**ga Microservices architecture patterns ->** Orchestrator Pattern & Bulk Head Pattern -> <https://www.youtube.com/watch?v=xuH81XGWeGQ>

[**https://dzone.com/articles/design-patterns-for-microservices**](https://dzone.com/articles/design-patterns-for-microservices) **=> must read**

### Decomposition Patterns – splitted monolithic into several small individual application by => Decompose by Business Capability, Decompose by Subdomain, Strangler Pattern

### Integration Patterns ->

### API Gateway Pattern -> proxy for all the requests and handle

### Aggregator Pattern -> collected data from different Microservices and aggregate them into single for loading it into the UI

### Client-Side UI Composition Pattern -> angular js to place different section in different places and load them individually ( SPA ).

**How to handle exception from the repository level class and inform it to calling Microservices**

Using @controller Advice we can handle the exception at global application level and return the response entity.

<https://www.toptal.com/java/spring-boot-rest-api-error-handling>

**Configuration in Microservices**

<https://walkingtechie.blogspot.com/2018/06/spring-boot-features.html>

[**https://www.javainuse.com/spring/spring\_cloud\_config\_server\_using\_git**](https://www.javainuse.com/spring/spring_cloud_config_server_using_git)

[**https://www.javainuse.com/spring/cloud-vault**](https://www.javainuse.com/spring/cloud-vault)

**Cascade types in JPA –** all , merge , persist detach

[**https://www.baeldung.com/jpa-cascade-types**](https://www.baeldung.com/jpa-cascade-types)

**Is IoC and dependency injection the same thing?**

**Inversion** of Control(**IoC**) is also known as **Dependency injection** (DI). ... In **Inversion** of Control(**IoC**), instead of an application calling the framework – it is the framework that calls the components specified by the application. **IoC** is also known as **dependency injection** (DI).

**What is IoC container?**

**IoC Container** (a.k.a. DI **Container**) is a framework for implementing automatic dependency injection. ... The **IoC container** creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time.

**Spring provides following two types of containers.**

* BeanFactory **container**.
* ApplicationContext **container**.

<https://malliktalksjava.com/2015/06/20/different-types-of-spring-containers/>

**There are basically two types of dependency injection:**

1. **constructor injection**: the **dependencies** are provided through a class **constructor**.
2. setter **injection**: the client exposes a setter method that the injector uses to **inject** the **dependency**.

[**https://www.baeldung.com/spring-framework-design-patterns**](https://www.baeldung.com/spring-framework-design-patterns)

**Basically Beans are singleton to change them into different scope use this below annotation**

**@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE) annotation**.

* Singleton –

creates a single instance of that bean, and all requests for that bean name will return the same object, which is cached. Any modifications to the object will be reflected in all references to the bean. This scope is the default value if no other scope is specified.

* Prototype - prototype scope will return a different instance every time it is requested from the container.
* Request -  creates a bean instance for a single HTTP request
* Session - creates for an HTTP Session
* Application - creates the bean instance for the lifecycle of a ServletContext
* Websocket - creates it for a particular WebSocket session

**Difference between request scope and prototype scope :**

Prototype creates a brand new instance everytime you call getBean on the ApplicationContext. Whereas for Request, only one instance is created for an HttpRequest. So in a single HttpRequest, I can call getBean twice on Application and there will only ever be one bean instantiated, whereas that same bean scoped to Prototype in that same single HttpRequest would get 2 different instances.

**HttpRequest scope**

Mark mark1 = context.getBean("mark");

Mark mark2 = context.getBean("mark");

mark1 == mark2; //This will return true

**Prototype scope**

Mark mark1 = context.getBean("mark");

Mark mark2 = context.getBean("mark");

mark1 == mark2; //This will return false

Hope that clears it up for you.

**https://www.geeksforgeeks.org/classloader-in-java/**

**N +1 solution problem in hibernate**

**Type erasure ->** in short replacing types with object during runtime

https://www.baeldung.com/java-type-erasure

**Difference between comparator and comparable**

<https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/>

To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if you sorting needs to be done on attributes of different objects, then use Comparator in Java.

**How to store object as a key in hash map**

**Can we store object inside hashmap?**

A simple thumb rule is to use **immutable objects as keys** in a HashMap.

because:

If it were mutable, then the hashcode() value or equals() condition might change, and you would never be able to retrieve the key from your HashMap.

More precisely, class fields that are used to compute equals() and hashcode() should be immutable!

Now, suppose you create your own class:

* To compare two objects of your class you will have to override equals()
* To use it as a key in any Hash based Data structure you will have to override hashcode() (again, keeping immutability in mind)

Remember that if two objects are equal(), then their hashcode() should be equal as well!

**How hash map works internally**

**For searching which list will u use array list or linked list**

ArrayList as it holds index it is easy to search than traversing

**12) Types of garbage collectors**

* Serial Garbage Collector.
* Parallel Garbage Collector.
* CMS Garbage Collector.
* G1 Garbage Collector.

<https://javapapers.com/java/types-of-java-garbage-collectors/>

**13)Generation in a JVM**

**14Permgen in Java 8**

<https://dzone.com/articles/java-8-permgen-metaspace>

**15) Java 8 Concurrency Tutorial: Threads and Executors**

**https://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/**

**16)Java Executor Service examples**

<https://dzone.com/articles/guide-to-java-8-concurrency-using-executors>

**17) exception handling in Springboot Microservices**

<https://www.tutorialspoint.com/spring_boot/spring_boot_exception_handling.htm>

<https://medium.com/@kousikpaul/microservice-global-exception-handling-and-field-validations-3fc955a80692>

Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives compile time error.

**Sorting multiple fields using Java 8**

<https://howtodoinjava.com/java8/sort-stream-multiple-fields/>

**Spring Cloud- REST call using Netflix Feign Client**

<https://www.javainuse.com/spring/spring-cloud-netflix-feign-tutorial>

**Difference between Setter vs Constructor Injection in Spring**  
 <https://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html#ixzz6QmZO1gsI>

**Final Class in java**

<https://stackoverflow.com/questions/5181578/what-is-the-point-of-final-class-in-java>

**Java8**

<https://javaconceptoftheday.com/solving-real-time-queries-using-java-8-features-employee-management-system/>

**Illegal forward reference**

<https://javaconceptoftheday.com/illegal-forward-reference-error-java/>

type casting

<https://javaconceptoftheday.com/type-casting-in-java/>

super and this :

You can’t use super and this keywords in a static method and in a static initialization block even though you are referring static members.

<https://javaconceptoftheday.com/final-keyword-in-java/>

<https://javaconceptoftheday.com/variable-hiding-shadowing-java/>

**Is an exception occurred in one thread causes other threads to terminate.?**

No, exception is thread wise. Only that thread will terminate in which exception has occurred. Other threads will continue to execute.

array size cannot be negative. If you specify array size as negative, there will be no compile time error, but you will get NegativeArraySizeException at run time.

To be :

<https://javaconceptoftheday.com/number-pattern-programs-in-java/>

<https://javaconceptoftheday.com/java-interview-programs-with-solutions/>

<https://javaconceptoftheday.com/java-priorityqueue-example/>

three rules to solve diamond problem in Java 8 :

<https://javaconceptoftheday.com/java-8-interface-changes-default-methods-and-static-methods/>

<https://javaconceptoftheday.com/java-8-streams-beginners-guide/>

<https://www.telerik.com/kendo-angular-ui/components/dateinputs/calendar/week-num-column/>

<https://ej2.syncfusion.com/angular/documentation/calendar/how-to/render-the-calendar-with-week-numbers/>

<https://stackoverflow.com/questions/35474506/angular-2-date-pipe-weeknumber>

<https://angular.io/api/common/DatePipe>

<https://www.npmjs.com/package/current-week-number>

<https://www.geeksforgeeks.org/how-to-get-the-current-weeknumber-of-the-year/>

**How spring boot runs schema.sql and Data.sql on startup and how to avoid that :**

**https://walkingtechie.blogspot.com/2018/12/execute-schema-and-data-sql-on-startup-spring-boot.html#:~:text=in%20production%20environment.-,Initialize%20a%20Database,sql%20and%20data.**

Spring Boot automatically creates the schema of embedded DataSource. You can customize this behaviour using spring.datasource.initialization-mode property. You can always initialize the DataSource regardless of its type setting spring.datasource.initialization-mode=always. The property spring.datasource.initialization-mode supports three values:

* always

Always initialize the datasource.

