**fail fast and fail safe iterator in java ?**

**ConcurrentModificationException :**

If a thread is iterating over the colletion meanwhile same thread or other thread trying to modify the collection data(either adding the element or by deleting the element or by updating the value at particular position in the collection) is known as ConcurrentModification.

**FailFast**:

If a thread is iterating over the colletion meanwhile SAME THREAD or OTHER THREAD trying to modify the collection data(either adding element or by deleting the element or by updating the value at particular position in the collection) then ITERATOR WILL THROW CONCURRENTMODIFICATIONEXCEPTION. Original Collection data is used while iteration.

**Example** : ArrayList, Vector, HashMap

**why it is throwing ConcurrentModificationException ?**

when we take iterator() reference that time it is going to maintain the expectedModcount(modcount may be the no of items added in the HashMap). that modcount gets modified when another thread tries to add another item. when latestModcount and expectedModCount does not match it will throw ConcurrentModificationException.

**FailSafe**:

In failsafe iterator will not throw CONCURRENTMODIFICATIONEXCEPTION. whatever data we are reading from is not from original collection data.Here iterator is not reading from original data it is reading from dummy data.

**Example** : CopyOnWriteArrayList, ConcurrentHashMap

If we use ConcurrentHashMap, we can modify the HashMap while iterating over it because when we take iterator() reference that time it is not going to maintain expectedModCount and it's not iterate original collection object. it going to interate copy of the object.

**HasMap(jdk-1.5) vs HashTable vs SynchronizedHashMap vs ConcurrentHashMap :**

https://1.bp.blogspot.com/-E6GG1fx8xmw/YEpvdGfNJ\_I/AAAAAAAABhQ/5gY0qRL1CR0t5daBOuhC4jma4\_R7yD8QgCLcBGAsYHQ/s956/Hashmap.png

[Hashmap.png (956×423) (bp.blogspot.com)](https://1.bp.blogspot.com/-E6GG1fx8xmw/YEpvdGfNJ_I/AAAAAAAABhQ/5gY0qRL1CR0t5daBOuhC4jma4_R7yD8QgCLcBGAsYHQ/s956/Hashmap.png)

Bydefault java Hashmap is not synchronized.

It means at a time mulitple threads can attack and can read/get/write from hashmap object.

HashTable and SynchronizedHashMap lock is applied at OBJECT LEVEL.

Hashtable(jdk-1.2) is synchronized . It means if one thread is working on hashmap obj. it will not allow other threads to perform any task as it has a lock on it

**SynchronizedMap** is is synchronized . It means if one thread is working on hashmap obj. it will not allow other threads to perform any task as it has a lock on it

**the difference between Hashtable and synchronizedMap is**

Hashtable not allow NULL KEY and NULL VALUES

SynchronizedHashMap allows NULL KEY and multiple NULL VALUES

**ConcurrentHashMap** is synchronized. but the lock is applied on the SEGMENT LEVEL.

segment means In hashmap we have 16 segments(0-15).

If we perform write operation, then one thread perform at one segment and another thread perform at another segment

If we perform read operation, then multiple threads can access same segment and read the values.

NULL KEY and NULL VALUES are not allowed.

Diagram

Description automatically generated

**Comparable Vs Comparator :**

comparabe object has two limitations:

In comparable we mix sorting logics & business logics in single class and one property sorting possible

In comparator we can seperate the logics and we can do the multiple property soritng is available.

Comparable provides default sorting order. we can use comparable for customized soritng order but only one property sorting is available.

To overcome this we have comparator interface. Here we can do sorting based on multiple properties.

we go for comparable when we need default sorting order. For example : string and wrapper classes are implementing comparable interface.

we go for comparator when we need customized sorting order

**Comparable** provides compareTo() method to sort elements whereas Comparator provides compare() method to sort elements in Java.

compatable -- Collections.sort(al)

comparator -- Collections.sort(al,new Eid\_Comparator)

Comparable interface is present in java.lang package whereas Comparator interface is present in java.util package.

Comparable affects the original class whereas comparator doesn’t affect the original class.

**Advantages of using microservices over monolitic applications ?**

In case of Must re-build and re-deploy in case of single module modification.

Microservices makes app independent, no need to re-build all modules/services.

if one module is failed that may cause other modules even problem.

Microservices makes app independent, effect rate will be very less.

Downtime/Deployment time will be more incase of App is verybig.

All are paralllel deployable components

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**explain about equals() and Hashcode() ?**

*For suppose in HashMap object, hashcode method is used to calculate the hashcode of an object and it defines where the object is stored. It tells in which bucket it need to be stored and equals method is used to compare the two objects for duplicate elimination*

To compare the collection object, we use equals and Hashcode methods. If we don't want two employee objects or duplicate employee object inside collection

object like arraylist or Hashcode. we can override equals and hashcode. Sometimes two employees can have the same hashcode so at that time compiler will

check for duplicate with the help of equals() method

**Externalization and serialization**

**Layers in Spring Boot :**

There are 5 Layers.

Client Tier

Presentation Tier

Business Tier

Integration Tier

Resource Tier

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Difference b/w arrayList and LinkedList? when to use arraylist over linkedList ?**

add,remove fetch,search

ArrayList implemeted with the concept of Dynamic array and linked list is implemented with the concept of double linkedlist.

add and remove element is very expensive and time consuming as if we add an element at an index 4. the element at index 4 and

other elements after that index should move one element next. moving all items one index next will consume time and it gives O(n) notation

In case of linked list, it has two nodes, one is prev and other is next while adding element in between, we can simple change next and prev node.

It will not affect all the elements in the list.

while searching we have to use index number. If we have index number it is easy to search. So for seaching and fetching ArrayList is fast.

LinkedList is slow.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Internal working of Hashmap ?**

when we put any key in Hasmap. first it calculate the value of hashcode.

It has buckets. Bydefault it contains 16 buckets.

Based on the hashcode it finds the bucket. Each bucket is a node it contains four elements pre,next,key,value.

It stored into the bucket.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**why string is immutable ?**

string used string constant pool to store it value. If we create string1 with value x, then string2 with value x. then compiler will check for the value

x in string constant pool. It it exists. string2 will point to the memory where string1 is pointing. We can have a lot strings like this pointing to

the same value. If we change the value for one string. that change will impact all the strings. So it was immutable by default.

String is immutable as it is final class.

**Difference between Heap and stack memory ?**

when we create object with new keyword, it will store in the heap memory. Method level local variable store in the stack memory. when method

execution completed stack memory content also destroyed. Instance variable stored in Heap memory.

**what is an optional class ?**

There may be a chance of getting NullPointerException. By using optional class we can avoid NullPointerException. while fetching object from optional class. we can check whether the element is exist or not. If it is there we call the method.

**why lambda ?**

to enable the functional interface. to consice the code.

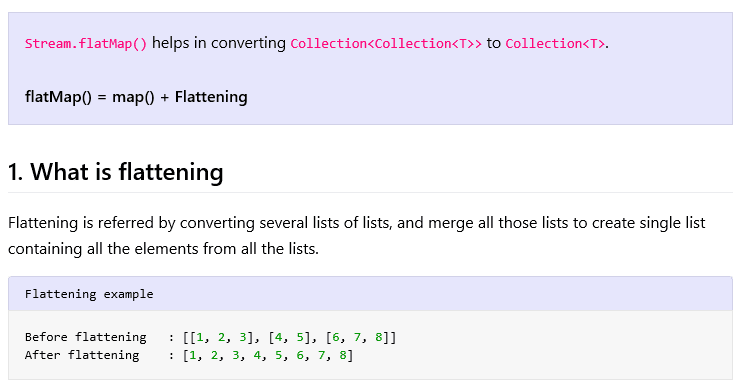
**Difference b/w Map and Flatmap ?**

[java8/mapVsflatMap at master · Java-Techie-jt/java8 (github.com)](https://github.com/Java-Techie-jt/java8/tree/master/mapVsflatMap)

map method in stream api is used for data transformation.

Flatmap is used for data transformation and data flattering.

Converting stream of stream into single stream is called data *flattering*.



**Map vs Filter ?**

Map takes function as parameter. Filter takes predicate as parameter.

Map is used for data transformation based on the statement we provide.

Filter is used for filtering the data based on the condition Boolean value return.

**Difference b/w sleep and wait ?**

Sleep is a static method that can have a particular time. But a thread is in wait state it can come out from that state using notify or notifyAll methods only.

**Can we start a thread twice in Java?**

**no**, once a thread is started, it can never be started again. Doing so will throw an IllegalThreadStateException.

**What is deadlock in java?**

Deadlock is a situation where multiple threads are waiting to acquire lock which is acquired by another thread.

**How to avoid deadlock situation in java?**

1. ***Don't use locks.***
2. If you must, keep your locks local. Global locks can be really tricky.
3. Do as little as possible when you hold the lock.
4. ***Use***[***stripes***](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/util/concurrent/Striped.html)***to only lock segments of your data***
5. Prefer Immutable types. Many times this means copying data instead of sharing data.
6. Use compare and set (CAS) mechanics instead, See [AtomicReference](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/atomic/AtomicReference.html) for example.

**Runnable in java ?**

Runnable is used in multithreading concept. We can create thread either by implementing runnable or by extending thread class. But the difference is if we want to extend another class. We cannot do it while extending thread class. But we can do it while implementing Runnable interface.

**Contact b/w equals and hashmap:**

When we put key and value pair. First it will create Hashcode and the key and value is placed based on the hashcode.

Equals method checks the equality of the object. If two objects are same based on the equals method return type. Object will be placed.

**Class Loaders:**

**Classloaders are 3 types:**

Bootstrap class loader

Extension class loader

Application/System class loader

**Bootstrap class loader :**

It is a ***primordial*** class loader.  It loads standard JDK class files from **rt.jar** and other core classes. It is a parent of all class loaders

**Primordial class loader** is the**first class loader responsible to load class loader like "BootStrap class loader"** which is part of JVM and written in native language.

**Extension class loader:**

This class loader is parent of Application class loader and child class loader of Bootstrap class loader. It loads the extensions of core java classes from the respective JDK Extension library. It loads classes from **jre/lib/ext directory** or any other directory as **java.ext.dirs**.

**System class loader:**

It is child class loader of Extension class loader. It loads application specific classes from the CLASSPATH environment variable.

**Difference b/w arraylist and vector ?**

|  |  |
| --- | --- |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| 2) ArrayList **increments 50%** of current array size if the number of elements exceeds from its capacity. | Vector **increments 100%** means doubles the array size if the total number of elements exceeds than its capacity. |
| 3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object. |
| 5) ArrayList uses the **Iterator** interface to traverse the elements. | A Vector can use the **Iterator** interface or **Enumeration** interface to traverse the elements. |

**Difference b/w Enumeration vs Iterator vs ListIterator :**

Iterator is used to traverse any collection irrespective of the type.

Iterator allows us to remove elements from collection during traversal but Enumeration doesn’t allow that.

Deletion is not allowed incase of Iterator.

In iterator we can’t traverse back.

ListIterator is used to iterate only List collection implemented classes like arrayList,linkedList etc.,

ListIterator can replace or delete an element in list collection.

In listIterator we can traverse back using previousIndex().

**Singleton class:**

It is a class that can be instantiated only once.

This pattern can be useful when managing shared resources.

**public** **class** SingletonClass {

**private** **static** SingletonClass *instance* = **null**;

**public** String str;

**private** SingletonClass() {

}

**public** **static** SingletonClass getInstance() {

**if** (*instance* == **null**) {

*instance* = **new** SingletonClass();

}

**return** *instance*;

}

}

**Factory Pattern: -- creational DP**

[Design Pattern - Factory Pattern (tutorialspoint.com)](https://www.tutorialspoint.com/design_pattern/factory_pattern.htm)

We can create object without exposing creation logic to the client.

If we have an interface. That interface is implemented by three classes. Then we want to create object without exposing object creation logic. Then we can write a get method in a class that can take a string. Based on the string we can create object and return the interface name.

**Strategy Pattern: -- Behavioral DP**

[Design Patterns - Strategy Pattern (tutorialspoint.com)](https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm)

In strategy pattern, we can create objects that represent various strategies and the behavior of the class object changes based on the object we pass as a constructor (while creating the object).

If we want to select object creation at runtime. We will go with Strategy pattern.

