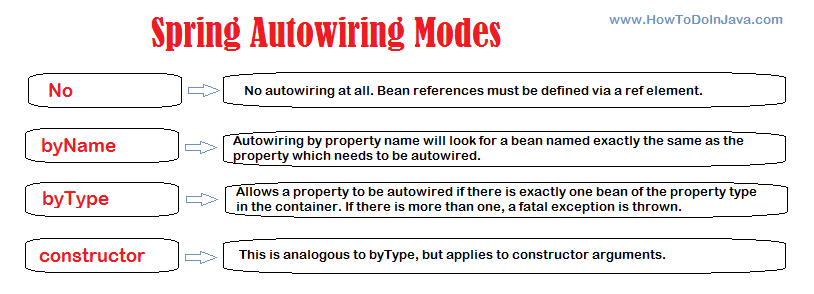
the contexts we use it.

44) default bean scope ?

A) Singleton

45) What is autowired annotation and types? by name,class

A) Autowiring feature of spring framework **enables you to inject the object dependency implicitly**. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values.



46) how to call one spring bean to other bean class ?

47) how to convert singleton to prototype? how will you do that @scope...

48) autowire to by type to byname. @Qualifier

49) when are you starting and what are the layers?

50) custom property in spring boot?

JPA

entity class...

Cascade [parent and child relationship]

@table annotation

[@table,

if table name is not given what will be issue, ] entity,

JPA repostiory

select \* from employee WHERE name LIKE 'Da%'

What is the Java Persistence API

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

What are the steps to persist an entity object?

1. Creating an entity manager factory object. The EntityManagerFactory interface present in java. ...
2. Obtaining an entity manager from factory. ...
3. Intializing an entity manager. ...
4. Persisting a data into relational database. ...
5. Closing the transaction. ...
6. Releasing the factory resources.

What are the steps to insert an entity?

What are the steps to find an entity?

What are the steps to update an entity

What are the steps to delete an entity

What are the different types of entity mapping

A) one to one, one to many, many to many mapping

What are the different directions of entity mapping?

### A) Unidirectional, Bidirectional, Orphan Removal in, Cascade Operations and Relationship & Queries and Relationship Direction

### Persist: is similar to save (with transaction) and it adds the entity object to the persistent context, so any further changes are tracked. If the object properties are changed before the transaction is committed or session is flushed, it will also be saved into database.

Java :

1. Internal working of HashMap , hash set and Tree map

2. Collections API

3. Stream API and methods inside stream .

4. Multi-threading and Thread Lifecycle.

Thread Lifecycle   
New  
Active 🡪 Runnable, Running  
Blocked/waiting

Timed waiting

Terminated

5. Constructor chaining

6. String memory allocation

7. Singleton pattern 🡪 In software engineering, the singleton pattern is **a software design pattern that restricts the instantiation of a class to one "single" instance**. This is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of a singleton. t is used **where only a single instance of a class is required to control the action throughout the execution**. A singleton class shouldn't have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections

8. Serializable and de-serializable

Deserialization is **the reverse process where the byte stream is used to recreate the actual Java object in memory**. Serialization is **the conversion of the state of an object into a byte stream**

9. Predicate in Java

10. How do write customised hash code

11. What happens if we still get exception in finally block

12. Customised exceptional handling

13. **Unmodifiable** list🡪 The **unmodifiableList()** method of **Java Collections** class is used to get an unmodifiable view of the specified list. If any attempt occurs to modify the returned list whether direct or via its iterator, results in an UnsupportedOperationException.

14. **copyonwritearray** list

| **ArrayList** | **CopyOnWriteArrayList** |
| --- | --- |
| ArrayList is not synchronized. | CopyOnWriteArrayList is synchronized. |
| ArrayList is not thread safe. | CopyOnWriteArrayList is thread safe. |
| ArrayList iterator is fail-fast and ArrayList throws ConcurrentModificationException if concurrent modification happens during iteration. | CopyOnWriteArrayList is fail-safe and it will never throw ConcurrentModificationException during iteration. The reason behind the it that CopyOnWriteArrayList creates a new arraylist every time it is modified. |
| ArrayList iterator supports removal of element during iteration. | CopyOnWriteArrayList.remove() method throws exception if elements are tried to be removed during iteration. |
| ArrayList is faster. | CopyOnWriteArrayList is slower than ArrayList. |

14. How to create immutable classes in java 🡪create class as final

15. Types of interfaces

16. Oops concepts

Spring :

1. Bean lifecycle

Bean life cycle is **managed by the spring container**. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.