* embedded

Only initialize an embedded datasource.

* never

Do not initialize the datasource.

By default Spring Boot enables the fail-fast feature of the Spring JDBC initializer. This means application fails to start when scripts causes exception. You can tune that behavior by setting spring.datasource.continue-on-error.

You can **exclude a bean from autowiring** using the autowire-candidate attribute of the <beans/> element. This makes the bean unavailable to the spring container for autowiring.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:annotation-config/>

<bean class="com.memorynotfound.spring.core.autowired.Job" **autowire-candidate="false"**>

<property name="name" value="Java Developer"/>

</bean>

<bean class="com.memorynotfound.spring.core.autowired.Person">

<property name="name" value="John Doe"/>

</bean>

</beans>

The @**Qualifier** annotation is used to resolve the autowiring conflict, when there are multiple beans of same type. The @**Qualifier** annotation can be used on any class annotated with @Component or on methods annotated with @Bean . This annotation can also be applied on constructor arguments or method parameters.

Both @**Service** and @**Repository annotations** are the specializations over the @Component **annotation**. @**Repository is** a stereotype used **for** persistence layer. It translates any persistence related exceptions into a Spring's DataAccessException. @**Service is** used **for** the beans at the **service** layer.

**Parallel streams** divide the provided task into many and run them in different threads, utilizing multiple cores of the computer. On the other hand sequential **streams** work just like for-loop using a single core

**named query :**

I believe you are talking about Hibernate.In simple terms, a named query is a query that can be identified by a name. You could define a named query as below and use it by its name.

@NamedQuery name="findAllUsers" query="SELECT u FROM Users u"

findByNamedQuery("findAllUsers")

You have more options and can pass in parameters to it as well.

**New Date and time API** -> is located inside java.time package and some of the key classes are the following :

* Instant - It represents a timestamp
* LocalDate - a date without time e.g. 2014-01-14. It can be used to store birthday, anniversary, date of joining etc.
* LocalTime - represents time without a date
* LocalDateTime - is used to combine date and time, but still without any offset or time-zone
* ZonedDateTime - a complete date-time with time-zone and resolved offset from UTC/Greenwich
* How to check for recurring events e.g. birthday in Java 8  
    
  Read more: <https://javarevisited.blogspot.com/2015/03/20-examples-of-date-and-time-api-from-Java8.html#ixzz6Y8OFnVeU>

Stream API new methods like count , joining and distinct explained with example.

<https://www.java67.com/2014/04/java-8-stream-examples-and-tutorial.html>

we call the summaryStatistics(), which returns an instance of an IntSummaryStatistics.  
  
It is this object which provides us utility method like getMin(), getMax(), getSum() or getAverage().

Streams - > Once Terminal operations are used to produce a result, and after that, you cannot reuse them.

**ObjectMapper** :

The simple **readValue** API of the ObjectMapper is a good entry point. We can use it to parse or deserialize JSON content into a Java object.

Also, on the writing side, we can use the **writeValue** API to serialize any Java object as JSON output**.**

<https://www.baeldung.com/jackson-object-mapper-tutorial>

difference between collection and string in java

<https://www.javatpoint.com/difference-between-comparable-and-comparator>

**Will finally block execute when we put return or system.exit() :**

in Java is that finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call System.exit() from try or catch block.

**https://www.java67.com/2012/09/top-10-tricky-java-interview-questions-answers.html**

**Question: Can you override a private or static method in Java?**  
Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, [you can not override a private or static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html), if you create a similar method with same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding.

Similarly, you cannot override a private method in sub class because it's not accessible there, what you do is create another private method with the same name in the child class

1. When doesn't Singleton remain Singleton in Java?

scenario when your Singleton doesn't behave like.

* Multiple Singletons in Two or More Virtual Machines.
* Multiple Singletons Simultaneously Loaded by Different Class Loaders.
* Singleton Classes Destroyed by Garbage Collection, then Reloaded.
* Purposely Reloaded Singleton Classes.
* Copies of a Singleton Object that has Undergone Serialization and Deserialization

1. is it possible to load a class by two ClassLoader?

* the same **class** can be loaded twice by **two** different **ClassLoader** instances.

1. is it possible for equals() to return false, even if contents of two Objects are same?

* Yes incase we have the same content with different instance, eg two strings with same content but different instances.

1. Why compareTo() should be consistent to equals() method in Java?

<https://dzone.com/articles/the-hidden-contract-between-equals-and-comparable>.

Not mandatory , but it will be efficient if compareTo also used for checking equality in both object level and content level.

1. When do Double and BigDecimal give different answers for equals() and compareTo() == 0.
2. What happens when an exception is thrown by a Thread?

When a task throws an **exception** that it does not handle, the runtime prints the **exception** stack trace to the console and then returns the **thread** to the **thread** pool. There is no such thing as an unhandled **exception** on a **thread** created with the Start method of the **Thread** class.

1. Difference between notify() and notifyAll() call?
2. Difference between System.exit() and System.halt() method?
3. Does following code legal in Java? is it an example of method overloading or overriding?

public String getDescription(Object obj){

return obj.toString;

}

public String getDescription(String obj){

return obj;

}

and

public void getDescription(String obj){

return obj;

}

When would you use a **singleton**?

It is used **to** provide global point of access **to** the object. In terms of practical **use Singleton** patterns are used in logging, caches, thread pools, configuration settings, device driver objects. Design pattern is often used in conjunction with Factory design pattern.

**What is difference between == equals () and compareTo () method?**

**equals()** checks if two objects are the same or not and returns a boolean. **compareTo()** (from interface Comparable) returns an integer. It checks which of the two objects is "less than", "**equal** to" or "greater than" the other. ... Note that **equals()** doesn't define the ordering **between** objects, which **compareTo()** does

**Difference between equals method and "==" operator in Java - Interview Question**  
  
Read more: <https://javarevisited.blogspot.com/2012/12/difference-between-equals-method-and-equality-operator-java.html#ixzz6YB5ymSdd>

== can be used for primitive types.

.equals for object type.

<https://prismoskills.appspot.com/lessons/Java/Chapter_03_-_Hashcode_Equals_and_CompareTo.jsp>

Difference between hasnext and next();

<https://dzone.com/articles/how-hashmap-works-internally-in-java#:~:text=HashMap%20in%20Java%20works%20on,key%20and%20value%20in%20HashMap>.

<https://www.javatpoint.com/collections-in-java>

What is the difference between peek(),poll() and remove() method of the Queue interface?

Both poll() and remove() method is used to remove head object of the Queue.

The main difference lies when the Queue is empty(). If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException . peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

Set – may have one null elements

Hashset – will have only unique elements

**Methods inside entryset**

https://www.java67.com/2016/05/keyset-vs-entryset-vs-values-in-java-map-example.html

**Fail fast and fail safe iterator**

[Iterators](https://contribute.geeksforgeeks.org/iterators-in-java/) in java are used to iterate over the Collection objects.Fail-Fast iterators immediately throw ConcurrentModificationException if there is **structural modification** of the collection. Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on **ArrayList, HashMap** classes are some examples of fail-fast Iterator.

**Fail-Safe iterators** don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on **CopyOnWriteArrayList, ConcurrentHashMap** classes are examples of fail-safe Iterator.

https://www.geeksforgeeks.org/fail-fast-fail-safe-iterators-java/#:~:text=Difference%20between%20Fail%20Fast%20Iterator,as%20the%20fail%2Dsafe%20iterator.

https://beginnersbook.com/java-collections-tutorials/

**Immutable classes in Java -> how to create them**

<https://www.geeksforgeeks.org/create-immutable-class-java/#:~:text=Immutable%20class%20means%20that%20once,own%20immutable%20class%20as%20well>.