If we have an interface. That interface is implemented by three classes. ***Then we want to select any one of the class at run-time then we go with this pattern***. We will write a class that can be created by taking an object as pattern. Then based on the class we can do operations.

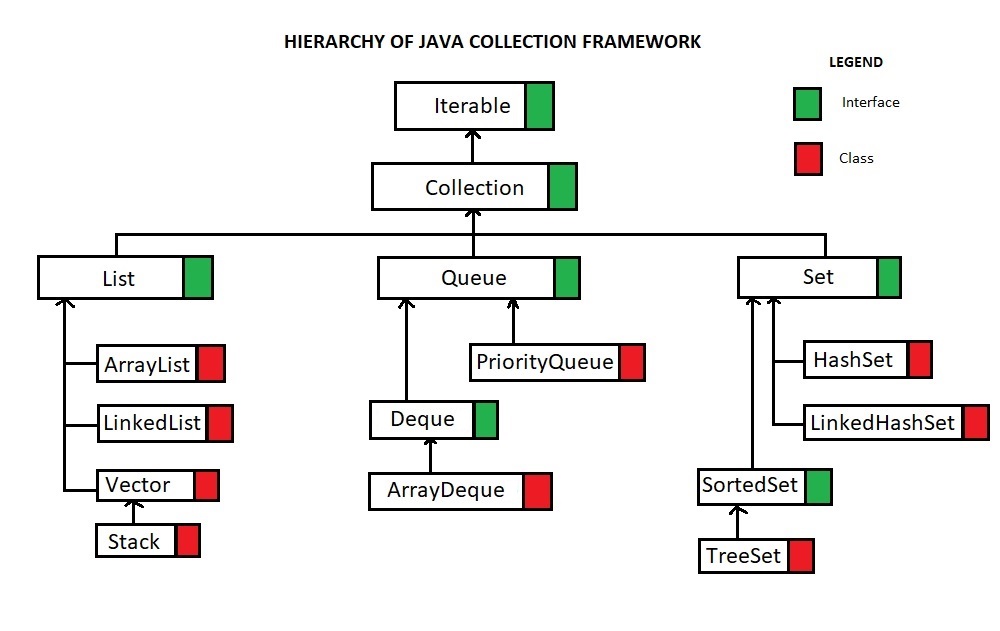
**Adapter Pattern: -- Strutchral DP**

[Design Patterns - Adapter Pattern (tutorialspoint.com)](https://www.tutorialspoint.com/design_pattern/adapter_pattern.htm)

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

We can create object for interface implementation classes inside the another interface implementation classes method.

**Hirarechy of Collection Interface :**



Diagram

Description automatically generated

Iterator vs Iterable:-

*The target elements in for-each loop should be iterable. And it is interface present in* ***java.lang package****.*

*If an object is* ***iterable*** *that class should implements* ***Iterable interface****.*

***Iterator*** *is* ***cursor*** *we will discussed in* ***Collections*** *framework.*

**Methods in Object class:**

Graphical user interface, application

Description automatically generated

**Final Vs Finally Vs Finalize:**

***final*** is a ***access*** ***modifier***. Which us used to apply restrictions on a class, variable or method.

***finally*** is a block in ***exception*** ***handling***. Used to execute important code irrespective of the exception occurred or not.

***finalize*** is a method of ***object*** ***class***. Which is used to perform clean up operations. It is involved each time before the object is garbage collected.

**Garbage Collection:**

Garbage Collection is process of ***reclaiming the runtime unused memory automatically***. In other words, it is a way to ***destroy the unused objects***.  In java it is performed ***automatically***. So, java provides better memory management.

**gc() :** The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in ***System and Runtime classes.***

**What are Generics in Java:**

The idea is to allow **type**. It avoids **typecasting** each object while **iterating.,etc**. If we use it . It will not allow other data type to be in collection object. For suppose we are using ArrayList of type String, compiler will not allow to insert other datatypes. It will allow to add only strings.

For **typeSafe** and to avoid **typecasting**.

**Default size of ArrayList** is ***10***.

**String vs StringBuilder vs StringBuffer:**

[String vs StringBuilder vs StringBuffer in Java | Edureka](https://www.edureka.co/blog/string-vs-stringbuffer-vs-stringbuilder/)

Strings are immutable so java came up with two other utility classes StringBuilder and StringBuffer to make string manuplation easy.

StringBuilder is not synchronized. So they are not thread safe. So it’s performance is fast compared to StringBuffer. It is stored in Heap.

StringBuffer is synchronized. So they are thread safe. So it’s performance is slow compared to StringBuilder but faster than string. It is stored in Heap.

String is not used in multithreaded environment. It is slower then StringBuilder. It is stored in string constant pool/String pool.

**Why we prefer using string as key in HashMap ?**

we generally use String as a key in HashMap is that since**String is immutable in Java that allows String to cache its hashcode**, being immutable String in Java caches its hash code and do not calculate every time we call hashcode method of String, which makes it very fast as HashMap key.

**Scopes in Spring Bean ?**

**Singleton** : Returns a single bean instance per Spring IoC Container.

**Prototype** : returns a new bean instance each time when requested.

**Request** : returns a single instance for every HTTP request call.

**Session** : Returns a single instance for every HTTP Session.

**Global** **Session** : global session scope is equal as session scope on portlet bases web applications.

Portlets provide a means of presenting data from multiple sources in a meaningful and related way. Portlets can display excerpts of other web sites, generate summaries of key information, perform searches, and access assembled collections of information from a variety of data sources.

@Scope(value = ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

@Configuration

public class PrototypeConfiguration {

@Bean

@Scope("prototype")

public MyBean prototypeBean() {

return new MyBean("prototype");

}

}

|  |
| --- |
| <!DOCTYPE beans PUBLIC       "-//SPRING//DTD BEAN 2.0//EN"       "<http://www.springframework.org/dtd/spring-beans-2.0.dtd>">  <beans>       <!--configure the bean HelloWorld.java           and declare its scope-->       < bean           id = "hw"           class= "bean.HelloWorld"           scope = "singleton" / >  </beans> |

**what is spring starters ?**

Spring Boot Starters are the dependency descriptors. In the Spring Boot Framework, all the

starters follow a similar naming pattern: spring-boot-starter-\*, where \* denotes a particular type of application

It is used to include the third party jars/dependencyies like jpa,security.,etc

**Difference between spring vs springBoot:**

|  |  |  |
| --- | --- | --- |
| S.No. | Spring | Spring Boot |
| 1. | Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on **top of the conventional spring framework**, widely used to develop REST APIs. |
| 2. | The most important feature of the Spring Framework is **dependency injection**. | The most important feature of the Spring Boot is **Autoconfiguration**. |
| 3. | It helps to create a loosely coupled application. | It helps to create a stand-alone application. |
| 4. | To run the Spring application, we **need to set the server** explicitly. | Spring Boot **provides embedded servers** such as Tomcat and Jetty etc. |
| 5. | To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| 6. | To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| 7. | It **doesn’t provide support for the in-memory database**. | It **provides support for the in-memory database** such as H2. |

Springboot avoids wiring boiler plate code.

Starter pom – which simplifies maven configurations in the boot file.

VersionManagement – we have to do it manually in spring based applications, but in springboot it is taken care by the starter project.

Autoconfiguration – we have to write configurations in server-contxext.xml/root-context.xml/web.xml. we have to configure listerns in web.xml. In servlet.xml we configuer business components require. But all these is provided as a boiler plate code by adding spring-boot-starter-web dependency. These WEB-INF folder itself not required.

ComponentScan - we have to manually provide the base package to scan in server-contect.xml/root-context.xml files, but in springboot it take the package of starter class as base package to scan.

Embeded Server – In spring based application we need a external server, but spring boot provides embedded server.

Spring-boot-starter-web comes with tomcat server. We have embedded servers like tomcat, jetty and undertow.

In Memory DB – DB starts on our application and stops on our application. For unit testing, poc or R&D we use in – memory db. In Spring framework we don’t have support for in-memory database. In Spring boot we have support for in-memory databases.

Actuators – Actuator support is not provided in spring framework. It is supported in spring boot.

when application is deployed to the production environment 24/7 monitoring is required.

How is it running ?

How much heap space is consumed ?

How many objects are there ?

How many metespace is consumed by the classes ?

How many threads are running ?

What are the configuration properties running on our application ?

What is the health of our application ? we can perform heapdump and threaddump to see the details.

**What are @RequestBody and @ResponseBody?**

a. @RequestBody : Reads data from Http Request Body Section (ex: JSON/XML) and converts into Object given as method parameter (input/request/parameter)

b. @ResponseBody : Reads output/return type from Controller method and convert into JSON/XML Format, placed in Http Response body section (output/response/return)

**what is @ResponseEntity?**

ResponseEntity **represents the whole HTTP response: status code, headers, and body**. As a result, we can use it to fully configure the HTTP response. If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

\*\*\* ResponseEntity<T> (T=String/Class/Collection)

[ Http Header, Http Status, Body(Data in Global Format) ]

**What is Devtools?**

It will restart server if any code changes are done in application.

> without devtools, we need to stop and start again manually.

> LiveReload Server is used interlally.

\*) in Spring boot just add dependency, that gives auto-configuration

LiveReloadServer, OptionalLiveReloadServer.

DEFAULT\_PORT = 35729

**What is @pathVariable?**

we can pass variable in the request URI, that can be used inside the method by passing as method parameter using @pathVariable.

**JWT token?**

Header, Payload, Signature

**Setting Datasource ?**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>6.0.6</version>

</dependency>

# MySQL

#spring.datasource.url=jdbc:mysql://localhost:3306/test

#spring.datasource.username=dbuser

#spring.datasource.password=dbpass

#spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect

We often ***do not need to specify the driver-class-name***, since Spring Boot can deduce it for most databases from the url.

**How can we configure separate databases in one springboot application ?**

**How do you handle exceptions in Springboot?**

We are writing our own custom exception handler class. Top of it we are giving ***@ControllerAdvice*** annotation. and top of method we are giving ***@exceptionHandler() and passing exception class as a parameter***. And that method will return ***response entity.***

@RestControllerAdvice

**public** **class** MyCustomExceptionHandler {

@ExceptionHandler(ProductNotFoundException.**class**)

**public** ResponseEntity<String> handleProductNotFound(

ProductNotFoundException pnfe)

{

**return** **new** ResponseEntity<String>(

pnfe.getMessage(),

HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

**What is Actuator ?**

Actuator provides pre-defined code for additional services. It provides application metrics. I mean it provides endpooints like health, info, mapping, beans.

/refresh,/shutdown

**SQL Questions:**

**Fetch unique records or values from DB Table:**

*SELECT DISTINCT keyword is very helpful when a table contains many duplicate values and we have to fetch unique records only*, for example in a table an employee name can be duplicated but we want the names without duplication.

SELECT DISTINCT customer\_id FROM customers;

**Truncate vs Delete vs Drop :**

**Delete** :It is used to delete rows or records based on conditions specified in the WHERE clause.

It is a DML command. As a result, the operation can be rolled back.

Sequence start from next value of last commit.

**Truncate** : It is used to delete the entire records of a table without affecting the schema of the table.

It is a DDL command. As a result, the changes cannot be rolled back or undone.

Sequence start from 1 after truncate.

**Drop**: It is used to delete the entire table along with its schema and structure respectively.

It is a DDL command. As a result, the changes cannot be rolled back or undone.

**Annotation on main class in spring boot:**

@SpringBootApplication

It has

@SpringBootConfiguration -- @Configuration

@EnableAutoConfiguration

@ComponentScan

@Configuration, @EnableAutoConfiguration and @ComponentScan

mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. This class automatically creates the ApplicationContext from the classpath, scan the configuration classes and launch the application.

**@Configuration** helps in spring annotation based configuration. For example we want to configure bean or datasource.

To represent java class as configuration class we use this annotations

@Configuration annotation indicated that a class declares one or more @Beans methods and may be process by the spring container

Since spring 2, we were writing our bean configuration to xml files. But spring 3.0 gave freedom to move bean definitions out of xml files. We can give bean definition inside java file itself. This is called java configuration(using @Configuration annotation)

@EnableAutoConfiguration

Based on the jar dependencies we have added. It auto-configure the beans that are present in the class-path. This simplifies the developer work of guess the required beans from classpath and configure to run the application.

For example, If I added H2 on your classpath, and you have not manually configured database connection beans, then spring will auto-configure an in-memory database. If we add web-starter, this annotation will guess that for this application tomcat server is required to run. If guess the components that are required to run the application.

