**JAVA – INTERVIEW QUESTIONS**

1. **Lamda expression to eliminate duplicate elemts in a list without set**

List<String> duplicates = **new** ArrayList<>();

duplicates.add("a");

duplicates.add("a");

duplicates.add("b");

**//with distinct method**

System.***out***.println(duplicates.stream().distinct().collect(Collectors.*toList*()));

**//with set**  System.***out***.println(duplicates.stream().collect(Collectors.*toSet*()));

1. **What is Prediate Functional Method?**

Predicate is a functional interface with a Boolean valued functional Method Test()

**public** **class** Test {

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

Predicate<String> pr = s -> s.equals("qwerty");

System.***out***.println(pr.test("qweqrty"));

}

}

1. **What is intermedaiate operations and Terminal Operations** ?

**intermedaiate operations:** These **operations** are used to pipeline other methods and to transform into the other streams

* sorted(Comparator<T>)
* peek(Consumer<T>)
* distinct()

**Terminal Operations:**  These **operations** are used to produce results

* forEach.
* count.

List<String> laptopList = **new** ArrayList();

laptopList.add("DELL");

laptopList.add("ACER");

laptopList.add("HCL");

// Intermediate operation

laptopList.sort((p1, p2) -> p1.compareTo(p2));

// Terminal Operation

laptopList.forEach(a -> {

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

});

1. **what are method references ?**

Method reference is used to refer method of functional interface

**interface** Sayable{

**void** say();

}

**public** **class** MethodReference {

**public** **static** **void** saySomething(){

        System.out.println("Hello, this is static method.");

    }

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

        // Referring static method

        Sayable sayable = MethodReference::saySomething;

        // Calling interface method

        sayable.say();

    }

}

1. **can we define user defined functional interface? Use of functional interface**

**YES, The major benefit of java 8 functional interfaces is that we can use lambda expressions to instantiate them and avoid using bulky anonymous class implementation.**

@FunctionalInterface

interface MyFunctionalInterface {

public int addMethod(int a, int b);

}

public class BeginnersBookClass {

public static void main(String args[]) {

// lambda expression

MyFunctionalInterface sum = (a, b) -> a + b;

System.out.println("Result: "+sum.addMethod(12, 100));

}

}

1. **Stream Api?**

Introduced in Java 8, the **Stream API** is used to process collections of objects. A **stream** is a sequence of objects that supports various methods which can be pipelined to produce the desired result

**7)Exception handling in sprng boot?**

**@RestControllerAdvice** is used to handle the exceptions globally

**@EceptionHandler** is used to handle specific exceptions

@ControllerAdvice

public class ProductExceptionController {

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

return new ResponseEntity<>("Product not found", HttpStatus.NOT\_FOUND);

}

}

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

1. **@PathVariable , @PathParam , @RequestParam**

**@PathParam** is a parameter annotation which allows you to map variable URI path fragments into your method call.

@**PathVariable** is to obtain some placeholder from the URI

## What is a Stored Procedure?

## A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

## CREATE PROCEDURE procedure\_name AS sql\_statement GO;

## EXEC procedure\_name;

## Stored Procedure with parameters

## CREATE PROCEDURE SelectAllCustomers @City nvarchar(30) AS SELECT \* FROM Customers WHERE City = @City GO;

## EXEC SelectAllCustomers @City = 'London';

1. **What is cursor? How do we write cursor?**

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

**DECLARE**

   c\_id customers.id%type;

   c\_name customers.**name**%type;

   c\_addr customers.address%type;

**CURSOR** c\_customers **is**

**SELECT** id, **name**, address **FROM** customers;

**BEGIN**

**OPEN** c\_customers;

   LOOP

**FETCH** c\_customers **into** c\_id, c\_name, c\_addr;

      EXIT **WHEN** c\_customers%notfound;

      dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);