Singleton class in java

Return type of comparator and comparable

Comparable and comparator

Equal and hashcode

### **Difference between wait() and sleep()**

* The fundamental difference is that wait() is from Object and sleep() is a static method of Thread.
* The major difference is that wait() releases the lock while sleep() doesn’t release any lock while waiting.
* wait() is used for inter-thread communication while sleep() is used to introduce a pause on execution, generally.
* wait() should be called from inside synchronise or else we get an IllegalMonitorStateException, while sleep() can be called anywhere.
* To start a thread again from wait(), you have to call notify() or notifyAll(). As for sleep(), the thread gets started after a specified time interval.

### **Similarities**

* Both make the current thread go into the **Not Runnable** state.
* Both are **native** methods.

Iterate and store only unique values from an array of integers

No of occurrences of each value in a array

Second highest number from a table and array of integer.

Difference between thread.start() and Thread.run() :

Start – will internally call run , create a new thread each time when a this method is invoked.

Run -> if we directly run the thread.run method it is invocated as normal method and no new thread will be created.

Main difference is that when program calls start() method a **new Thread**is created and code inside run() method is executed in new Thread while if you call run() method directly **no new Thread is created** and code inside run() will execute on **current Thread**

If you want to perform a time-consuming task than always call start() method otherwise your [main thread](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) will be stuck while performing a time-consuming task if you call run() method directly.   
  
Another**difference between start vs run in Java thread** is that you **can not call start() method twice**on thread object. once started, second call of start() will throw IllegalStateException in Java while you can call run() method twice  
  
<https://javarevisited.blogspot.com/2012/03/difference-between-start-and-run-method.html#ixzz6Ypt3IAYd>

<https://www.geeksforgeeks.org/private-constructors-and-singleton-classes-in-java/?ref=lbp>

**Array list Thread Safe ?**

Since it is unsynchronized it is not thread safe and to make it thread safe we can use

1) CopyOnWriteArraylist

2) Collections.synchronizedList(new ArrayList<YourClassNameHere>())

concurrent hashmap vs synchronized hashmap

diff bw design pattern

serilisation do we need serizilization key by default and what happens if we leave it as it is

extending interface. -> when we use an interface and add new features to it so that it will not be having issue in all the subsequent classes

asynchronous

diff bw functional interfaces

chess board problem

stream feautreus

Springboot -> factory and single

@controller,@repository , @service

JWT usage

How to use diff server default

How many thread can a tomcat server create when handling a request is there any modify it handle more requests

Service discovery and how other services knows each other

How to handle multiple instances and do load balancing

What if service does goes down

How to handle more no of timeouts.

Microservices fail fast and failsafe

Deployments -> do we need to redeploy for property file changes.

**second highest salary in MySQL**

**SELECT** **MAX**(salary) **FROM** Employee **WHERE** Salary NOT **IN** ( **SELECT** **Max**(Salary) **FROM** Employee);

This will return 200 in our case.  
  
Here is another solution which uses sub query but instead of IN clause it uses < operator

**SELECT** **MAX**(Salary) **From** Employee **WHERE** Salary **<** ( **SELECT** **Max**(Salary) **FROM** Employee);

<https://www.java67.com/2015/01/second-highest-salary-in-mysql-and-sql-server.html>

select DISTINCT(salary) from employee order by salary desc limit 1,1

limit 0,1 - Top max salary ,

limit 1,1 - Second max salary,

limit 2,1 - Third max salary,

limit 3,1 - Fourth max salary

https://stackoverflow.com/questions/21520038/find-max-and-second-max-salary-for-a-employee-table-mysql

hibernate object life cycle

difference between @Requestparam( data after ? mark and have a specific name)

and path variable – it is the data that is present in the url before the question mark.

1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects
3. **Stack:** Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

**What is the difference between @entity and @table in hibernate?**

**Entity** means the class which you will use in your program and **Table** means the actual Database **table** that you will access through your program. **Hibernate** being an ORM requires you to declare the relationship **between** the **tables** and the classes mapping to them.

<https://stackoverflow.com/questions/3126769/uniqueconstraint-annotation-in-java>

@Entity

@Table(uniqueConstraints={@UniqueConstraint(columnNames={"username"})})

public class SomeEntity {

@Column(name = "username")

public String username;

}

Example 2:

@Entity

@Table(uniqueConstraints={@UniqueConstraint(columnNames = {"id\_1" , "id\_2"})})

public class class\_name {

@Id

@GeneratedValue

public Long id;

@NotNull

public Long id\_1;

@NotNull

public Long id\_2;

}

**Map multiple table inside an entity -> one to one mapping example:**

https://thorben-janssen.com/hibernate-tips-how-to-map-an-entity-to-multiple-tables/

@Entity

// first table

@Table(name = "author")

// second table with id as foreign key

@SecondaryTable(name = "author\_details")

public class Author {

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    @Column(updatable = false, nullable = false)

    private Long id;

    @Version

    private int version;

    private String firstName;

    private String lastName;

    @Column(table = "author\_details") // specify the table name of the //secondary table

    private String pseudonym;

    @Column(table = "author\_details")

   private Category category;

    ...

}

**One to many and Many to one Mapping :**

https://www.baeldung.com/hibernate-one-to-many

we have to map the entity with @OneToMany(mappedBy="cart")

@Entity

@Table(name="CART")

public class Cart {

//...

**@OneToMany(mappedBy="cart")**

private Set<Items> items;

// getters and setters

}

where cart is the entity that is mapped with items entity using one to many relationship

@Entity

@Table(name="ITEMS")

public class Items {

**@ManyToOne**

**@JoinColumn(name="cart\_id", nullable=false)**

private Cart cart;

public Items() {}

// getters and setters

}

Here in Join column annotation we need to map the primary key value and put them next to ManyToOne

**Many to one relationship :**

public class ItemsOIO {

@**ManyToOne**

@**JoinColumn**(name = "cart\_id", insertable = false, updatable = false)

private CartOIO cart;

}

public class CartOIO {

@**OneToMany**

@**JoinColumn**(name = "cart\_id") // we need to duplicate the physical information

private Set<ItemsOIO> items;

}

# How to Specify Named Parameters Using the NamedParameterJdbcTemplate

<https://dzone.com/articles/how-to-specify-named-parameters-using-the-namedpar>

# @OnDelete in Hibernate

Fetchtype

Cascade type

# Difference Between get() and load() in Hibernate

In hibernate, get() and load() are two methods which is used to fetch data for the given identifier. They both belong to Hibernate session class. Get() method return null, If no row is available in the session cache or the database for the given identifier whereas load() method throws object not found exception.

https://www.tutorialspoint.com/difference-between-get-and-load-in-hibernate#:~:text=In%20hibernate%2C%20get()%20and,throws%20object%20not%20found%20exception.

Eg:

SessionFactory sessionFactory = HibernateUtil.getSessionFactory(); Session session = sessionFactory.openSession(); Transaction tx = session.beginTransaction(); //Get Example Employee emp = (Employee) session.get(Employee.class, new Long(2)); System.out.println("Employee get called"); System.out.println("Employee ID= "+emp.getId()); System.out.println("Employee Get Details:: "+emp+"\n"); //load Example Employee emp1 = (Employee) session.load(Employee.class, new Long(1)); System.out.println("Employee load called"); System.out.println("Employee ID= "+emp1.getId()); System.out.println("Employee load Details:: "+emp1+"\n"); //Close resources tx.commit(); sessionFactory.close();

**All to-one relationships use FetchType.EAGER and all to-many relationships FetchType.LAZY.**

@OneToMany(mappedBy = “Column\_name” , fetch = FetchType.EAGER)

Internal working of String – Substring() method :

<https://www.geeksforgeeks.org/java-substring-method-memory-leak-issue-and-fix/>

**Problem caused by substring() in JDK 6**  
This method works well with small Strings. But when it comes with taking substring() from a String with more characters, it leads to serious memory issues if you are using JDK 6 or below.

In Java, **final** is the access modifier which can be used with a filed class and a method.

* When a method if final it cannot be overridden.
* When a variable is final its value cannot be modified further.
* When a class is final it cannot be extended.

## **Declaring final variable without initialization ?**

If you declare a final variable later on you cannot modify or, assign values to it. Moreover, like instance variables, final variables will not be initialized with default values.