***To know the components that are loading by Autoconfiguration we can go to***

*Project / MAVEN DEPENDENCEIS/ spring-boot-auto-configure-xxxxx.jar/META-INF/spring.factories*

**@ComponentScan**

It tells to look for other components, configurations and services in the specified package.

Like we are creating our own component, service, controller and repository classes. These are considered as bean after scanning.

Spring is able to auto-scan, detect and register your beans or components from pre-defined project package

If no package is specified then current class package is taken as the root package.

@Configuration is used to represent our java classes as a configuration class that is used to avoid XML based configuration.

@EnableAutoConfiguration Based on the jar dependencies provided it will autoconfigure the beans from the class path. To reduce to developer for guessing what kind of beans is required for configuring the application to start.

@ComponentScan is used to scan all component, controller, service, repository, configuration classes and register them as beans in the spring container.

**How a spring container is created?**

We have a starter class that calls SpringApplication.run(). Inside the run() method we have another run() method. That run method has the logic for bootstrapping the spring boot application. Inside this run method, we are calling createApplicationContext(). ApplicationContext is the class that stores all the information to start the application.

createApplicationcontext() method. From this method we are calling another method that is create() method that takes webApplicationType as parameter. Create() method is responsible for creation of spring container.

If we add dependency spring-boot-starter. It is default webApplicationType. So it executes AnnotationConfigApplicationContext class to create the spring container.

If we add dependency spring-boot-starter-web. It is sevelet webApplicationType. So it executes

AnnotaionConfigServletWebServerApplicationContext class to create the spring container.

If we add dependency spring-boot-starter-web-flux. It is reactive webApplicationType. So it executes

AnnotationConfigReativeWebServerApplicationContext class to create the spring container.

In Spring we used BeanFactory, ApplicationContext interfaces used to create container.

Then BeanUtils.instantiateClass(contextClass) will be called to create the object. As part of object creation

IoC Container will start

**What run() method of SpringApplication will do ?**

Stop watch starting logic is available

Getting listerners and starting listeners.

Preparing the environment

Printing the banner

Creating ApplicationContext

Calling runners

**Difference between stereotype annotations ?**

When we use @Component annotation that class bean is registered into the spring container

When we use @Controller annotation that class bean is registered into the spring container and Dispatcher will scan that class for mapping method and detect @RequestMapping

When we use @Service annotation that class bean is registered into the spring container and provides some business functionalities.

When we use @Repository annotation that class bean is registered into the spring container and it provides mechanism for storage, retrieval, search, update, delete operation on objects

When we extend JpaRepository that database exceptions can be thrown by @Repository.. If we use @Component some extra tasks and exceptions thrown it can’t able to do.

**How can we change the default port number in springboot ?**

In application.properties file, provide the key value **server.port = 8081**

**Application.properties for datasource and hibernate :**

#application.properties

#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:XE

spring.datasource.username=nit

spring.datasource.password=nit

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

#spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

-------application.properties--------------

#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/boot630pm

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

#spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

**H2 database:**

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

server.port=8080

#enable the console and provide URL

spring.h2.console.enabled=true

spring.datasource.url=jdbc:h2:mem:testdb

spring.jpa.show-sql=true

#logging.file.name=D:/mylogs/sample.log

logging.file.name=../mylogs/sample.log

logging.file.max-size=10KB

logging.file.max-history=7

logging.file.clean-history-on-start=false

#logging.pattern.file=%d [%t] %C- %M ----------- %m%n

logging.level.in.nareshit.raghu=DEBUG

#logging.level.org.springframework=ERROR

#logging.level.org.hibernate=ERROR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***hbm2ddl.auto*** :- Used for indication of Schema Creation.

It has 4 possible values.

I. ***create*** : On App start up, creates new table

(if old tables exist drop them)

II. ***update*** : Creates tables if not exist

(if old tables exist use them)

III. ***validate***:(default) Programmer has to create/modify/drop tables

IV. ***create-drop***: drop old tables -- create new tables

--- app stop --drop all tables

**How are you deploying your application ?**

In our project, we are using the git and then building jar. They are migrating to Jenkins. Jenkins deploy into the AWS.

Scheduler kind of things in Jenkins. DevOps team take care of it.

**Basic Spring Security :**

***Authentication Filter:***

Every request is intercepted by Authentication Filter. It will use Authentication Manager to perform the Authentication. If Authentication is successful, it is going to store user Authentication details into the Security Context.

***Authentication:***

Using the supplied values from the user like username and Password, authentication object will be formed which will be given as an input to the **AuthenticationManager** interface.

***AuthenticationManager:***

Once request is received from filter, it delegates the validating of the user details to the AuthenticationProvider.

***Authentication Provider:***

It has all logic of validating user details using UserDetailsService and PasswordEncoder.

***UserDetailsService:***

It retrives UserDetails and implements the User Interface using the supplied username.

***Passwordencoder:***

It is used to encode the password.

Graphical user interface, diagram

Description automatically generated

AbstractSessionFixationProtectionStrategy

**Authentication vs Authorization:**

**Authentication** is the process of validating the user credentials to access the application. Done using login pages.

Verifying login details.

**Authorization** is the process of checking whether the user is accessible to particular functionality/resource or not.

Verifying role of the user. Based of the role functionality/resources are enabled.

@EnableWebSecurity

Public class SecurityConfig extends WebSecurityConfigerAdaptor(){

}

@Configuration

@EnableAuthorizationServer

**public** **class** OAuth2AuthorizationServer **extends** AuthorizationServerConfigurerAdapter {

}

@Configuration

@EnableResourceServer

**public** **class** OAuth2ResourceServer **extends** ResourceServerConfigurerAdapter {

}

**Polymorphism :**

**Polymorphism** is the ability of an object to take on many forms. The most common use of **polymorphism** in OOP occurs when a parent class reference is used to refer to a child class object.

**Exception** is an abnormal condition which occurs during the execution of a program.

Handled using try, catch, finally blocks

**Throw and Throws:**

[Java throw, throws and finally in Exception Handling | Studytonight](https://www.studytonight.com/java/throw-throws-and-finally-keyword.php#:~:text=Java%20throw%2C%20throws%20and%20finally%20Keyword%201%20Java,keyword%20that%20will%20be%20handle%20by%20catch%20block.)

 The throw keyword is used to throw an exception to intercept the normal execution flow of a program.

and throws is used to declare the list of possible exceptions along with the method signature.

***Throws example;***

class Test

{

static void check() throws ***ArithmeticException***

{

System.out.println("Inside check function");

throw new ArithmeticException("demo");

}

public static void main(String args[])

{

try

{

check();

}

catch(ArithmeticException e)

{

System.out.println("caught" + e);

}

}

}

***Throw example:***

class Test

{

static void avg()

{

try

{

**throw** new ArithmeticException("demo");

}

catch(ArithmeticException e)

{

System.out.println("Exception caught");

}

}

public static void main(String args[])

{

avg();

}

}

**Exception vs Error :**

[Exception Vs Error in Java - Javatpoint](https://www.javatpoint.com/exception-vs-error-in-java)

|  |  |  |
| --- | --- | --- |
| **Basis of Comparison** | **Exception** | **Error** |
| **Recoverable/ Irrecoverable** | Exception can be recovered by using the try-catch block. An error cannot be recovered. |  |
| **Type** | It can be classified into two categories i.e. checked and unchecked. | All errors in Java are unchecked. |
| **Occurrence** | It occurs at compile time or run time. | It occurs at run time. |
| **Package** | **It belongs to java.lang.Exception package.** | **It belongs to java.lang.Error package.** |
| **Known or unknown** | **Only checked exceptions are known to the compiler.** | **Errors will not be known to the compiler.** |
| **Causes** | **It is mainly caused by the application itself.** | **It is mostly caused by the environment in which the application is running.** |

**Exception Handling with Method Overriding in java :**

[Exception Handling with Method Overriding in Java - javatpoint](https://www.javatpoint.com/exception-handling-with-method-overriding)

* **If the superclass method does not declare an exception**
  + **If the superclass method does not declare an exception,** subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
* **If the superclass method declares an exception**
  + **If the superclass method declares an exception,** subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

**Try-with-resource :**

[Java Try with Resources - JournalDev](https://www.journaldev.com/592/java-try-with-resources#:~:text=Java%20try%20with%20resources%20benefits%201%20More%20readable,separated%20by%20a%20semicolon.%20...%20More%20items...%20)

Java 7 has introduced a new interface java.lang.AutoCloseable. To use any resource in try-with-resources, it must implement AutoCloseable interface else java compiler will throw compilation error.

**ClassNotFoundException vs NoClassDefFoundError:**

These both errors occur when a particular class is not found at runtime. How ever they occur in different scenario.

***ClassNotFoundException*** is an exception that occurs when you try to load a class at runtime 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 runtime.

**Inversion Of Control :**

**IOC(Inversion of control)** is design principle and **DI(Dependency Injection)** is implementation of this design principle into DI design pattern.

Maintaining the dependencies when we try to inject the dependenicies. That is where container comes into picture.

It will create object, inject objet, autowire them, destroy them and maintain the life cycle methods as well.

**Dependency Injection:**

**a.** **Setter Injection (SI):** Provide data using variable set Method

(in this case default constructor is used)

Employee e = new Employee();

e.setEmpId(10);

e.setEmpName("A");

**b.** **Constrcutor Injection (CI):** Provide data using parameterized Constrcutor

Employee e = new Employee(10,"A",2.2,addr,list1);

**c.** **LookUp Method Injection (LMI)**

Special case : HAS-A

Parent--Single and Child --Prototype.

It injects the dependencies/objects which are created inside the spring container. When we use @Autowired annotation.

**Negative array size:**

int arr[] = new arr[-1];

**NegitiveArraySizeException**

**Best practices:**

followed code coverage

used small methods for each functionality and it should have it's own responsibility so that we can test it.

used interface and then implemented it.

**Concurency:**

Concurrency is simply **executing multiple tasks in parallel to each other**.

**Façade Design Pattern:**

Check for Explanation -------- [Facade Design Pattern In Java - DZone Java](https://dzone.com/articles/facade-design-pattern-in-java#:~:text=Facade%20Design%20Pattern%201%20The%20Facade%20is%20a,dedicated%20set%20of%20functionalities.%20...%20More%20items...%20)

Check for example ----- [Facade Pattern - Javatpoint](https://www.javatpoint.com/facade-pattern)

* The primary purpose of the **Facade** is to hide complexities of a system/subsystem by providing simpler interface to deal with.
* So, **Facade**hides minor and inner details of any third party library, system or subsystem we should know before we deal with it.
* In Java there are many features like **JDBC, JPA, JAX-RS** etc. which hides the minor details and provide a simpler interface in form of [annotations](https://docs.oracle.com/javase/tutorial/java/annotations/) or easier configuration to deal with.

**Volatile:**

[**https://www.youtube.com/watch?v=WH5UvQJizH0**](https://www.youtube.com/watch?v=WH5UvQJizH0)

To solve visibility problem we use volatile keyword.

At one time If one thread modifies/writes the values of a variable. That is visible to the other thread.

Avoid Cache Inchoherence. It will read/write value to cache. It will update the variable in cache as well as main memory.

**Atmoic:**

[**https://www.youtube.com/watch?v=WH5UvQJizH0**](https://www.youtube.com/watch?v=WH5UvQJizH0)**]**

To solve synchronization problem we use Atomic keywords.

It provides methods that can allow only one thread to work on that methods. When a method is executing other threads are not allowed to execute it.

These are classes. Classes comes from java.util.concurrent.atomic package

Ex: AtomicInteger, AtomicLong.,etc

**Child Interface Having Multiple Impl classes and created beans:**

[Fine-tune auto wiring using @Primary and @Qualifier | Jstobigdata](https://jstobigdata.com/spring/fine-tune-auto-wiring-using-primary-and-qualifier/#:~:text=Both%20the%20annotations%20%40Primary%20and%20%40Qualifier%20in%20Spring,usually%20used%20with%20%40Autowired%20or%20%40Inject%20etc%20annotations.)

\*) Use **@Qualifier annotation** at HAS-A Variable level to select one child object

incase of multiple found. (use this if childs are pre-defined we have Jar files

or .class files, no .java file)

\*) Use **@Primary annotation** at any one child Impl class level, that says if multiple

objects are exist then choose current class only.