2. Dependency injection

3. Scope of bean

4. IOC,AOP

The IoC container is responsible to instantiate, configure and assemble the objects. The main tasks performed by IoC container are:

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

**Aspect Oriented Programming** (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

Spring Boot:

1. @springbootapplication,@Autowired,@Component,@Service,@Entity,@RestController,@RabbitMqListner,

@Consumes, @produces, @Swagger, @Configurations, @PostConstruct

| **Annotation** | **Meaning** |
| --- | --- |
| @Component | generic stereotype for any Spring-managed component |
| @Repository | stereotype for persistence layer |
| @Service | stereotype for service layer |
| @Controller | stereotype for presentation layer (spring-mvc) |

2. Keys for database, RabbitMQ in application.properties to connect.

3. Flow of spring MVC



1. Security for application-Oauth/bearer token

The most common way of accessing OAuth 2.0 APIs is using a “Bearer Token”. This is **a single string which acts as the authentication of the API request, sent in an HTTP “Authorization” header**.

5. Retrofithttp package

6. How to run spring boot application using runtime arguments and how to configure in your project.

7. HTTP Client and Rest template

JPA :

1. @Table, @Column, @NamedQuery, @JoinColumn, @manytoone, @onetomany , @Temporal

2. @Repository

3. JPA Queries

4. Native Queries

5. Joins

By using @RequestBody annotation you will get your values mapped with the model you created in your system for handling any specific call. While by using @ResponseBody you can send anything back to the place from where the request was generated.

You can find examples for writing OAuth clients here:

* <https://github.com/spring-projects/spring-security-oauth>

**// If optional is non-empty, get the value in stream, otherwise return empty**

**List<String>** filteredListJava8 = list.stream().**flatMap**(o -> o.isPresent() ? Stream.of(o.get()) : Stream.empty()).**collect**(Collectors.toList());

**// Optional::stream method can return a stream of either one or zero element if data is present or not.**

**List<String>** filteredListJava9 = list.stream()

            .**flatMap**(**Optional::stream**)

            .**collect**(Collectors.toList());

      System.out.println(filteredListJava8);

      System.out.println(filteredListJava9);

1. **Note:** mvn install -DskipTests=true 🡪 to disable the test case execution

hUsing RestTemplate override createRequest(uri,httpmethod) method where we added all details in header like client id, client secret & source system id. Using OAuth2RestTemplate override createRequest(uri,httpmethod) method where we added all details in header like client id, client secret & source system id

POST is always for creating a resource ( does not matter if it was duplicated ) PUT is for checking if resource exists then update, else create new resource. PATCH is always for updating a resource.

restTemplate.exchange("

http://localhost:8080/products", HttpMethod.GET, entity, String.class).getBody();

or

restTemplate.exchange(RequestEntity.get(URI.create(“http://localhost:8080/products”).header(HttpHeaders.CONTENT\_TYPE,MediaType.APPLICATION\_JSON\_VALUE).build(),Java.class)

restTemplate.exchange(

"http://localhost:8080/products", HttpMethod.POST, entity, String.class).getBody();

Or

restTemplate.exchange(RequestEntity.post(URI.create(“http://localhost:8080/products”).header(HttpHeaders.CONTENT\_TYPE,MediaType.APPLICATION\_JSON\_VALUE).body(java),Java.class)

restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.PUT, entity, String.class).getBody();

Or

restTemplate.exchange(RequestEntity.put(URI.create(“http://localhost:8080/products”).header(HttpHeaders.CONTENT\_TYPE,MediaType.APPLICATION\_JSON\_VALUE).body(requestbody),Map.class)

restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.PATCH, entity, String.class).getBody();

Or

restTemplate.exchange(RequestEntity.patch(URI.create(“http://localhost:8080/products”).header(HttpHeaders.CONTENT\_TYPE,MediaType.APPLICATION\_JSON\_VALUE).body(java),java.class)

restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.DELETE, entity, String.class).getBody();

Or

restTemplate.exchange(RequestEntity.delete(URI.create(“http://localhost:8080/products”).header(HttpHeaders.CONTENT\_TYPE,MediaType.APPLICATION\_JSON\_VALUE).build(),Map.class)

**SQL**

UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;

SELECT FullName

FROM EmployeeDetails

WHERE EmpId IN

(SELECT EmpId FROM EmployeeSalary

WHERE Salary BETWEEN 5000 AND 10000);

SELECT \*  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;