Therefore, it is mandatory to initialize final variables once you declare them.

**Can a static method be overridden in Java, or *can you override and overload static method in Java***

The answer is, No, [you can not override static method in Java](http://java67.blogspot.com/2012/08/can-we-override-static-method-in-java.html), though you can declare a method with the same signature in a subclass. It won't be overridden in the exact sense, instead, that is called **method hiding**. But at the same time, you can overload static methods in Java, there is nothing wrong declaring static methods with the same name, but different arguments. Sometime interviewer also ask, *Why you cannot override static methods in Java*? The answer to this question lies in the time of resolution.

**static block in Java**

A **static block** in Java is a block of code that is executed at the time of loading a class for use in a Java application. It starts with a 'static {' and it is used for initializing static Class members in general — and is also known as a 'Static Initializer'. The most powerful use of a static block can be realized while performing operations that are required to be executed only once for a Class in an application lifecycle.

Also, static blocks are executed before constructors.

**What if we want to execute some code for every object?**  
We use [Initializer Block in Java](https://www.geeksforgeeks.org/g-fact-26-the-initializer-block-in-java/)

Initializer block contains the code that is always executed whenever an instance is created. It is used to declare/initialize the common part of various constructors of a class.

**Instance Initialization Blocks**

* Instance Initialization Blocks run every time a new instance is created.
* Initialization Blocks run in the order they appear in the program
* The Instance Initialization Block is invoked after the parent class constructor is invoked (i.e. after super() constructor call)

# CountDownLatch in Java

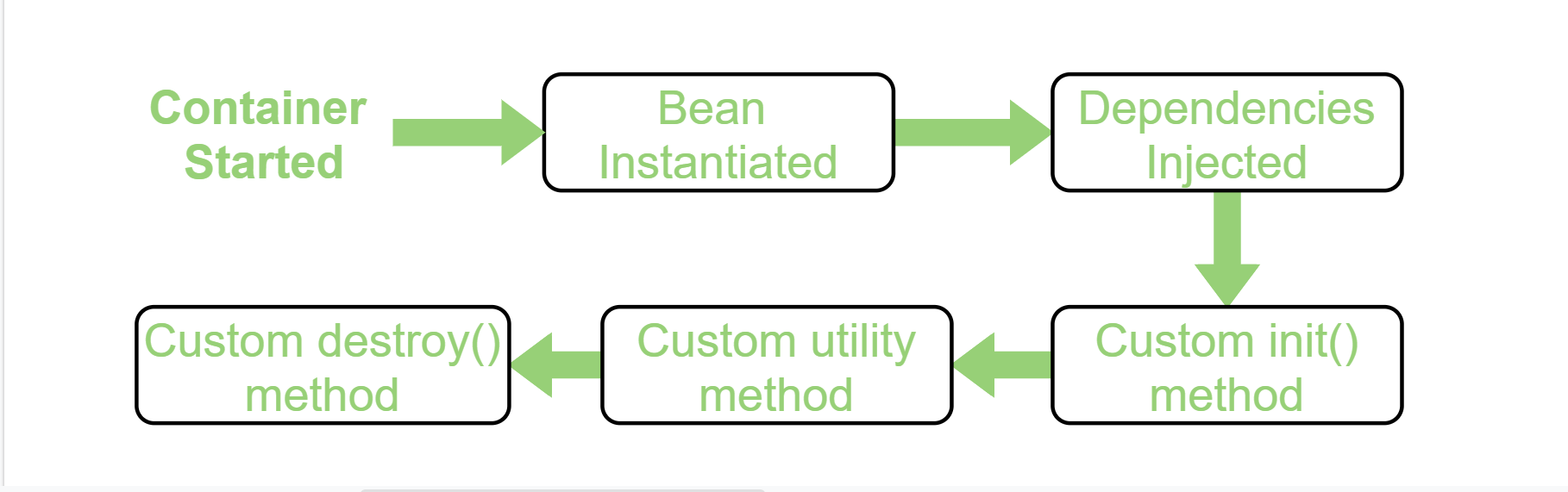
When we create an object of CountDownLatch, we specify the number of threads it should wait for, all such thread are required to do count down by calling CountDownLatch.countDown() once they are completed or ready to the job. As soon as count reaches zero, the waiting task starts running.

<https://www.geeksforgeeks.org/countdownlatch-in-java/?ref=lbp>

**Primary and Qualifier annotation**

https://www.baeldung.com/spring-qualifier-annotation#:~:text=%40Qualifier%20vs%20%40Primary,the%20same%20type%20are%20present.

**Executor service in Multithreading**

**Bean Life cycle :**

**Distinct() :**

**To filter out only unique elements in a stream and also we can obtain the count of the using count();**

**Internal working of hashmap:**

<https://www.geeksforgeeks.org/internal-working-of-hashmap-java/>

write a program to handle string , integer and integer array comparator in java

write a program to print the below format when user enter 4

1

3\*2

4\*5\*6

10\*9\*8\*6

Whole minute dialemma -> create pair of integer from a array list which is divisible by 60

Similarly create a program to get value whose weight is K.

Create list of palindrome string from the provided palindrome string

{abba} => a , b , b , a, abba => count is 5

**How does hashing takes place**

<https://www.geeksforgeeks.org/java-concurrency-yield-sleep-and-join-methods/?ref=lbp>

<https://www.geeksforgeeks.org/implement-runnable-vs-extend-thread-in-java/?ref=rp>

difference between classnotfound and noclassdef

**ClassNotFoundException** is an exception that occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath.

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

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

1. **public** **class** InternExample{
2. **public** **static** **void** main(String args[]){
3. String s1=**new** String("hello");
4. String s2="hello";
5. String s3=s1.intern();//returns string from pool, now it will be same as s2
6. System.out.println(s1==s2);//false because reference variables are pointing to different instance   🡺 false
7. System.out.println(s2==s3);//true because reference variables are pointing to same instance  🡺 True
8. }}

FDC design pattern :

Decomposition Patterns -> Decompose by Business Capability

### **Integration Patterns** ->

### API Gateway Pattern (APIGEE gateway to handle request and authorization and authentication), Aggregator Pattern ( combine the data into a pattern to show them in front end (i.e.,) converting common DTO based on request )

### Client-Side UI Composition Pattern ( application is divided into several frames and places properly with SPA).

### **Database Patterns** -> Shared Database per Service ( a common database across all services )

## **Observability Patterns** -> log aggregation ( service / instance level inside cloud watch log ) , distributed tracing ,health check and performance metrics ( data dog ) .

### **Cross-Cutting Concern Patterns** -> external configuration of data using amazon parameter store , Service Discovery Pattern -> route 53 used service discovery to maintain health check

### Circuit Breaker Pattern -> not sure in our application

### Blue-Green Deployment Pattern -> deployement swap of container for zero or very minimal outage

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

It stores **key**-**value** pairs similar to like HashMap. It **allows** only distinct keys. **Duplicate** keys are not possible. It cannot have null **key** but **can** have multiple null **values**.

**HashMap** doesn't **allow duplicate keys** but allows **duplicate** values. ... **HashMap** allows null **key** also but only once and multiple null values. This class makes no guarantees as to the order of the map; in particular, it does **not** guarantee that the order will remain constant over time.

**Final Object declaration scenario :**

When we declare an object as final we can set the internal data , but we cannot reference them to new object.

**Possible scenario :**

final A ob = new A();

ob.setI(6)

**Not possible:**

final A ob = new A();

ob = new A();

**Karate** is an open-source general-purpose test-automation **framework** that can script calls to HTTP end-points and assert that the JSON or XML responses are as expected. ... **Karate** is implemented in Java but test-scripts are written in Gherkin since **Karate** was originally an extension of the

Cucumber **framework**.

## There are three kinds of **directives in Angular**: **Components**—**directives** with a template. Structural **directives**—change the DOM layout by adding and removing DOM elements. Attribute **directives**—change the appearance or behavior of an element, **component**, or another **directive**.