=> **@Qualifier** compares given objectname with every object in container

(bit slow), **@Primary** says to **Autowire** "DO NOT SEARCH ANY OTHER, LINK TO THIS".

@Autowired

**@Qualifier(<beanName>)**

private ClassName referenceType;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@Component

**@Primary**

public class className{

…….

}

**BeanFactory and ApplicationContext:**

**\*) Spring Container**

**a. BeanFactory(I)** [Legacy Container] (XML)

**b. ApplicationContext(I)** [New Container] (XML/Java/Annotation)

**\*) ApplicationContext(I)** extends BeanFactory(I) internally.

So, all methods those exist in BeanFactory(I) we can even access

using ApplicationContext(I) Reference.

**\*) What Spring Container will do?**

> Loading Properties files

> Specify Packages to Scan your classes

> Load Clases, Create objects, Link Objects(HAS-A)

> Execute Life Cycle methods (init/destroy -- optional)

> Provide Scopes (singleton, prototype, request, session, context)

> Events

> prepostProcessors/ Aware Interfaces

> Your normal/regular methods are never called by Spring container.

> Methods called by container like: set method, life cycle methods,

event methods, Aware Methods..etc

## **How to Customize the Bean Life Cycle**

[Spring - Bean Life Cycle Explained - HowToDoInJava](https://howtodoinjava.com/spring-core/spring-bean-life-cycle/)

Spring framework provides the following **four ways for controlling life cycle events** of a bean:

1. InitializingBean and DisposableBean callback interfaces
2. \*Aware interfaces for specific behavior
3. Custom init() and destroy() methods in bean configuration file
4. @PostConstruct and @PreDestroy annotations

**What is spring boot starter parent:**

The spring-boot-starter-parent project is a special starter project that is provided by the pivotal company– that provides default configurations for our application and a complete dependency tree to quickly build our Spring Boot project.

**Spring Boot Starter Parent** helps us with managing dependency versions, the java version used by project and the default configuration for plug-ins, as we don't have to specify a lot of things manually.

It helps us with the following :

* Configuration
* Dependency management
* Default plugin configuration (default configurations for maven-failsafe-plugin, maven-jar-plugin and maven-surefire-plugin etc)

**Public , Private ,Protected, default :**

**Protected members** are visible only to the immediate subclass of the inheritance hierarchy not in all the classes of inheritance hierarchy.

**Serialization and Deserialization :**

Marker interface is a interface it may have or may not have methods but it provides special run time capabilities

Marker interfaces are empty interfaces. Emptiness not doing anything but by seeing interface implementation jvm provide special run time capabilities to the implementation class object.

Special runtime capabilities means supporting cloning process, supporting serialization, remote method invocation.

**Serialization means the process of converting java object’s data into stream of bits and bytes.**

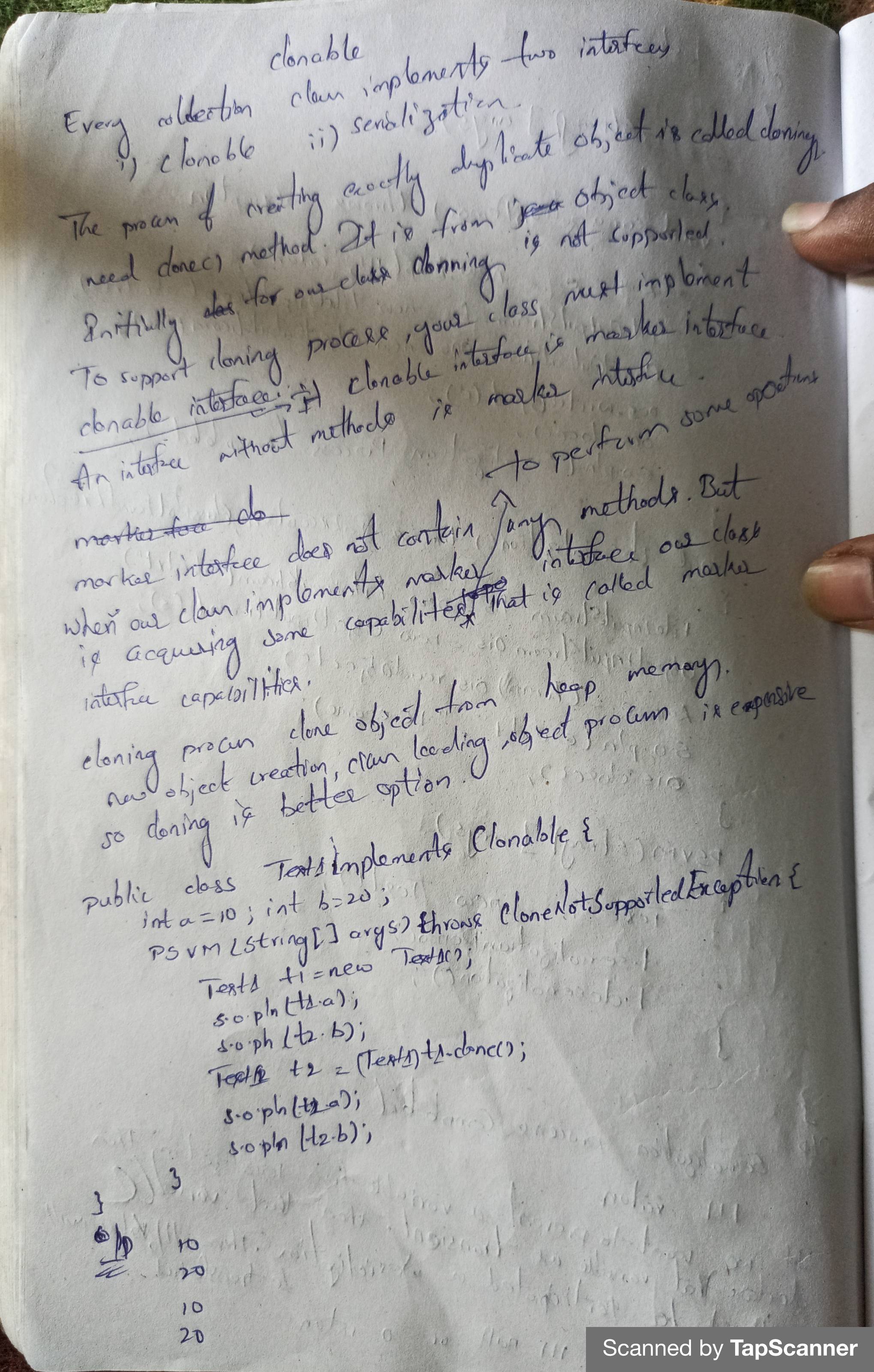
Text, letter

Description automatically generated

Text, letter

Description automatically generated

**Clonable:**



**What is marker interface?**

An [interface](https://www.javatpoint.com/interface-in-java) that does not contain methods, fields, and constants is known as **marker interface**. In other words, an empty interface is known as **marker interface** or **tag interface.** *It delivers the run-time type information about an object.* It is the reason that the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) and compiler have additional information about an object.

**Externalization:**

JVM has the control to create the object in serialization.

Initially our class does not support externalization.

To support Externalization our class has to implement Externalizable interface. We have to implement unimplemented methods.

readExternal(ObjectInput in){

in.readObject(<fieldName>); //typecast it if required

}

writeExternal(ObjectOutput out){

our.writeObject(<fieldName>);

}

If we have 100 private variables, we want only one variable to serialize. That time we can use transient keyword for all 99 variables. It is time consuming and expensive. So we follow Extenalization concept to do it.

If we use transient in externalization. There will be no effect on the code.

**Code:**

**EmployeeExter:**

package com.example.practice.externalization;

import java.io.Externalizable;

import java.io.IOException;

import java.io.ObjectInput;

import java.io.ObjectOutput;

public class EmployeeExter implements Externalizable{

private Integer id;

private String name;

private Double salary;

public EmployeeExter() {

}

public EmployeeExter(Integer id, String name, Double salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Double getSalary() {

return salary;

}

public void setSalary(Double salary) {

this.salary = salary;

}

@Override

public void writeExternal(ObjectOutput out) throws IOException {

out.writeInt(id);

out.writeObject(name);

out.writeDouble(salary);

}

@Override

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {

id = in.readInt();

name = (String)in.readObject();

salary = in.readDouble();

}

}

**ExternalizableDemo:**

**package** com.example.practice.externalization;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**public** **class** ExternalizableDemo {

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

EmployeeExter e = **new** EmployeeExter(1,"rajiv",200000.00);

**try**(

ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream("C:\\Users\\chinnapapakka.reddy\\Documents\\employee.txt"));

ObjectInputStream in = **new** ObjectInputStream(**new** FileInputStream("C:\\Users\\chinnapapakka.reddy\\Documents\\employee.txt"));

) {

//ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("C:\\Users\\chinnapapakka.reddy\\Documents\\employee.txt"));

out.writeObject(e);

//out.close();

System.***out***.println("writing is done.......\n");

//ObjectInputStream in = new ObjectInputStream(new FileInputStream("C:\\Users\\chinnapapakka.reddy\\Documents\\employee.txt"));

EmployeeExter et = (EmployeeExter)in.readObject();

System.***out***.println(et.getId());

System.***out***.println(et.getName());

System.***out***.println(et.getSalary());

System.***out***.println("reading is done.......");

//in.close();

} **catch** (IOException e1) {

e1.printStackTrace();

} **catch** (ClassNotFoundException e1) {

e1.printStackTrace();

}

}

}

**Output:**

writing is done.......

1

rajiv

200000.0

reading is done.......

**HashCode Collision in HashMap:**

Hashcode method calculates the hascode of an object by taking the key as input. If the hashCode of two objects is same value. Then it should be placed in linkedList till java 7 and in BinaryTree from java 8.

**MicroService Architecture:**

Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility.

When client sends a request, it goes to the **API** **Gateway**. It has a **GateWay** **handler**. It checks the path of the **predicate** if the **path** matches request will be send to its corresponding **URI**. We will be specifying **Eureka** **Instance** **ID** as URI. It will automatically calculate the least **load** **factor**. Based on the load factor, request will be send to a particular instance.

For **MicroService** **Communication** we can use **Feign** **Client**/**loadBalancer**.

For setting up the common properties we are using **Config** **Server**.

To check whether all the application/instances status and to check it’s properties we are using **Admin** **Server**.

For Some DB operation we are using **Apache** **Kafka** as **message** **queues**.

For Distributed tracing(finding execution path or time taken for a request and order of execution is complex. So using this approach) we are using zipkin and sleuth.

**Sleuth :** It creates **TraceId** and **SpanId** to find the execution path details and store in temp memory.(need to add sleuth in all #MS as it generated two ids and add zipkin client as it will collect and send to zipkin server). No need to add zipkin to any microsevice. Zipkin client will send data to zipkin server

**Zipkin :** It collects data from Sleuth(Sampling) and send to UI(Zipkin server)

**Zipkin Server** : A Central Server that will display all details of execution path.

Whenever a request is made. It is identified using a number. That is traceid.

How many microservices are executed and order is taken care by sleuth and this data is collected by zipkin client and store it in zipkin server.

We have spanID for each #MS and traceID for a request

Sampler collects data and sent loggers to zipkin server.

**traceId :** unique number generated to find full path details for a request.

**spanId :** To find only one MS# execution details.

**parentId :** Current MS# parentID is taken from previous MS# spanId.

Zipkin never show log statements. We need to use Log aggregator tools ie., ***ELK(Elastic Search, Logstak and kibana)***

*ELK stack support:*

*Elastic Search – (search/process/filter) search engine(based on json)*

*Logstash – (collects) – data collector, also called as datapine line b/w logstash and elastic search*

*Kibana – (display) UI where it will display our log file data*

*private static final Logger log = LoggerFactory.getLogger(<className>.class);*

**appender(File) and layout(pattern):**

**log4j.properties:**

# rootLogger=PriorityMethod, app names

log4j.rootLogger=TRACE, abc, xyz

# appender

log4j.appender.abc=org.apache.log4j.ConsoleAppender

log4j.appender.abc.Target=System.out

# layout

log4j.appender.abc.layout=org.apache.log4j.PatternLayout

log4j.appender.abc.layout.ConversionPattern=%d{dd/MM/yy} %C [%M] - %p : %m %n

# File Appender

log4j.appender.xyz=org.apache.log4j.FileAppender

log4j.appender.xyz.File=d:/mylogs/myapp.log

log4j.appender.xyz.layout=org.apache.log4j.PatternLayout

log4j.appender.xyz.layout.ConversionPattern=%d{dd/MM/yy} %C [%M] - %p : %m %n

Diagram

Description automatically generated

**Internal implementation of HashMap ?**

When we put a key-value pair in hashMap object. First it will calculate the hascode of the key and then it will perform some binary operation on top of it. Then based on the hascode. HashCode will be used to determine the bucket in which the value will be stored.