**END** LOOP;

**CLOSE** c\_customers;

**END**;

/

1. **Map() vs FlatMap()?**

**Map:** **one-to-one** mapping between input and output element.

List<String> laptopList = **new** ArrayList();

laptopList.add("DELL");

laptopList.add("ACER");

laptopList.add("HCL");

List<List<String>> lists = **new** ArrayList<>();

lists.add(laptopList);

System.***out***.println(laptopList.stream().map(p>p).collect(Collectors.*toList*()));

**FlatMap:** List of Lists

System.***out***.println(lists.stream().flatMap(p>p.stream()).collect(Collectors.*toList*()));

1. **Annotations?**

@**Component** is a generic stereotype for any Spring-managed **component** or bean. @Repository is a stereotype for the persistence layer. @**Service** is a stereotype for the **service** layer. @Controller is a stereotype for the presentation layer (spring-MVC).

1. **Sorting**

**Higher to lower**

laptopList.sort((p1, p2) -> p2.compareTo(p1));

**lower to Higher**

laptopList.sort((p1, p2) -> p2.compareTo(p1));

**13)Log4J and Slf4J?**

**Log4J is the Implementation of logging framework**

**Slf4J** it **is** an abstraction for all those logging frameworks. It is not a logging component instead it is a absraction layer where can implement with the help of other loggings. We cannot compare both.

1. **How to implement Distributed tracing in Spring Boot?**

Distribute tracing provides a place where we can see that "what is happening with a specific request?" It is important because there are a variety of components that are involved in the microservices.

1. **How to find the highest number in the given List?**

employeelist.stream().max(Comparator.*comparing*(Employee::getSal)).

ifPresent(z->System.***out***.println("Maximum string in the set is " + z.getSal()));

**MultiThreading**

* When the thread gets a chance to execute, its target run() method will run.

### **Java Thread Example by extending Thread class**

**class** Multi **extends** Thread{

**public** **void** run(){

System.out.println("thread is running...");

}

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

Multi t1=**new** Multi();   // here obj is a thread

t1.start();

### **Java Thread Example by implementing Runnable interface**

**class** Multi3 **implements** Runnable{

**public** **void** run(){

System.out.println("thread is running...");

}

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

Multi3 m1=**new** Multi3();  // this is not thread obj

Thread t1 =**new** Thread(m1);   // Making it as a thread onj

t1.start();

 }

}

1. **How do u convert JSON to POJO /Vice Versa?**

The **ObjectMapper** class of the Jackson API in Java provides methods to convert a Java object to JSON object and vice versa.

The **writeValueAsString()** method of this class accepts a JSON object as a parameter and returns its respective JSON String

<dependency>

    <groupId>com.fasterxml.jackson.core</groupId>

    <artifactId>jackson-core</artifactId>

    <version>2.6.4</version>

</dependency>

* Use objectMapper.writeValueAsString() method to convert pojo to JSON

**Json to pojo**

//JSON input

String json = "{\"id\":1,\"name\":\"Lokesh Gupta\",\"age\":34,\"location\":\"India\"}";

**//Object mapper instance**

**ObjectMapper mapper = new ObjectMapper();**

**//Convert JSON to POJO**

**Employee emp = mapper.readValue(json, Employee.class);**

1. **How to return Response type as Xml/plain Txt in controller?**

**Use Produces=”application/xml”(add @XmlRootelement for the response model) or “Plain/text”**

**By default spring boot provides json**

@Component

//@Scope("prototype")

@XmlRootElement

**public** **class** Test {

**private** String car;

**private** String engin

1. @GetMapping(value="/sendMsg" ,produces="application/xml")
2. **public** Test sendMsg(@RequestParam(name="comment") String msg)
3. {
4. Test t = **new** Test();
5. t.setCar(msg);
6. t.setEngine(msg);
7. **return** t;
8. }