<https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/>

From Java 8 onward, Java has started using Self Balancing BST instead of linked list for chaining. The advantage of self balancing bst is, we get worst case (when every key maps to same slot) search time as O(Log n)

As it is told that HashMap is unsynchronized i.e. multiple threads can access it simultaneously. If multiple threads access this class simultaneously and at least one thread manipulates it structurally then it is necessary to make it synchronized externally. It is done by synchronizing some object which encapsulates the map. If No such object exists then it can be wrapped around Collections.synchronizedMap() to make HashMap synchronized and avoid accidental unsynchronized access. As in the following example:

Map m = Collections.synchronizedMap(new HashMap(...));

geeksforgeeks.org/hashset-in-java/

<https://www.geeksforgeeks.org/applications-of-hashing/>

<https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/> -> used inside treemap to avoid duplicates and sorting order

<https://howtodoinjava.com/java-interview-puzzles-answers/>

<https://howtodoinjava.com/java-examples/examples/>

<https://howtodoinjava.com/java-concurrency-tutorial/>

<https://howtodoinjava.com/java/serialization/java-serialization/>

**Why iterator is preferred instead of for each ?**

In for-each loop, we can’t modify collection, it will throw a [ConcurrentModificationException](https://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html)on the other hand with iterator we can modify collection.

Modifying a collection simply means removing an element or changing content of an item stored in the collection. This occurs because for-each loop implicitly creates an iterator but it is not exposed to the user thus we can’t modify the items in the collections.

**Rehashing** of a hash map is done when the number of elements in the map reaches the maximum threshold value. ... Basically when **rehashing** occurs the number of buckets are approximately doubled and hence the new index at which the value has to be put changes

**Auto boxing and unboxing**

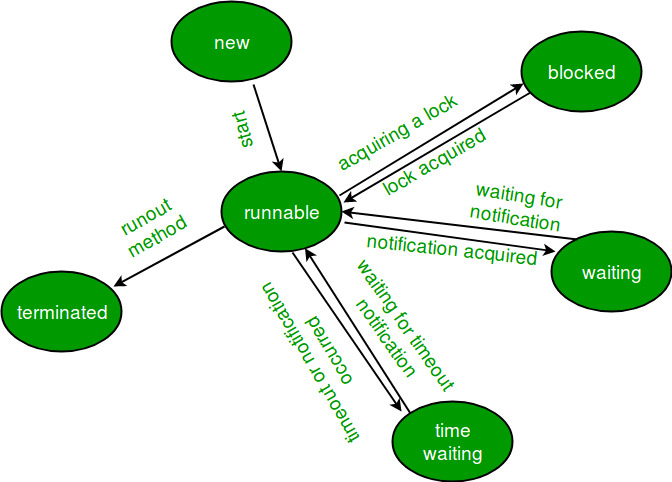
**Autoboxing:**Converting a primitive value into an object of the corresponding [wrapper class](https://www.geeksforgeeks.org/wrapper-classes-java/) is called auto boxing. For example, converting int to [Integer class](https://www.geeksforgeeks.org/wrapper-classes-java/). The Java compiler applies autoboxing when a primitive value is:

* Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
* Assigned to a variable of the corresponding **wrapper class**.

**Unboxing:** Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example conversion of [Integer](https://www.geeksforgeeks.org/wrapper-classes-java/) to int. The Java compiler applies unboxing when an object of a wrapper class is:

* Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
* Assigned to a variable of the corresponding **primitive type**.

**Thread life cycle**



**https://www.geeksforgeeks.org/lifecycle-and-states-of-a-thread-in-java/**

Volatile variable -> mainly used to replace synchronized where visibility alone is required instead of mutual exclusion (Locking a shared resource for modifying)

volatile is a keyword. volatile forces all threads to get latest value of the variable from main memory instead of cache. No locking is required to access volatile variables. All threads can access volatile variable value at same time.

**Transient Variable** -> https://www.geeksforgeeks.org/transient-keyword-java/?ref=lbp

Reading files in java and Types file input reader :

<https://www.baeldung.com/reading-file-in-java>

<https://mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>

Inputstream vs filestream

Buffer reader and buffer writer

@primary and @qualifier

Spring bean life cycle

Circular dependency injection

Features of java 7

Arraylist as readonly

Executor and submit

Callable and runnable

How multithreading after java 5

Input output

Aot and jit

Component data passing

Auto closable

Exception handling from static

https://www.baeldung.com/circular-dependencies-in-spring

Configure email from java

How to handle deadlock with database resource

How to use idle connection closed

Closed in java appln side but idle and available at db end

How to reuse them or make them available during no connection issue

Hibernate

Young generation memory vs old generation memory

How to increase heap size if no memory exception occurs

https://youtu.be/g7bpa9h20jA

While a static class allows only static methods and and you cannot pass static class as parameter. A Singleton can implement interfaces, inherit from other classes and allow inheritance. While a static class cannot inherit their instance members. So Singleton is more flexible than static classes and can maintain state.

Save and persist

Advice in springboot

Criteria api

Procedure call from jdbc template

Index

Synchronisation and concurrency

Young generation vs old generation memory

How to hold another to update the database table when one thread updates/modifies the data

**two important features of locks and synchronization :**

1. **Mutual Exclusion:** It means that only one thread or process can execute a block of code (critical section) at a time.
2. **Visibility**: It means that changes made by one thread to shared data are visible to other threads.

**Atomic classes vs Wrapper Classes**

The **atomic classes** are mutable, but have strong memory consistency guarantees with regard to modifications. So they serve a different purpose from the immutable wrapper classes.

The real advantage of the Atomic\* classes is that they expose an atomic [compare-and-swap](https://en.wikipedia.org/wiki/Compare-and-swap) method, which can be very useful for implementing [lock-free algorithms](https://en.wikipedia.org/wiki/Non-blocking_algorithm).

Like many intermediate to advanced concurrency tools, if you can't imagine why you would need such a thing then you probably shouldn't try to use them. If you stick to immutability or explicit locking everywhere then you probably won't need atomics.

**Checked exception in java**

checked exceptions gets checked during compile time. If we didn’t handled/declare the exceptions, our program gave the compilation error.

Here are the few other Checked Exceptions –

* SQLException
* IOException
* ClassNotFoundException
* InvocationTargetException

**Unchecked exceptions** are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **RuntimeException** class.

Here are the few unchecked exception classes:

* NullPointerException
* ArrayIndexOutOfBoundsException
* ArithmeticException
* IllegalArgumentException
* NumberFormatException

**Dynamic Method Dispatch:**

Method overriding is one of the ways in which Java supports Runtime Polymorphism. **Dynamic method dispatch** is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

**Default methods in Interface :**

Interfaces can have default methods with implementation in Java 8 on later.

1. Interfaces can have static methods as well, similar to static methods in classes.
2. Default methods were introduced to provide backward compatibility for old interfaces so that they can have new methods without affecting existing code.

**Reflection** is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

* The required classes for reflection are provided under java.lang.reflect package.
* Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
* Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

geeksforgeeks.org/reflection-in-java/#:~:text=Reflection%20is%20an%20API%20which,reflect%20package.&text=Through%20reflection%20we%20can%20invoke,access%20specifier%20used%20with%20them.