Basically HashMap has buckets. It has 16 buckets(0-15). A bucket has key, Value, Prev and Next.

**Internal Implementation of ArrayList ?**

*1.Internally ArrayList uses Array of Object class to store it’s elements.*

*2.When initializing an ArrayList you can provide initial capacity then the array would be of the size provided as initial capacity.*

*3.If initialcapacity is not specified then Default Capacity is used to create an array . DefaultCapacity is 10.*

*4.When an element is added to an ArrayList it first verifies whether it can accommodate the new element or it needs to grow, in case capacity needs to be increased then the new capacity is calculated which is 50 % more than the old capacity and the array is increased by that much capacity*

*5.when elements are removed from an ArrayList space created by the removal of an element has to be filled by underlying array. This is done by shifting any subsequent elements to the left*

It’s a resizable array implementation of list interface.

Arraylist provides three cosntructs we can use any one of the constructor while creating its object.

**Default** ArrayList

Here ArrayList initializes to an empty array i.e.,elementData

**Parameterized** arraylist which takes capacity and created an Object Array with initialCapacity i.e.,elementData

**Collection** **Data** as parameter

If we pass collection, collection will be converted into an array using c.toArray() and will be initialized to an array i.e, elementData.

When we add an element to arrayList, a method called ensureCapacityInternal() method will get executed.

There we are calculating the initial size of arrayList. If we are providing initial size it will be taken or else DefaultValue 10 will be taken as minCapacity.

ensureExplicitCapacity() here modcount is incremented by 1 as we are adding one element and grow() method will decide whether the size should be increase or not.

When we remove element, the created by removing an element will be filled by the underlying array,that is done by the shifiting any subsequent element ot the left. It is done by the method System.arraycopy takes 5 parameters.

Till java 6 capacity is calculated using **int newCapacity = (oldCapacity \* 3)/2 + 1;**

Form java 7 , **Int newCapacity = (oldCapacity + oldCapacity >> 1);**

we pass minimum capacity into it. It will calculate the max capacity and executes the ensureExplicatCapaity(int minCapacity)[minCapacity value is 10].. It increments modCount by 1. And checks the capacity. If the is incremented then grow() method increments the capacity by 50% doing right shift.

**How to reverse a LinkedList ?**

**HashSet vs HashTree ?**

## **Difference Between HashSet and TreeSet**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **HashSet** | **TreeSet** |
| **Ordering or Sorting** | It does not provide a guarantee to sort the data. | It provides a guarantee to sort the data. The sorting depends on the supplied Comparator. |
| **Null Objects** | In HashSet, **only an element** can be null. | It does not allow null elements. |
| **Comparison** | It uses **hashCode()** or **equals()** method for comparison. | It uses **compare()** or **compareTo()** method for comparison. |
| **Performance** | It is **faster** than TreeSet. | It is **slower** in comparison to HashSet. |
| **Implementation** | Internally it uses **HashMap** to store its elements. | Internally it uses **TreeMap** to store its elements. |
| **Data Structure** | HashSet is backed up by a hash table. | TreeSet is backed up by a Red-black Tree. |
| **Values Stored** | It allows only **heterogeneous** value. |  |

**If parent bean scope is prototype what is the default bean scope of child bean ?**

Child bean will not get parents scope.

**Difference between POST vs PUT :**

**POST** : create new resource at server

**PUT** : Fully Modify resource at server

**Difference between POST vs GET ?**

**GET : In Get if we want to provide data we have to provide data in the url**

**POST : He we are providing the data/parameters in the HttpBody Section**

s

**Find the second highest salary ?**

**Highest Salary:**

select \* from employee where salary=(select max(salary) from employee);

Highest name of employee who has highest salary and age>10:

Select e.name from employee e where e.age>10 and e.salary=(select max(salary) from employee)

Select e.name from employee e orderby e.salary desc limit 2,1;

Select e.name from employee e orderby salary desc limit 2,1;

**Second Highest Salary:**

**Find nth highest value:-**

> select distinct bcost from booktab order by bcost desc limit 2,1;

> select \* from booktab limit rowId,maxRows;

> select \* from booktab limit 0,3;

> RowId is givenby Database starts from zero.

**Find the second largest element in Array ?**

list.stream().sorted(Comparator.reverseOrder()).limit(2).skip(1).findFirst()

returns optional

use optionalObject.get() to get the value.

**Stateful Authentication and Stateless Authentication :**

*State means memory or data of client at server side.*

**Stateless Authentication** is also called as Token based Authentication. Ex: JWT Authneticaiton.

If login credentials are valid(Auntenticated successfully), one token will be generated. that token will be sent to client as a response body or cookie. Client stores the token as a cookie or it can store in inmemory storage or databases.

When client making any request, tokens are submitted along with the request. Sever will read token from request and validate it.

If token is not available(empty request), it says us to login again.

Token will be generated using secret key. Secret key will be maintained at server side and not be shared to any client, By using this secret key only we can read this token as this is in unreadable format(encoded format).

No Logout type of concept, after sometime token gets expired. We are providing some time to expire the token.

In webservices we use stateless authentication(Server to Server).

In horizontal scaling(Microservices), Resource grant concept(Oauth.open Authentication) also we use stateless authentication.

Diagram

Description automatically generated

Stateless concept never allocates any memory at server. Statefull uses session concept. It need memory as session is an object.

**Limitation :**

Token must be validated on every request.

If token is shared with others then they can access client data/services.

**Statefull Authentication:** It completely works on session concept.

One memory is created in *server* when client is *Authenticated* successfully. That *memory* is called session. This session can store data in the form of key-value pair. It has unique number that is session ID. This *sessionID* will be sent as response in the cookie to the client. *Cookie* will be stored at client side or browser side.

When client making any request, cookies are submitted along with the request. Server will check session ID is matching. If matching server will give *response*.

When client perform *logout*, session will be *invalidated*. *Invalidate*() method gets invoked on session object.

HttpSession is an interface. It’s object is created using HttpServeltRequest object.

General web applications uss Statefull authentication.

Diagram

Description automatically generated

Diagram

Description automatically generated

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**JWT(Json Web Token):**

It is a open source service that supports generating Token based on client details and secret key.

JSON Format:

It contains three parts

1.Header : JWT Specific Information

2.Payload : claims (ClientId, ClientName, providerName, date, expDate)

3.Signature : base64Encoder(Header) + base64Encoder(payload) . It is signed with secret key.

**Example :**

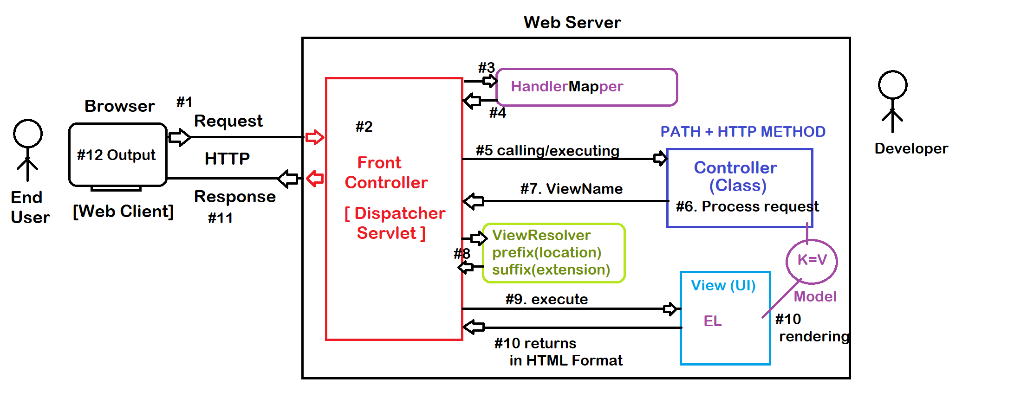
Header . payload . signature

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Builder patterns:**

[**https://howtodoinjava.com/design-patterns/creational/builder-pattern-in-java/**](https://howtodoinjava.com/design-patterns/creational/builder-pattern-in-java/)

**SpringMVC pattern :**



**Gang of Four Design Patterns :**

<https://springframework.guru/gang-of-four-design-patterns/>

**Check the max limit How load is balancing on microservices ?**

Checked.

**Thymeleaf :**

$ -- Reading Data from Model memory

\* -- point to variable in object (Form Binding)

@ -- used to indicate path/URL/location

# -- SpEL - Spring Expression Language

**HikariPool** is a JDBC DataSource implementation that provided connection pooling mechanism.

**Self Intro:**

My name is varun. I'm from kalikiri. I have done my graduation from MITS with 84%.

Then I joined HCL as Fresher. Since then I was working as Java full Stack Developer.

I have 2.5 years experience in technlogies like Spring Boot, Hibernate/JPA, mysql ReactJS and little bit experience on Microservices.

My Project name is Service Assurance. It's a Telecom Domain.

Our project deals on the services in 5g provided by the carrier to the customers.

These services would be including the activities such as voice calls, data, mobility(change from cell to cell), and policies which are promised by the carrier.

The services provided by the carrier are all measured using the parameters specified in 3GPP specs. These we actually called

KPI (key performance indicators).

So this project created by us is a POC where it will be showcased to the Vendors/carriers by our HCL sales team where the customer would be interested to give us the real data and can seek help from us to as we are aware of the predicted anomalies.

while calling we might get 100% performance there might be disturbances in the quality of service provided.

So we do calculate the quality of service getting effected using kpi's.

So that anamalies are showcased and solutions to fix it are then planned and executed to fix it for better efficiency.

we are using tools like Agile, Spring boot, Hibernate, JPA, mysql, ReactJS.

My responsibilties are I implemented spring security along with JWT token that can allow only some group of people to access the resources.

I worked on some modules like core, ran, transport, e2e, slice. We have endpoints like Live Data Monitoring, Anomaly Summary, Anomaly Detection, Cross Layer Correlation, Root Cause Analysis. We are using graphs like bar graph, line graph, Area graph, Sector Graph, pie diagram to show the KPI values.

We are showing Root Causes using variables and show casing variables as well.

I worked on another project it is Shopping Cart. We have different modules. This is a microservice project

Continue…..

**\*) Common Http Status code**

200 OK - Successfully processed

201 CREATED -- Successfully created new resource at server (POST)

400 BAD REQUEST -- Data is Missing for Request

401 Unauthorized - Un/pwd is invalid

403 FORBIDDEN - Role is invalid to access resource

404 Not Found - Resource not exist at Producer

405 Method not allowed - Request HttpMethod, Code HttpMethod not matching

406 Not Acceptable - Request MediaType output is not supported

Consider you did not add ***jackson-dataformat-xml dependency*** and Request Header 'Accept=application/xml' is provided. In that case output is : ***406 - Not acceptable***

415 Unsupported MediaType - Request Body convertsion can not be done

If we provide XML as input and ***jackson-dataformat-xml*** not exist, then ***415-Unsupported MediaType*** error is provided.

500 Internal Server Error - Exception at producer application

**1xx: Informational** – Communicates transfer protocol-level information.

**2xx: Success** – Indicates that the client’s request was accepted successfully.

**3xx: Redirection** – Indicates that the client must take some additional action in order to complete their request.

**4xx: Client Error** – This category of error status codes points the finger at clients.

**5xx: Server Error** – The server takes responsibility for these error status codes.

**Nested classes in java:**

Nested classes are two types :

1. **Static Nested Classes**
2. **Non-Static Nested Classes**

**Inner classes :**

Outer class methods are not allowed to access inner class methods.

Inner class is allowed to access outer class private members

Inside inner class static members are not allowed

**Method local inner classes**

Class inside the method is called Method local inner class.

Method local variables should be final till java 7 after then it is optional.

**Annoyomous inner class**

Implementing abstract method while creating object.

**Swagger:**

Swagger client/Swagger UI is a Open Source HTTP Client.