16) **How to convert Java Pojo to json in java?**

<dependency>

    <groupId>com.fasterxml.jackson.core</groupId>

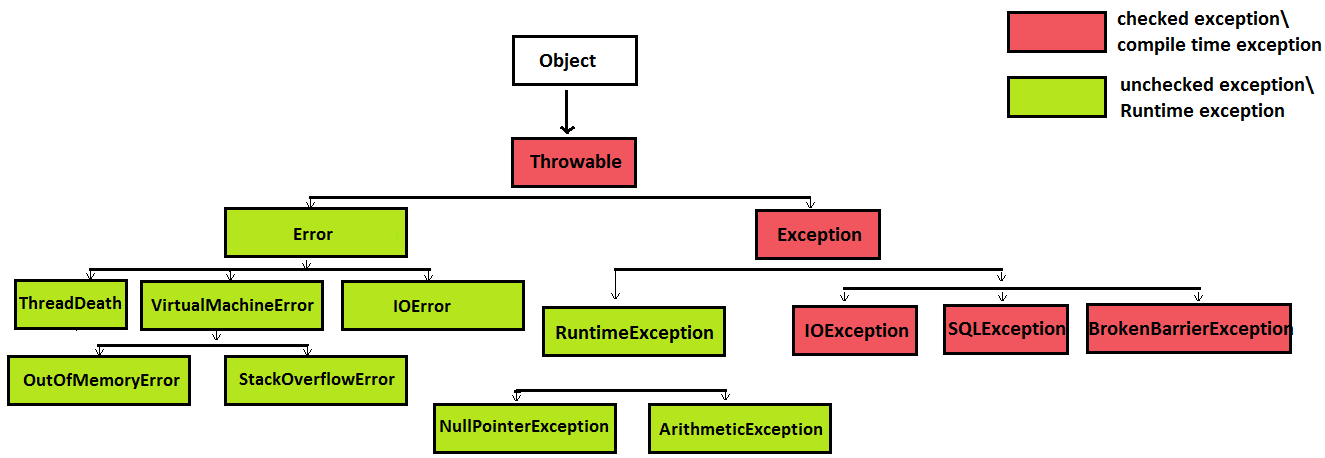
    <artifactId>jackson-core</artifactId>

    <version>2.6.4</version>

</dependency>

* Use objectMapper.writeValueAsString() method to convert pojo to JSON

**17)Exceptions hierarchy**



**1)Checked/compile-time Exceptions**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

**2)unchecked/Runtime Exceptions**

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3**)Error(System Failures)**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

17**) How to create custom checked Exception Class?**

**For checked exception make sure the exception clas should extends Exception super class**

**public** **class** **IncorrectFileNameException** **extends** **Exception** {

**public** **IncorrectFileNameException**(String errorMessage) {

**super**(errorMessage);

} }

17**) How to create custom uncheckedchecked Exception Class?**

**For exception make sure the exception clas should extends RuntimeException super class**

**public** **class** **IncorrectFileExtensionException** **extends** **RuntimeException** { **public** **IncorrectFileExtensionException**(String errorMessage, Throwable err) {

**super**(errorMessage, err)

; } }

**Java Collections**



**19)ArrayDeque**

Arraydeque is the implementation class of deque it does not allow null values . we can add, remove elements on both the ends.

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

Deque<String> deque=**new** ArrayDeque<String>();

deque.offer("arvind"); //add element

deque.offer("vimal");

deque.add("mukul");

deque.offerLast("adad");

deque.offerFirst("jai"); // add elemeny in the first

deque.offerFirst("adad");

System.***out***.println("After offerFirst Traversal...");

**for**(String s:deque){

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

}

deque.poll(); // removed head element only

deque.pollLast(); // removed tail element only

System.***out***.println("After pollLast() Traversal...");

**for**(String s:deque){

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

}

}

## Java Map Hierarchy

Java Map Hierarchy

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

// Adding elements to map

map.put(1, "Amit");

map.put(5, "Rahul");

map.put(2, "Jai");

map.put(6, "Amit");