[https://dzone.com/articles/**5-different-ways-to-create-objects**-in-java-with-ex](https://dzone.com/articles/5-different-ways-to-create-objects-in-java-with-ex)

**Microservices communication types :**

<https://blog.logrocket.com/methods-for-microservice-communication/>

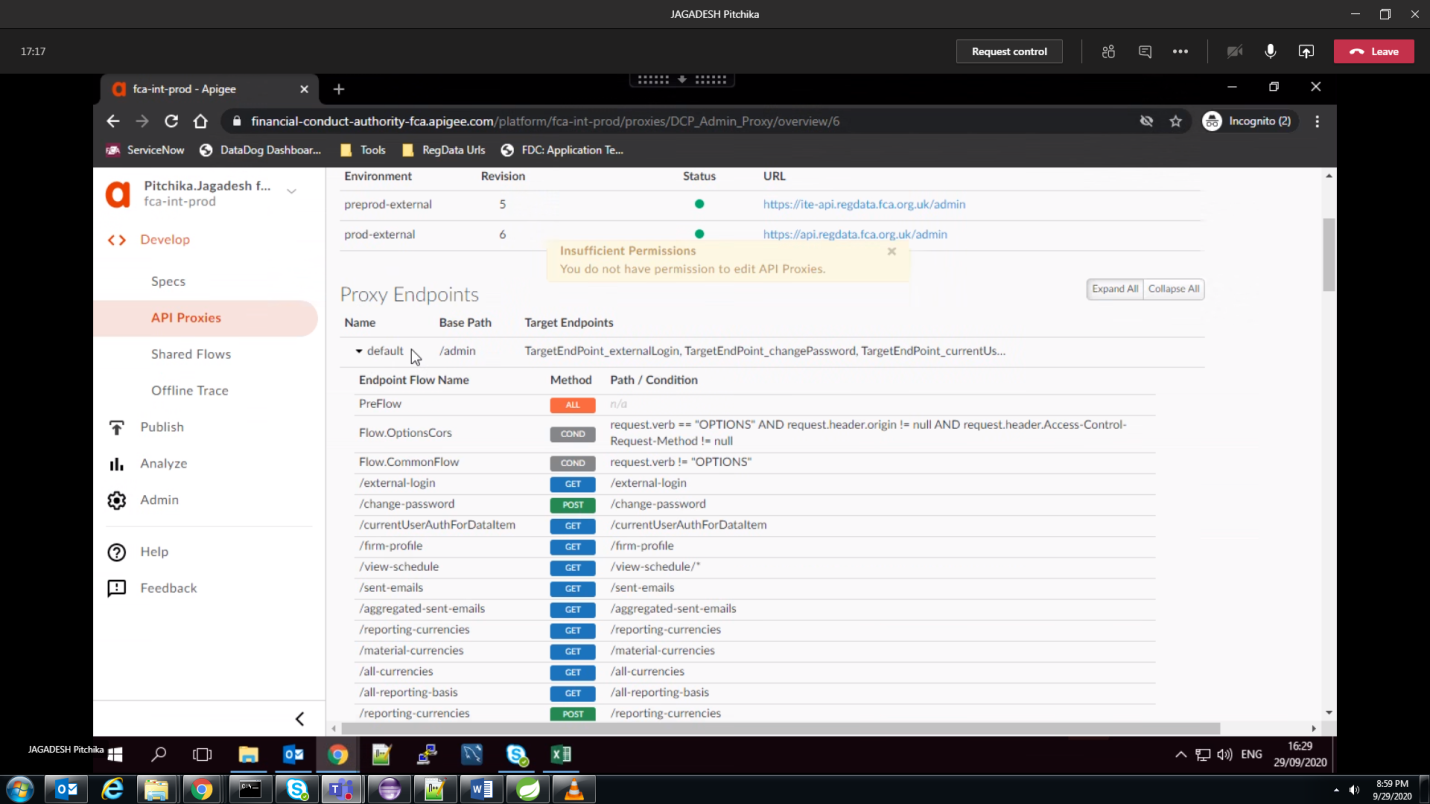
**@import annotation in spring boot :**

It is replacement for @configuration ,in our application we use them on the top of @springbootApplication to set the data source.

We are currently loading the data from the param store -> PlatformCoreParamStore -> initParameterStore method.

Using this the data loaded based on the environment and using the same by **@value** annotation

LinkedMultiValueMap and MultiValueMap



RestTemplate.PostForEntity() ->

<https://www.baeldung.com/rest-template>

Exchange api in rest template ?

ResponseEntity<byte[]> result =

restTemplate.exchange("http://localhost:7070/spring-rest-provider/krams/person/{id}",

HttpMethod.GET, entity, byte[].class, id);

We have the following:

* A [GET](https://tools.ietf.org/html/rfc7231#section-4.3.1) request will be performed to the given URL sending the HTTP headers that are wrapped in the [HttpEntity](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/HttpEntity.html) instance.
* The given URL contains a template variable ({id}). It will be replaced with the value given in the last method parameter (id).
* The response entity will be returned​ as a byte[] wrapped into a [ResponseEntity](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html) instance.

<https://www.softwaretestinghelp.com/java-coding-interview-programs/>

<https://www.journaldev.com/370/java-programming-interview-questions>

<https://javahungry.blogspot.com/2018/09/70-top-java-interview-programs-with-coding-solutions.html>

<https://javaconceptoftheday.com/java-interview-programs-with-solutions/>

<https://javarevisited.blogspot.com/2017/07/top-50-java-programs-from-coding-Interviews.html>

Java.util.Collections.frequency() in Java

The method is a java.util.Collections class method. It counts the frequency of the specified element in the given list. It [override the equals()](https://www.geeksforgeeks.org/overriding-equals-method-in-java/) method to perform the comparison to check if the specified Object and the Object in the list are equal or not.

Similar to break the for loop in the middle of the iteration

Stream.of("cat", "dog", "elephant", "fox", "rabbit", "duck")

.takeWhile(n -> n.length() % 2 != 0)

.forEach(System.out::println);

<https://www.baeldung.com/java-break-stream-foreach>

A Future represents the result of an asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation.

It has Get() method to retrieve the result of the computation.

interface ArchiveSearcher { String search(String target); }

class App {

ExecutorService executor = ...

ArchiveSearcher searcher = ...

void showSearch(final String target)

throws InterruptedException {

Future<String> future

= executor.submit(new Callable<String>() {

public String call() {

return searcher.search(target);

}});

displayOtherThings(); // do other things while searching

try {

**displayText(future.get()); // use future**

} catch (ExecutionException ex) { cleanup(); return; }

}

}

The [FutureTask](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html" \o "class in java.util.concurrent) class is an implementation of Future that implements Runnable, and so may be executed by an Executor. For example, the above construction with submit could be replaced by:

FutureTask<String> future =

new FutureTask<String>(new Callable<String>() {

public String call() {

return searcher.search(target);

}});

executor.execute(future);

both are same -> instead of executor.submit we use execute method to implement the future task

<https://dzone.com/articles/java-executor-service-types>

how to find third max salary

**select DISTINCT(salary) from employee order by salary desc limit 1,1 – second max salary**

**select DISTINCT(salary) from employee order by salary desc limit 2,1 – third max salary**

**correlated query in sql**

**indexing in sql**

**pagination in Springboot**

**config server**

https://cloud.spring.io/spring-cloud-config/multi/multi\_\_spring\_cloud\_config\_server.html

**distributed tracing**

https://docs.datadoghq.com/tracing/setup/java/?tab=websphere

**feign client**

<https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html>

**how to do connect with 2 datasource / database in Springboot**

<https://springframework.guru/how-to-configure-multiple-data-sources-in-a-spring-boot-application/>

**communicate between components in angular**

<https://medium.com/@liutingchun_95744/angular-four-ways-for-communication-between-components-b743b9653f8>

**promise and observables**

|  |  |
| --- | --- |
| **Observables** | **Promises** |
| Emit multiple values over a period of time. | Emit a single value at a time. |
| Are lazy: they’re not executed until we subscribe to them using the subscribe() method. | Are not lazy: execute immediately after creation. |
| Have subscriptions that are cancellable using the unsubscribe() method, which stops the listener from receiving further values. | Are not cancellable. |
| Provide the map for forEach, filter, reduce, retry, and retryWhen operators. | Don’t provide any operations. |
| Deliver errors to the subscribers. | Push errors to the child promises. |

<https://www.syncfusion.com/blogs/post/angular-promises-versus-observables.aspx>

|  |  |  |
| --- | --- | --- |
| **Operations** | **Observables** | **Promises** |
| Creation | |  |  | | --- | --- | | 1  2  3  4  5 | const obs = new  Observable((observer)  => {    observer.next(10);    }) ; | | |  |  | | --- | --- | | 1  2  3  4  5 | const promise = new Promise(() => {    resolve(10);    }); | |
| Transform | |  |  | | --- | --- | | 1 | Obs.pipe(map(value) =>  value \* 2); | | |  |  | | --- | --- | | 1 | promise.then((value) => value \* 2); | |
| Subscribe | |  |  | | --- | --- | | 1  2  3  4  5 | const sub = obs.subscribe(  (value) => {    console.log(value)    }); | | |  |  | | --- | --- | | 1  2  3  4  5 | promise.then((value) => {    console.log(value)    }); | |
| Unsubscribe | |  |  | | --- | --- | | 1 | sub.unsubscribe(); | | |  |  | | --- | --- | | 1 | Cannot unsubscribe | |