\*) This can be configured in application one time that will scan

all our controller classes and creates one dynamic UI

which makes HTTP Request to test those method with input

and display Final Response.

\*) No need to enter URLs manually, no need to provide HTTP methods manually.

Model class views given. Easy to execute, hide using Profiles.

\*) Swagger Configuration output is Docket(Documented Configured Output)

\*) We must provide input of common package name for all RestController

to scan them using

.apis(basePackage(\_\_\_\_\_\_\_\_\_\_))

with condition having one common path

.paths(regex("\_\_\_\_\_")

dot(.) - character Star(\*) - zero to many

/rest.\* => starts with /rest and may contain zero to many other chars

@Configuration

@EnableSwagger2

public class SwaggerConfig {

@Bean

public Docket createDocument() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(basePackage("in.nareshit.raghu.rest"))

.paths(regex("/rest.\*"))

.build()

.apiInfo(apiInfo());

}

private ApiInfo apiInfo() {

return new ApiInfo(

"MY PRODUCT APP",

"SAMPLE",

"3.3GA",

"https://nareshit.in/",

new Contact("AA", "http://aa.com", "aa@gmail.com"),

"NIT Ltd", "https://nareshit.in/", Collections.EMPTY\_LIST

);

}

}

\*)

**How to test a endpoint:**

First we will create a mock for webEnvironment using @SpringBootTest annotation

@SpringBootTest(webEnvironment = WebEnvironment.MOCK)

@AutoConfigureMockMvc to create required object and link each other.

Then we have to create a http request using MockHttpRequestBuilders that can return MockHttpServletRequestBuilder.

We process this request using process method in Mockmvc and it returns a MvcResult.

MvcResult object contains request + response + other details.

We use getResponse() method to get MockHttpServletResponse.

**@Query:**

We write this annotation over the method in repository interface.

When we are unable to fetch the records using findBy methods like for join operations then we will use @Query annotation.

@Query(“<sql query use members of entity class>”)

If we want to execute native sql query we pass two params to @Query annotation. value and nativeQuery

@Query(value=”SQL”,nativeQuery=”<pure db query>”)

If we are using query to update/modify the table or DML operation. Then we have to give to more annotations

Over @Query annotation

@Modifying

@Transactional

@Quey(“<query>”)

**Stored Procedure Query:**

Procedure is a named block which has set of sql queries and logics that will be executed when ever we call it.

We can write in three ways:

**Normal procedure**

No input and no output

DELIMITER $$

CREATE PROCEDURE PROCEDURENAME()

START

SELECT \* FROM EMPLOYEE;

END $$

DELIMITER;

**In procedure**

Only In param no out param

DELIMITER $$

CREATE PROCEDURE PROCEDURENAME(IN EMP\_DEPT VARCHAR(20) )

BEGIN

SELECT \* FROM EMPLOYEE E WHERE E.EMP\_DEPARTMENT=EMP\_DEPT;

END $$

DELIMITER ;

**iIn-Out Procedure**

in and out params

DELIMITER $$

CREATE PROCEDURE PROCEDURENAME(IN EMP\_DEPT VARCHAR(20),OUT DESC\_COUNT INT)

START

SELECT COUNT(\*) INTO DESC\_COUNT FROM EMPLOYEE E WHERE E.EMP\_DEPARTMENT = EMP\_DEPT;

END $$

DELIMITER

**To execute procedure query**

We have to autowire EntityManager and then createStoredProcedureQuery() using em

if we have any in or out param, register using registerStoredProcedureParameter(position,type,modeOfInOrOut) it then execute it

Then set inparam using setParam(“name”,value)

**Example:**

in param, OUT PRAMETER

DELIMITER $$

CREATE PROCEDURE GETEMPSCOUNTBYDESG(IN EMP\_DESG VARCHAR(20), OUT DESG\_COUNT INT)

BEGIN

SELECT COUNT(\*) INTO DESG\_COUNT FROM EMPLOYEE E WHERE E.edeg=EMP\_DESG;

END$$

DELIMITER ;

**Program:**

@Autowired

private EntityManager em;

*inside a method execution is like below*

/ /create SP Query

StoredProcedureQuery query = em.createStoredProcedureQuery("GETEMPSCOUNTBYDESG");

//register params

query.registerStoredProcedureParameter("EMP\_DESG", String.class, ParameterMode.IN);

query.registerStoredProcedureParameter("DESG\_COUNT", Integer.class, ParameterMode.OUT);

//set

query.setParameter("EMP\_DESG", "SE");

//execute

query.execute();

//read result and print

int count = (Integer) query.getOutputParameterValue("DESG\_COUNT");

System.out.println(count);

**How to call private method from outside of the class:**

<https://www.javatpoint.com/how-to-call-private-method-from-another-class-in-java>

We can call private method form outside of the class by changing the runtime behavior using java.lang.class and java.lang.reflect.Method Class

*File: A.java*

**public** **class** A {

**private** **void** message(){System.out.println("hello java"); }

}

*File: MethodCall.java*

**import** java.lang.reflect.Method;

**public** **class** MethodCall{

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

Class c = Class.forName("A");

Object o= c.newInstance();

    Method m =c.getDeclaredMethod("message", **null**);

    m.setAccessible(**true**);

    m.invoke(o, **null**);

}

}

**How to make a class immutable ?**

Make class as final, members as private and final, there should not be any setters. Make deep object if we have any object as member.

Not final subclass can extend and change the value. Private as usual and if not private it will be accessed from outside.

If members are not Final. We can change it by creating the object. If there are setters, others can set the value by creating the object.

**Why hasmap keys are strings most of the time ?**

*How immutability helps strings as a key in HashMap?*

*String Hashcode caching*

When we calculate the hashcode of the string it will be cached that forever.

Hashcode for the string is computed only one time, then getting cached inside the string object itself. If we get the same value instead it will use the hashcode which is already calculated because string is immutable it content will not be changed

**Wrapper classes in java:**

Char, Boolean, Byte, Short, Integer, Long, Float, Double

String is not wrapper class

**Queue:**

[Stack Implementation in Java - Studytonight](https://www.studytonight.com/post/stack-implementation-in-java#:~:text=Stack%20implementation%20using%20Stack%20Class%201%20Creation%20of,have%20covered%20the%20Stack%20Data%20Structure%20in%20Java.)

**Dequeue:**

[Queue Implementation in Java - Studytonight](https://www.studytonight.com/post/queue-implementation-in-java#:~:text=Queue%20Implementation%20in%20Java%201%20Basic%20Functions%20of,in%20Java.%20...%205%20You%20may%20also%20like%3A)

**Abstract vs Interface:**

[Difference between Abstract class and Interface - Javatpoint](https://www.javatpoint.com/difference-between-abstract-class-and-interface#:~:text=Difference%20between%20abstract%20class%20and%20interface%20%20,public%20b%20...%20%205%20more%20rows%20)

**How string working in the background with new keyword and “” [Constant Literals].**

**When we create a string using new keyword**

**String s1 = new String(“Java”);**

First it will read the constant that has literals and create object in string constant pool. And new means it has run time operation so it created on more object in Heap. As of now we have two string objects with content “java” one in String constant pool and the other one is in Heap.

**String s2 = new String(“Java”)**

Now if we create one more object using same content with new keyword. It will read contant that has literals and try to create object in string constant pool. As content java already exists it will not create new object. And it has run time operation it will create new Heap object s2 with content java as it don’t have nay restrictions on creating duplicates but String constant pool will not allow duplication.

Now one object in String constant pool and the other two objects in Heap area. So in total we have three objects.

String s1 = “java”;

s1=”content”;

System.out.println(s1); // content

Internally first s1 object created and referred to content “java” which is in String Constant Pool. when we again assign s1 with other content with constant along with literals. JVM will check the value is present in the String Constant Pool. If it is there it will not create object instead it will just assign reference to the existed object. If it is not there, object will be create with content “content” and assign reference to s1.

**How can we convert json to object and object to json?**

We have ObjectMapper class it has method writeValuesToString(<object>) that takes object as paramerter and convert into string and return string.

ProductInfo pinfo = new ProductInfo("TEST-HYD","NIT");

String data = new ObjectMapper().writeValueAsString(pinfo);

We can have the same ObjectMapper class it has method readValues(string,<ClassName.class>) that can convert string into class type and return it.

ProductInfo pinfo = new ObjectMapper().readValue(json,ProductInfo.class);

Message Queues:

Queue, Topic

**What is stream api ?**

Stream api is used to perform bulk operations on a sequence of elements

It is used to process collection of objects. A Stream is a sequence of objects that supports various methods which can be pipelined to produce the desired output.

**Lamda expressions vs method reference :**

If the job of lamda expression is already available in another method of a class. Then we can simple refer that method.

functional interface refer that method.

We no need to write lamda expression, we can directly refer the method of a particular class.

Advantage is code reusability.

Purpose of double colon operator(::) is method reference or constructor reference.

**Method reference :**

Functional interface method can be referred to our specified/already existed method by using double colan operator.

irrespective of the return type and modifiers. But it always cares about the no of arguments.

Runnable r = Test::m1; //if m1 is static // arguments of run() method in runnable interface should match with the arguments of m1() method in Test class.

Runnable r = new Test():m1; if m1 is static

If implementation of method is already available, we can go with that method reference to reuse the code.

**Constructor Reference:**

Interface get() method refers to constructor of a particular Sample class which can take same type of argument. If a method is creating object of another class. We can use this Constructor Reference. Here also arguments to be marched. It will always call it’s corresponding constructor.

If we have a method in functional interface that is returning the class. Then we can use constructor reference.

To use this, the interface method return type should be class. We no need to write return object.

If method reference and constructor reference is not possible then go with lamda expressions. If lamda expressions also not possible then for normal implementation concept.

**when will we go for functional interfaces?**

When we are implementing an interface that has single abstract method then we will go with functional interface.

*class Sample{*

*Sample(){*

*System.out.println(“Sample Constructor execution….”);*

*}*

*}*

*Interface interf{*

*public Sample get();*

*}*

*Class Test{*

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

*interf i = ()=> new Sample(); // using lamda expressions*

*Interf i = Sample::new; // using constructor reference// here we are just refering*

*Sample s1 = i.get();*

*}*

*}*

*Output:*

*Sample Constructor execution….*

Indexes in Database :

Index makes the database retrival faster. Indexes are used by queries to find the data from tables quickly. If there is no index to help the query then query engine performs a entire table scan.

If we apply index on any column. It will sort the column in ascending order and store it in separate memory along with the address of the row.

If we apply index on salary column. This column will be sorted in ascending order and sorted column alone will be stored in separate memory along with the address of the row.

When we query for a record using where clause on salary, then internally, it will first go to the sorted column which is in separate memory, then it will search and get the index. Then it directly go to the row in the original table based on the index.

Separate space for index is created.

CREATE INDEX <nameOfIndex> ON TABLENAME(COLUMNNAME)

DROP INDEX <nameOfIndex> ON TABLENAME

To get index of tables we have to know the table\_id first

To find the table id:

Select table\_id from information\_schema.innodb\_sys\_tables where name=’publisher/test5’;

Here publisher is the database name and table5 is the table name

Select \* from information\_schema.innodb\_sys\_indexes where table\_id=1576

Where table\_id value is the one which is obtained from previous query.

Type of index is 0. This is secondary index.

Clustered and secondary index:

If we apply clustered index like primarykey or any other. The table will be sorted on the index will be applied on the same table.

Primary key and foreign key creation in mysql:

Generation key values in :

What if we have to same string in hashmap key ?

Overrides equals and hashcode methods in model class.

**Some design patterns ?**

Creation pattern, Structural pattern and behavioral pattern.

**Builder pattern**

[Builder design pattern in java with real life example and code (stacktraceguru.com)](https://stacktraceguru.com/builder-pattern/)

For simplifying the creation of complex objects.

Different representation of object is possible.

**Context Object Design Pattern :**

It is used to store reference to configure information and services needed by other objects.

* [**ApplicationContext**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html)is the central interface within a Spring application for providing configuration information to the application.
* [**SecurityContext**](https://docs.oracle.com/javaee/7/api/javax/ws/rs/core/SecurityContext.html) is used to store the details of the currently authenticated user and which can further be accessed through the application.
* [**ServletContext**](https://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html)is used to share configuration information with all the servlets.

**Observable – Observer Design Pattern :**

Observable is the object that notifies all the observers when it’s state changed.