<https://www.javainuse.com/messaging/rabbitmq/listeners>

<https://dzone.com/articles/spring-boot-rabbitmq-tutorial-retry-and-error-hand>

<https://www.javainuse.com/spring/spring-boot-rabbitmq-hello-world>

<https://www.javainuse.com/messaging/rabbitmq/exchange>

<https://dzone.com/articles/spring-boot-rabbitmq-tutorial-retry-and-error-hand>

<https://dzone.com/articles/microservice-architecture-with-spring-cloud-and-do>

<https://www.javacodegeeks.com/2018/12/apigee-an-introduction-to-api-gateway.html>

<https://www.baeldung.com/spring-import-annotation>

<https://www.baeldung.com/spring-component-scanning>

<https://www.baeldung.com/spring-component-repository-service>

<https://javarevisited.blogspot.com/2017/11/difference-between-component-service.html#:~:text=is%20also%20available.-,Difference%20between%20%40Component%2C%20%40Service%2C%20%40,Controller%2C%20and%20%40Repository%20in%20Spring&text=%40Component%20is%20a%20generic%20stereotype,stereotype%20for%20the%20service%20layer>.

<https://medium.com/@karthi.net/docker-tutorial-build-docker-images-using-jenkins-d2880e65b74>

**Amazon Route 53 provides three main functions:**

* **Domain** registration. allows you to register **domain** names.
* **Domain** Name System (DNS) service. translates friendly domains names like www.example.com into IP addresses like 192.0.2.1. ...
* Health checking. can monitor the health of resources such as web and email servers.

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-discovery.html>

<https://softwareengineering.stackexchange.com/questions/337018/java-processing-a-large-file-concurrently>

<https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream>

<https://howtodoinjava.com/java8/read-file-line-by-line/>

<https://dzone.com/articles/batch-processing-large-data-sets-with-spring-boot>

there are twenty user who are accessing an application and an admin who access an application how many instance would be created for them.

<https://www.geeksforgeeks.org/internal-working-of-sethashset-in-java/>

<https://stackoverflow.com/questions/54966111/how-to-store-distinct-characters-of-a-string>

**which data type would you use to store the sensitive information in Java** -> char array should be used in order to keep them separated and to avoid leaking of sensitive data

In this blog post we will talk about **Which Java data type would you choose for storing sensitive information, like passwords, and Why?** Below are the main reason to choose **char** datatype to store sensitive information, like password.

* There is no way to clear a **String** Object from the memory, it’s up to GC to collect it.
* String objects are immutable and stored in a **String Pool**(may reside inside a **PermGen** space) which may not at all be GC’d.
* Any person taking the heap dump can easily see the **String**
* In case of a char array, we can always nullify it once we are done with the information, so not much dependency on the GC, thus we are narrowing the time window for the life of sensitive data.

<http://bytepadding.com/java/java-core/java-8-get-all-unique-characters-from-list-of-string-using-flatmap/>

<https://kkjavatutorials.com/what-are-the-differences-between-32-bit-and-64-bit-versions-of-java/>

We can say that **java is not 100% Object-oriented language** because it utilizes eight kinds of primitive data types and they are – boolean, char byte, int, float, double, long, short. These datatypes are not objects so it would not be wrong to say that Java is not 100% Object oriented language.

<https://kkjavatutorials.com/can-we-call-static-method-with-null-object/>

**can we override a method by only changing return type and also not changing the parameter types ?**

yes this can be done if the overridden method return type is sub type of the return type.

**Can we call an abstract method inside a constructor ?**

There are a few more restrictions that a class must obey to allow inheritance. **Constructors must not invoke overridable methods**, directly or indirectly. If you violate this rule, program failure will result. The superclass constructor runs before the subclass constructor, so the overriding method in the subclass will be invoked before the subclass constructor has run. If the overriding method depends on any initialization performed by the subclass constructor, the method will not behave as expected.

<https://kkjavatutorials.com/what-is-difference-between-using-serializable-externalizable-interfaces-in-java/>

**Can we Override static method in sub class ?**

Even if we use the same method signature in the sub class it does not override the parent class method so overriding is not applicable with static method

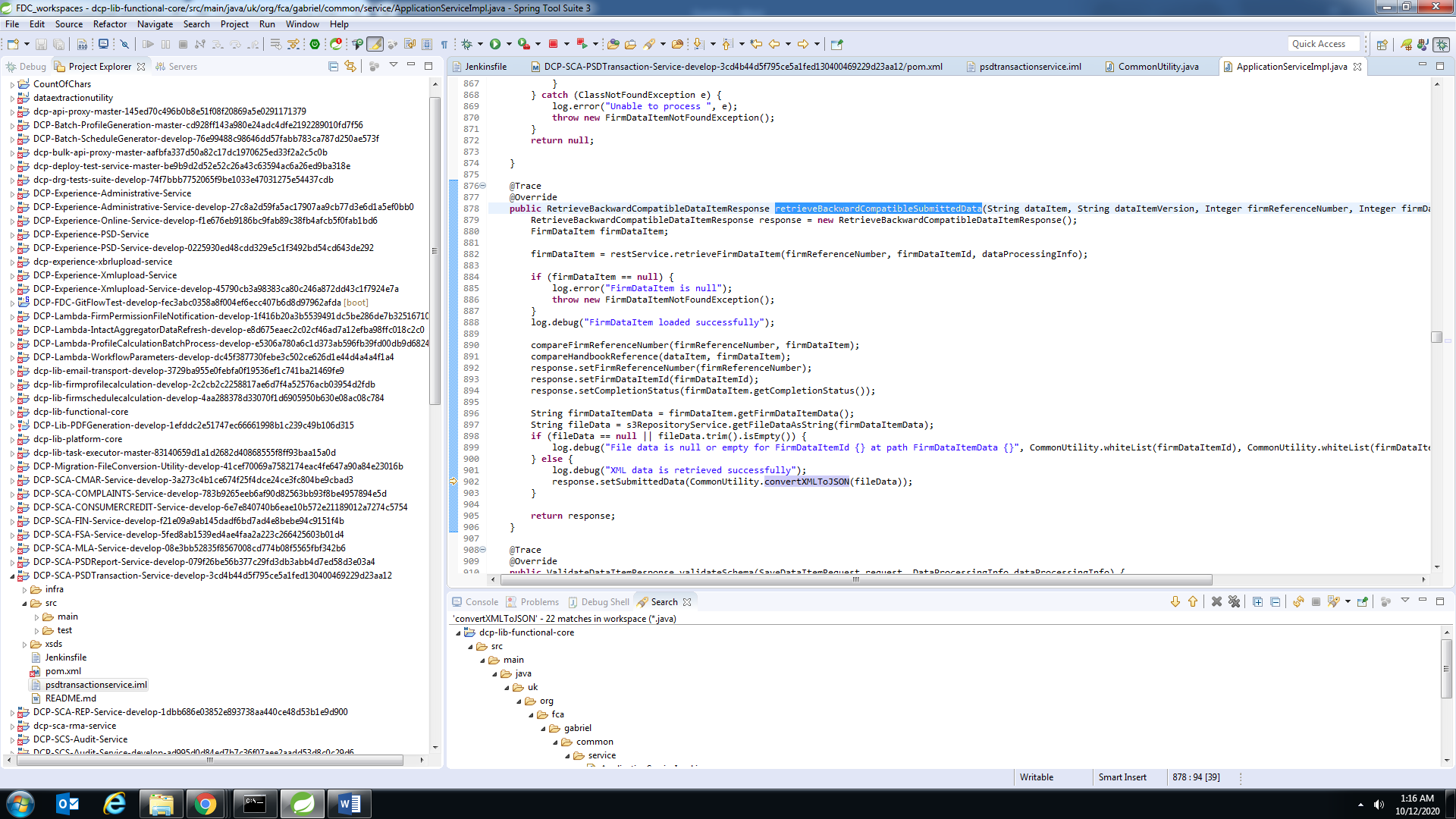
<https://kkjavatutorials.com/can-we-override-static-method-in-java/>