**Java 8 interface default method call ?**

 The reason we have default methods in interfaces is to allow the developers to add new methods to the interfaces without affecting the classes that implements these interfaces

Static methods in interfaces are similar to the default methods except that we cannot override these methods in the classes that implements these interfaces.

**interface** Clickable{

**default** **void** click(){

System.out.println("click");

}

**default** **void** print(){

System.out.println("Clickable");

}

}

**interface** Accessible{

**default** **void** access(){

System.out.println("access");

}

**default** **void** print(){

System.out.println("Accessible");

}

}

**public** **class** Button **implements** Clickable, Accessible {

**public** **void** print(){

Clickable.**super**.print();

Accessible.**super**.print();

}

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

Button button = **new** Button();

button.click();

button.access();

button.print();

}

}

**How to define our own exception ?**

We have to extend Exception class and write parametrized constructor and pass a message as parameter inside that constructor call super(message) and pass message there.

How to define our own uncheckedException ?

We have to override RuntimeException class and write parametrized constructor and pass message as parameter inside that constructor and call super(message) and pass message there.

class CustomException extends Exception {

   String message;

   CustomException(String str) {

      message = str;

   }

   public String toString() {

      return ("Custom Exception Occurred : " + message);

   }

}

public class MainException {

   public static void main(String args[]) {

      try {

         throw new CustomException("This is a custom message");

      } catch(CustomException e) {

         System.out.println(e);

      }

   }

}

## Output

Custom Exception Occurred : This is a custom message

**Can we override a static method ?**

[Can We Override Static Method in Java - Javatpoint](https://www.javatpoint.com/can-we-override-static-method-in-java#:~:text=Can%20we%20override%20a%20static%20method%3F%20No%2C%20we,object%20that%20calls%20the%20static%20method.%20It%20means%3A)

### **Can we override a static method?**

**No,** we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.

The calling of method depends upon the type of object that calls the static method. It means:

* If we call a static method by using the parent class object, the original static method will be called from the parent class.
* If we call a static method by using the child class object, the static method of the child class will be called.

**public** **class** OverloadStaticMethodExample3

{

**public** **static** **void** main(String args[])

{

ParentClass pc = **new** ChildClass();

//calling display() method by parent class object

pc.display();

}

}

//parent class

**class** ParentClass

{

//we cannot override the display() method

**public** **static** **void** display()

{

System.out.printf("display() method of the parent class.");

}

}

//child class

**class** ChildClass **extends** ParentClass

{

//the same method also exists in the ParentClass

//it does not override, actually it is method hiding

**public** **static** **void** display()

{

System.out.println("Overridden static method in Child Class in Java");

}

}

**Output:**

display() method of the parent class.

**Difference b.w class.forName() vs getClass() vs using .class to load class at runtime:**

**Class.forname to load the class at runtime.**

**getClass() method to get the runtime type of the class as fully Qualified Class Name.**

**When we call the str.getClass() method, we get the runtime type of the str object. On the other hand, String.class evaluates the String class statically**.

public final Class<?> getClass()

: Returns the runtime class of of the Object on which it is invoked.If an instance of an object is available, then the simplest way to get its Class is to invoke Object.getClass()

public static Class<?> forName(String className) :

Returns the Class object associated with the class or interface with the given string name.

A call to Class.forName("fullyQualifiedClassName") causes the class to be dynamically loaded (at runtime). A call to forName("fullyQualifiedClassName") causes the class to be initialized (i.e., JVM executes all its static block after class loading)

**Comparator values returning:**

“ratan”.compareTo(“anu”) ==> +ve ==>change the order “ratan”.compareTo(“ratan”) ==> 0 ==>no change “anu”.compareTo(“ratan”) ==> -ve ==>no change

**equals() and ==**

**SOA(Service Oriented Architecture):**

This Design is used to create link between two applications by using mediator i.e., Register with Docement(XML)

SOA has 3 component and 3 operations:

1.Service Provider : Application Providing service(OTP Service, Email Alert, Converter Service etc.,)

2. Service Consumer : Application that reads services(any web application)

3. Service Register : Register is a memory. It contains documents(XML) of service providers

3 Operations :

1. PUBLISH : Read Skeleton(Provider Code) and Generate Document(XML) finally place in Service Register.
2. FIND : Read Document from Register and generate stubs(Consumer side code like classes at Consumer Application)
3. BIND : Using HTTP Protocol for communication link with data in XML Format.ie request and response. This flow has stub request and skeleton response.

Convert java object to xml is called ***Marshalling*** and converting xml to java object is called ***unMarshalling***.

**Dynamic overloading dispatcher ?**

**Business Delegate Design pattern ?**

**What is the difference between After, After Returning, After Throwing Advices?**

After advices is executed next to b.method either success or fail.

After Returning advice is executed only on succssful execution of b.method

After Throwing advice is executed only on Failure(Exception) execution of b.method

**Spring AOP:**

**Spring Aspect Oriented Programmng:** It’s the process of separating the Addititonal Services from the actual code and finally linking when and where thery are required. To do that we need Aspect class advices pointcuts jointPoint and enabling Proxy objects

**Cross Cutting Concers :** Logic related to business class/Additional class is called concerns. Moving common code or additional code from business class method to other classes(Aspect class) and bind whenever they are required.

**Aspect (class):** Aspect is a annotation that is provided over the class that contains advices/additional services to the project.

**Advice(method) :** It is a method that contains the additional services and should need to be connected with any of the below types.

*@After, @Before, @Around, @AfterReturning, @AfterThrowing*

*Before/after means that method is executed before/after execution of business method.*

**Pointcuts:** Pointcut is an expression that will select business class methods which needs advices.

**JointPoint:** It is a combination of both advice + pointcut. It connects business class methods with advices.

**Target:** Pure business class object.

**Weaving :** it is a process of mixing business class method and their connected advices.

**Proxy:** It is final output(class/object) that contains both business class methods and advices.

**Pointcut syntax:**

//this says go to particular package and then classname and method that has specifier specifier and returntype.

@AdviceType(“execution=(Specifier ReturnType Package.className.MethodName(ParameterTypes))”);

**Allowed symbols:**

\* 🡪 any, . 🡪 in between packageName and ClassName, .. 🡪 all/any number of arguments to the method

Note : allowed symbols in Pointcut expression: \*(star), .(dot)

-------Examples-----------------------------------------------------

*#1) public int in.nit.dao.EmployeeDao.saveEmployee(Employee)*

=> saveEmployee() method having Parameter 'Employee' with return type 'int'

of type public defined in class EmployeeDao (in.nit.dao)

is selected to connecte with advice.

*#2) public \* in.nit.dao.EmployeeDao.\*()*

=> Zero Parametes only (no parameter)

=> Any method name/ Method inside EmployeeDao class

=> Any return type

*#3) public \* in.nit.dao.EmployeeDao.\*(..)*

Two dots inside method params indicated any no.of/type of parameters.

=>Zero or more params (of any type) [Any Parameter is fine]

*#4) public \* in.nit.dao.\*.\*(..)*

=> all classes which ar inside in.nit.dao package and thier methods

If we write pointcut and joint point it will not work as ***creating proxy object*** is not ***enabled***. To make it work we need to add annotation at main class level ***@EnableAspectJAutoProxy*** annotation.

**Step#1** Spring Starter Project

Name: SpringBoot2AopEx

pom.xml

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-aop</artifactId>*

*</dependency>*

***S#2*** At starter class: *@EnableAspectJAutoProxy*

**S#3 Business class**

*package in.nareshit.raghu.dao;*

*import org.springframework.stereotype.Component;*

*@Component*

*public class EmployeeDao {*

*public String saveEmployee() {*

*System.out.println("FROM SAVE EMPLOYEE");*

*/\*if(new Random().nextInt(15)<=10) {*

*throw new RuntimeException("DUMMY EXCEPTION");*

*}\*/*

*return "HELLO";*

*}*

*}*

**S#4 Aspect class:**

*@Aspect*

*@Component*

*public class TxService {*

*// pointcut : AS RT PACK.CLS.MN(PARAM)*

*@Pointcut("execution(public String in.nareshit.raghu.dao.EmployeeDao.saveEmployee())")*

*public void p1() {}*

*// advice*

*@Before("p1()") //joinpoint*

*public void beginTx() {*

*System.out.println("Tx Started!!");*

*}*

*@AfterReturning(value = "p1()", returning = "ob")*

*public void commitTx(Object ob) {*

*System.out.println("Tx is committed" + ob);*

*}*

*@AfterThrowing(value = "p1()",throwing = "th")*

*public void rollbackTx(Throwable th) {*

*System.out.println("Tx is rollback." + th.getMessage());*

*}*

*@After("p1()")*

*public void sendReport() {*

*System.out.println("Report sent!");*

*}*

*}*

**Batch Processing :**

Transfer larger data from Source to Destination with some operations.

**Job :** Actual task that indicated sending data from source to destination.

**Step :** One job can have 1 to n Steps. It contains ItemReader, ItemProcessor, ItemWriter**.**

**Reader:** Reades data to source

**Processor:** will calulate/operations

**Writer:** writes data to destination

Job is started by **JobLauncher.**

**Job Repository (H2/MySQL) :** Stores all details of jobs.

**To create a step we need :**

Step [I] = StepBuilderFactory()

stepName + reader implementation class object + processor implementation class object + writer implementation class object + chunk(int)

If we want to execute a logic before or after job. We have

JobExecutorinListenre [I}:

Void beforeJob(JobExecution je)

Void afterJob(JobExecution je)

Status of the batch like started, completed, at what time it is completed all are known by using enum BatchStatus

**To create job we need**

Job[I] : JobBuilderFactory[C]

[name + steps + execution order + listener]

**What Reader Implementation class will do ?**

Read/link file using setResource()

Read data line by line using LineMapper

Tokenize data using delimiter(,)

Converting given values into one object using fieldname and type.

**What writer class implementation class will do ?**

Create datasource connection

Collect data into List<T> based on chunk size

Define sql query for insert using named parameters

Generate sql with data

**What is Horizontal Scaling and Vertical Scaling ?**

**Horizontal Scaling** means adding one more instance of new application at new vm

**Vertical Scaling** means enhancing capacity and enhancing hardware configuration of existing system.

**How to increase the threads of tomcat ?**

Add this property in application.properties file

server.tomcat.threads.max = 200s

**What is Load Factor?**

Load Factor = Current Load / Max Load

**What are profiles ?**

--spring.active.profiles = qa

**What are different types of environment in springboot ?**

Dev, qa, prod

**What is Load Balancing ?**

Load Balance = Creating Multiple Server instances and handling them on LoadFactor for multiple user request.

**Feign Client:**

It is an abstract client(It’s an interface that have methods/abstract methods). This abstract client is used to make communicate with other MS#. We can give any name for interface and abstract method too. But over the interface we have to write an annotation i.e., @FeignClient("serviceId") , but the method’s return type, Parameter, Path, HttpMethod should match with the controller method which we are trying to communicate with.

[How to Implement Feign Client in Spring Boot Microservices? - Making Java easy to learn (javatechonline.com)](https://javatechonline.com/how-to-implement-feign-client-in-spring-boot-microservices/)

**Limitations of Monilithic Application ?**

\*) Limitations :-

a) Must re-build and re-deploy in case of single module modification.

b) These are not reliable (Single Point Failure), if one module is

failed that may cause other modules even problem.

c) Enhancements/adding new technologies may need re-design of application

(or may not suitable to add).

d) As No.of Modules gets increased then Code Size and bug rate gets increased.

e) Downtime/Deployment time will be more incase of App is verybig.

**How are you handling Authentication ?**

We have a API Gateway, all the requests from front-end comes to the api gateway. There we defined a ***Customfilter*** that extends ***GatewayFilter***. ***GatewayFilter*** is a abstract class that has a filter method. It takes two ***patameters*** that is ***SeverWebExchange*** and ***GatewayChainFilter***. ***SeverWebExchange*** has ***getRequest()*** method that can return ***ServerHttpRequest***. Inside that request we check for ***Authorization*** ***header***. If it is there we validate it using a ***screat*** ***key*** otherwise we send ***ServerHttpResponse*** by adding a status code ***UnAuthorized*** and made that response as complete using ***setComplete*** method. If it is successfully validated then we are using filterchain to continue the process.