**Maximum occurrence of a given string or string in array of strings**

<https://www.geeksforgeeks.org/frequent-word-array-strings/>

**Using SAX Parser to handle Large files ?**

<https://dzone.com/articles/conveniently-processing-large>



### **ArrayList vs. LinkedList vs. Vector**

From the hierarchy diagram, they all implement List interface. They are very similar to use. Their main difference is their implementation which causes different performance for different operations.  ArrayList is implemented as a resizable array. As more elements are added to ArrayList, its size is increased dynamically. It's elements can be accessed directly by using the get and set methods, since ArrayList is essentially an array. LinkedList is implemented as a double linked list. Its performance on add and remove is better than Arraylist, but worse on get and set methods. Vector is similar with ArrayList, but it is synchronized. ArrayList is a better choice if your program is thread-safe. Vector and ArrayList require space as more elements are added. Vector each time doubles its array size, while ArrayList grow 50% of its size each time. LinkedList, however, also implements Queue interface which adds more methods than ArrayList and Vector, such as offer(), peek(), poll(), etc.    Note: The default initial capacity of an ArrayList is pretty small. It is a good habit to construct the ArrayList with a higher initial capacity. This can avoid the resizing cost.

<https://www.geeksforgeeks.org/check-if-the-given-string-of-words-can-be-formed-from-words-present-in-the-dictionary/?ref=rp>

<https://www.geeksforgeeks.org/count-words-appear-exactly-two-times-array-words/?ref=rp>

<https://www.geeksforgeeks.org/program-to-print-the-given-digit-in-words/?ref=rp>

<https://www.geeksforgeeks.org/print-unique-words-string/?ref=rp>

<https://www.geeksforgeeks.org/print-words-string-reverse-order/?ref=rp>

# [Spring Boot - How to log all requests and responses with exceptions in single place?](https://stackoverflow.com/questions/33744875/spring-boot-how-to-log-all-requests-and-responses-with-exceptions-in-single-pl)

<https://tomgregory.com/building-a-spring-boot-application-in-jenkins/>

<https://www.geeksforgeeks.org/g-fact-24-finalfinally-and-finalize-in-java/>

# Can a bean class with private constructor be instantiated in Spring framework?

Yes, Spring can invoke private constructors and instantiate object. Spring uses the reflection API to set the bean constructor accessible when it find the constructor with the right arguments, regardless of its visibility.

Difference between @Inject and @Autowired in Spring Framework.

@Inject is part of the Java CDI (Contexts and Dependency Injection) standard introduced in Java EE 6.

@Autowired is Spring's own annotation. @Inject is part of CDI that defines a standard for dependency injection similar to Spring.

In any Spring application, the two annotations works the same way as Spring extends its support to CDI.

<https://dzone.com/articles/gateway-pattern>

<https://codepumpkin.com/immutable-class-with-mutable-member-fields-in-java/>

https://codepumpkin.com/immutable-class-with-mutable-member-fields-in-java/

multithreading - The difference between the Runnable and Callable interfaces in Java - Stack Overflow

The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. **A Runnable, however, does not return a result and cannot throw a checked exception.**

<https://javarevisited.blogspot.com/2016/08/useful-difference-between-callable-and-Runnable-in-Java.html>

<https://www.baeldung.com/java-runnable-callable>

<https://www.baeldung.com/spring-beanfactory-vs-applicationcontext>

bean factory – lazy loading , application Context – Eager Loading

*ApplicationContext* supports almost all types of bean scopes, but the *BeanFactory* only supports two scopes — *Singleton* and *Prototype.* Therefore, it's always preferable to use *ApplicationContext*when building complex enterprise applications.

<https://beginnersbook.com/2017/10/java-lambda-expressions-tutorial-with-examples/>

https://www.baeldung.com/spring-jdbc-jdbctemplate

to expire a hashmap content based on timer

<https://crunchify.com/clean-expired-element-from-map-while-adding-elements-at-the-same-time-java-timer-timertask-and-futures-complete-examples/>

*For DB locking to provide consistent update*

<https://dzone.com/articles/concurrency-and-locking-with-jpa-everything-you-ne>

<https://www.javacodegeeks.com/2014/09/a-beginners-guide-to-database-locking-and-the-lost-update-phenomena.html>

<https://www.objectdb.com/java/jpa/persistence/lock>

<https://www.baeldung.com/jpa-pessimistic-locking>

https://www.baeldung.com/java-jpa-transaction-locks

<https://www.java67.com/2013/01/difference-between-callable-and-runnable-java.html>

Difference between @RestController and @Controller Annotation in Spring MVC and REST

https://javarevisited.blogspot.com/2017/08/difference-between-restcontroller-and-controller-annotations-spring-mvc-rest.html

Mainly -> rest controller returns domain object , controller maps for view

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

<https://howtodoinjava.com/java/multi-threading/restart-thread-uncaughtexceptionhandler/>

**What problems will you have if you don't override the hashCode() method?**

<https://www.quora.com/What-problems-will-you-have-if-you-dont-override-the-hashCode-method>

if hashcode not overridden the super method will return the object’s reference ( address ) as the hashcode , so equality could not be checked with equals and hashcode contract.

# Java diamond operator – <> operator in Java

Map<String, Integer> params = new HashMap<>();

Map<String, Integer> params = new HashMap<String, Integer>();

Both are same by the power of <> operator

# Java WatchService Example to Auto Reload Properties

<https://howtodoinjava.com/java7/auto-reload-of-configuration-when-any-change-happen/>

<https://howtodoinjava.com/java/multi-threading/compare-and-swap-cas-algorithm/>

<https://howtodoinjava.com/java/keywords/java-synchronized/>

<https://howtodoinjava.com/java/multi-threading/object-vs-class-level-locking/>

<https://howtodoinjava.com/java/multi-threading/java-runnable-vs-thread/>

<https://howtodoinjava.com/java/multi-threading/wait-notify-and-notifyall-methods/>

<https://howtodoinjava.com/java/multi-threading/difference-between-yield-and-join-in-threads-in-java/>

<https://howtodoinjava.com/java/multi-threading/sleep-vs-wait/>

<https://howtodoinjava.com/java/multi-threading/multithreading-difference-between-lock-and-monitor/>

<https://howtodoinjava.com/java/multi-threading/java-callable-future-example/>

<https://howtodoinjava.com/java/multi-threading/executor-framework-tutorial/>

<https://dzone.com/articles/microservices-vs-soa-whats-the-difference>

1) what ate the securities implemented for ur services

2) how the code review process happens in ur cmpy

3)what are all ur roles and responsibilities

4)How will tou track issue hapens in ur micro service world

5)how to handle exceptions in micro services

6)bean life cycle

7)about config servers and profiles

8)any exposure in no sql servers

9)chanllenges to transfer monolitic application to microservice world

10)abt my currect project architecture

> How hashmap internally work - What is the use of equalsof method

> Executor framework service - Multi threading

> Threadpool Manager

> Callable interface,Runnable interface

> Heap memory,Stack Memory -difference What kind of exception it throws when it exceeds its memory allocation?

When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.

We can use -Xms and -Xmx JVM option to define the startup size and maximum size of heap memory. We can use -Xss to define the stack memory size.

> String is immutable

> Streams functions -map funtions (Transaction objects with many variables - I want to

> What is difference between ArrayList and Link…

Andrew:

- type of applications you had worked

- architecture wise

- exact role in the project

- team and process

Jegan:

- experience in API management tools

- Authentication mechanism

- Merchant

- jwt token

- algorithm for JWT

- TPS processing a day, transactions

- App dynamics

- how is gateway and microservices mapping?

- yaml, swagger and version

- filter & interceptor - usage?

- AOP and transactions - spring config

- required & request – annotation

zenith:

- design patterns

- High level categories of design pattern

- chain of responsibility design pattern?

- features used in java 1.5 - what was used in your project

- auto boxing & unboxing

- Diff between equal, equal to and not equal to method

- diff between primiti…

Ftpclient connection

Socket connection

How u pass xml in socket connection

Java Design patterns

Transactions related questions