[Spring Cloud Gateway](https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/)

**How are we validating Authorization Header ?**

We have a validate method that takes the token as input and then it use predefined Jwts class and then parse it using parse() method and then set a secret key while parsing and then we ask to parse claims by passing token and then we get body that has details of jwt token. And then we are adding the required details in the header.

***Claims body = Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();***

**How Gateway is working?**

We have defined some routes, there is a predicate for every route if path matches it will return true. And URI is associated with that route. URI contains service id of a particular microservice which is registered in Eureka server. Based on the lowest load factor the request will be forwarded to the particular instance.

*@Configuration*

*public class GatewayConfig {*

*@Autowired*

*private JwtAuthenticationFilter filter;*

*@Bean*

*public RouteLocator configureRoutes(RouteLocatorBuilder builder) {*

*return builder.routes()*

*.route(*

*"productId",*

*r->r.path("/api/product/\*\*")*

*.filters(f -> f.filter(filter))*

*.uri("lb://PRODUCT-SERVICE")*

*)*

*.route("authId",*

*r -> r.path("/auth/\*\*")*

*.filters(f -> f.filter(filter))*

*.uri("lb://AUTH-SERVICE"))*

*.build();*

*}*

*}*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*@Component*

*public class JwtAuthenticationFilter implements GatewayFilter {*

*@Autowired*

*private JwtUtil jwtUtil;*

*@Override*

*public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {*

*ServerHttpRequest request = (ServerHttpRequest) exchange.getRequest();*

*final List<String> apiEndpoints = List.of("/auth/register", "/auth/login");*

*Predicate<ServerHttpRequest> isApiSecured = r -> apiEndpoints.stream()*

*.noneMatch(uri -> r.getURI().getPath().contains(uri));*

*if (isApiSecured.test(request)) {*

*if (!request.getHeaders().containsKey("Authorization")) {*

*ServerHttpResponse response = exchange.getResponse();*

*response.setStatusCode(HttpStatus.UNAUTHORIZED);*

*return response.setComplete();*

*}*

*final String token = request.getHeaders().getOrEmpty("Authorization").get(0);*

*try {*

*jwtUtil.validateToken(token);*

*} catch (JwtTokenMalformedException | JwtTokenMissingException e) {*

*ServerHttpResponse response = exchange.getResponse();*

*response.setStatusCode(HttpStatus.BAD\_REQUEST);*

*return response.setComplete();*

*}*

*Claims claims = jwtUtil.getClaims(token);*

*exchange.getRequest().mutate().header("id", String.valueOf(claims.get("id"))).build();*

*}*

*return chain.filter(exchange);*

*}*

*}*

**How are you handling Session Management ?**

After successful validation of user HttpSession is created at the server side. To make use of it we can use HttpSession as input and use addAttrubute(key,value) method to add key-value pair to the session. After successful login and Id will be created and added to cookie that is JSessionID and sent in the response. Then whenever client send request this cookie is attached along with the request then server validates it with the help of JSessionId and process the request and successful validation.

**What are Solid Principals ?**

It enables us to solve, pre-plan for most of the software design problems or coding problems.

***Single* *Responsibility* *Principle*** -- this states that ***every java class must perform a single functionality***. Implementation of multiple functionalities in a single class mashup the code.

***Achieved*** by Separating/writing the methods in different classes.

***Open and Close Principle*** – this principle stats that ***according to new requirements the module should be open foe extension but closed for modification***.

***Achieved*** by extending the class and add new methods.

***Leskov’s Substitution*** ***Principle*** – It applies to inheritance in such a way that ***the derived classes must be completely substitutable for their base classes***. In other words, if class A is a subtype of class B, then we should be able to replace B with A without interrupting the behavior of the program.

***Achieved*** by taking methods in different interfaces and then use them in required implementation classes.

***Interface Segregation Principle*** – This states that ***the large interface split into smaller ones***. Because the implementation classes use only the methods that are required. We should not force the client to use the methods that they do not required.

***Achieved*** by separating the methods into different interfaces and we can implement multiple interfaces when required.

***Dependency Inversion Principle*** – this principle states ***that we must use abstraction(abstract classes and interfaces) instead of concrete implementation***. High-level modules should not depend on the low-level module but both depend on abstraction.

**What are Cross Site Request Forgery ?**

CSRF : Cross Site Request Forgery

A Request Object/Link/Form created by attacker, executed

by users, then server will execute such request.

**What are cors(Cross Origin Resource sharing) ?**

Cross-Origin Resource Sharing (CORS) is a standard that allows a web page from one domain or origin to access a resource with a different domain or make a cross-domain request

**Difference between SQL and NoSQL?**

SQL is a RDBMS and NoSQL is a non-RDBMS.

SQL stores data in tabular form and NoSQL stores data in Document or graph form.

In SQL data is stored in rows and columns but in NoSQL it is stored in json format.

SQL is vertical scalable and NoSQL is horizontally scalable.

SQL is a stretchered Query Language and No-SQL is a non-strutchered query language.

**Horizontal Scaling** means adding one more instance of new application at new vm

**Vertical Scaling** means enhancing capacity and enhancing hardware configuration of existing system.

**What is SQL Engine ?**

It’s a software that recognizes and interprets SQL commands to access a relational database and interrogate data

**Types of Joins in SQL :**

1.Equals Join

2.Self Join

3.Cross Join

4.Inner Join

5.Outer Join

Left Join

Right Join

Full Join

Business Delegator Pattern :

Dynamic Method Overloading :

Method overloading :

Method overriding :

**How Message Queues Work ?**

We cannot create object for abstract class. But we can create reference variable which can be referred to the object of derived class.

**What is normalization ?**

**Anomalies are the flaw/errors that occurs because of poor planning and storing everything in a single table.**

**Three types of anomalies.**

1. Insertion
2. Deletion
3. Updation

Types of nomal forms:

1NF, 2NF, 3NF, BCNF, 4NF, 5NF

2NF,3NF,BCNF depends on keys and functional dependency

4NF depends on keys and mulivalues dependency

Every high level is a subset of low level

**1NF:**

It is a property of a relation in RDB.

All attribures/fields in a relation/table must have only atomic/individual values/domains.

**Requirements:**

1. Each table should have primary key(which can uniquely identify a record).
2. In each column of a table values are atomic/individual(no multivalues are allowed)

**Ex:** allowed Bhanu

not allowed Bhanu,Priya

1. No repeating groups(same info in different columns/attributes will not be allowed)

Ex: C1 C2 C3

RAM RAM RAHIM

Table should have primary key. Multiple values not allowed. Repetition of information is not allowed.

**2NF:**

**Requirements:**

First, table must be in 1NF.

All non-key/non-prime attributes are fully functional dependents on the PK.

Key attribute or prime attribute : An attribute which is a part of PK is called key/prime key attributes

An attribute which is not a part of PK is called non-key attribute or non-prime attribute.

Functional Dependencies:

It is a relationship that exists when one attribute uniquely determines another attributes.

**3NF:**

First table must be in 2NF.

It should not contains any Transitive Dependency

To achieve data integrity we reduce the data duplication.

All non-prime attributes of a table must be functionally dependent on candidate key

Non-prime attributes should not have inter-dependency.

What is Transitive dependency?

If a->b (b dependent of a)

b->c (c dependent on b so c dependent of a also. We should avoid this) then a->c we should eliminate this.

**BCNF: (Boyce codd normal form) or 3.5 NF :**

First it should be in 3NF.

For every functionally dependent A->B (A should be super key of a table).

To el

It is the process of organizing data in the database.

**Used to reduce data redundancy(data repetition in db tables or data duplications is called redundency) we go with normalization. Redudce redundancy to make to inconsistent into consistent data.**

**Difference between topic and queue in Message queues:**

Queue is used when there is only one consumer.

Topic is used when there is multiple consumers.

It is specified in the code by using spring.jsm.pub-sub-domain=false

False means p2p(peer-to-peer communication) and True means pub-sub communication

**ActiveMQ:**

application.properties file :

we need to specify the activemq broker url, username and password

# Connection details with MOM

spring.activemq.broker-url=tcp://localhost:61616

spring.activemq.user=admin

spring.activemq.password=admin

# P2P communication

spring.jms.pub-sub-domain=false

# Destination-name

my.desti.name=my-queue-abc

Produer side we autowire JmsTemplate and invoke send() method and pass two parameters. One is destination name and the second one is the message using session.createTextMessage(message).

@Component

@Slf4j

public class ProducerService {

@Value("${my.desti.name}")

private String destination;

@Autowired

private JmsTemplate template;

public boolean send(String message) {

boolean sent = false;

try {

//destination,MessageCreator

template.send(

destination,

session-> session.createTextMessage(message)

);

log.info("MESSAGE SENT FROM PRODUCER ==> {}", message);

sent = true;

} catch (Exception e) {

e.printStackTrace();

sent = false;

log.error("MESSAGE SENDING FAILED..");

}

return sent;

}

}

At consumer side we use @JmsListener over the method and pass the destination name as parameters and write the method parameter to take the value from destination.

@Component

public class ConsumerService {

@JmsListener(destination = "${my.desti.name}")

public void readMessage(String message) {

System.out.println("At Consumer " + message);

}

}

**Apache Kafka working :**

**Components of kafka:**

**Pipeline in kafka:**

It has multiple brokers but activemq has only one broker.

Even if one broker is down, other broker will work.

Active mq supports tcp only but apache kafka works on any protocol.

Active mq supports both p2p and pub-sub communication but apache kafka supports only Topics not p2p.

But the no of copies are created based on the replication factor.

At producer side:

@Component

@Slf4j

public class ProducerService {

@Autowired

private KafkaTemplate<String, String> template;

@Value("${my.app.kafka.tpc-name}")

private String topic;

public void sendData(String data) {

log.info("SENDING DATA TO KAFKA {}",data);

template.send(topic, data);

}

}

At consumer side:

@Component

@Slf4j

public class ConsumerService {

@Autowired

private MessageStore store;

@KafkaListener(topics = "${my.app.kafka.tpc-name}",groupId = "sample-nit" )

public void readMessage(String message) {

try {

StockQuote quote = JsonUtil.jsonToObject(message);

log.info("Data at Consumer {}",quote);

store.create(quote);

} catch (Exception e) {

e.printStackTrace();

log.error("Problem at consumer {}",e.getMessage());

}

}

}

Classification of annotation in java:

There are two types of annotations in java.

1. Standard annotations
   1. General Purpose annotation
      1. @Override
      2. @SupressWarnings(“unchecked”)
      3. @Depricated
      4. @FunctionalInterface
   2. Meta Annotations – annotations which are used to prepare annotations
      1. @Documented – prepare documentation using doc tool. If we do it all members of class like constuctors, variables, methods are visible in the .html except the annotation. To make the annotation documented we use @Documented annotation over the interface while creating it.

@Documented

* + 1. @Inherited – by default annotation are not available to child class. So to make an annotation visible to child class we use @Inherited over the interface while creating it.

@Inherited

* + 1. @Target -- will tell to which type of programming element it is applicable. Programming elements like class, variable, method etc. It takes an enum as argument that is ElementType.Type for an annotation to be applicable only for classes. ElementType.Filed to be applicable only for variables. ElementType.Method to be applicable only for methods. ElementType.Constructor to be applicable for constructors only.

@Target(ElementType.FIELD)

* + 1. @Retention – It will tell the metadata to be available upto sourcecode/runtime. It takes enum as argument like RelentionPolicy.SOURCE metadata to be application only upto souce code. RetentionPolicy.CLASS metadata to be applicable only upto Class level. RetentionPolicy.RUNTIME meta data to be available upto runtime.

@Retention(RetentionPolicy.SOURCE)

1. Custom Annotation

Created using Meta annotations.

**ACID Properties:**

[ACID Properties | ACID Properties in DBMS | Gate Vidyalay](https://www.gatevidyalay.com/acid-properties/#:~:text=ACID%20Properties%20%7C%20ACID%20Properties%20in%20DBMS%201,simultaneously%20without%20causing%20any%20inconsistency.%204%20Durability-.%20)

Atomicity: This property ensures that either the transaction occurs completely or it does not occur at all.  
Consistency : it ensures that the database remains consistent before and after the transaction.

Isolation: This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.

Durability: This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk

**AWS:**

You can create up to**two hundred (200)** subnets per Amazon VPC\*. You can have up to five (5) Amazon VPC Elastic IP Addresses per AWS account per AWS Region\*.