// Traversing Map

Set<?> set= map.keySet();

Iterator<?> itr = set.iterator();

**while** (itr.hasNext()) {

Object i = itr.next();

System.***out***.println(i.toString());

}

Sort keys ascending order/descending order

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

      map.put(100,"Amit");

      map.put(101,"Vijay");

      map.put(102,"Rahul");

      //Returns a Set view of the mappings contained in this map

      map.entrySet()

      //Returns a sequential Stream with this collection as its source

      .stream()

      //Sorted according to the provided Comparator

      .sorted(Map.Entry.comparingByKey())

      //Performs an action for each element of this stream

      .forEach(System.out::println);

 }

-----------

map.entrySet()

      //Returns a sequential Stream with this collection as its source

      .stream()

      //Sorted according to the provided Comparator

      .sorted(Map.Entry.comparingByKey(Comparator.reverseOrder()))

      //Performs an action for each element of this stream

      .forEach(System.out::println);

## Internal Working of TreeMap

Like HashMap and LikedHasMap it does not use hashing for storing key-value pairs. Internally, it uses a data structure called the **Red-Black Tree**. In other words, it sorts the TreeMap object keys using the Red-Black Tree algorithm. For understanding the internal working of TreeMap, we must understand the Red-Black Tree algorithm.

1. TreeMap<String,String> treemap=**new** TreeMap<String,String>();
2. //we can take anything in the key such as integer, string, etc.
3. //adding elements to the TreeMap
4. treemap.put("H","Ahmedabad ");
5. treemap.put("D","Jaipur");
6. treemap.put("B","Delhi");
7. treemap.put("F","Agra");
8. treemap.put("P","Patna");

**Output:**

B Delhi

D Jaipur

F Agra

H Ahmedabad

P Patna

Employee e1 = **new** Employee("a", 100);

Employee e2 = **new** Employee("d", 200);

Employee e3 = **new** Employee("b", 300);

List<Employee> employeelist = **new** ArrayList<>();

employeelist.add(e1);

employeelist.add(e2);

employeelist.add(e3);

// using comparator to sort

employeelist.stream().

sorted(Comparator.*comparing*(Employee::getSal).

thenComparing(Employee::getName)).forEach(p-> System.***out***.println(p.getName()+"=="+p.getSal()));

//using comparator to get min salary

employeelist.stream().

min(Comparator.*comparing*(Employee::getSal)).

ifPresent(z->System.***out***.println("Maximum string in the set is " + z.getSal()));

//using comparatot for ascendin order

Collections.*sort*(employeelist,(x,y) ->

String.*valueOf*(y.getSal()).compareTo(String.*valueOf*(x.getSal())));

// using summarixen to add salary

employeelist.forEach(x-> System.***out***.println(x.getSal()));

employeelist.stream().collect(Collectors.*summarizingInt*(Employee::getSal));

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |

**Equals method override**

Employee e1 = **new** Employee("a", 100);

Employee e2 = **new** Employee("a", 100);

Employee e3 = **new** Employee("b", 300);

**if** (e1.equals(e2)) {

System.***out***.println("Both are equals e1 hash code" + e1.hashCode() + "e2 hashcocde+" + e2.hashCode());

} **else** {

System.***out***.println("Both are not equals");

------------------- }

@Override

**public** **boolean** equals(Object o) {

// If the object is compared with itself then return true

**if** (o == **this**) {

**return** **true**;

}

**if**(!(o **instanceof** Employee))

{

**return** **false**;

}

// typecast o to Complex so that we can compare data members

Employee c =(Employee)o;

// Compare the data members and return accordingly

**return** name.equals(c.name)

&&(sal==c.sal);

}

**String reverse using Recusrion?**

String reverse(String s)

{

**if**(s.isEmpty())

{

**return** **null**;

}

**else**

{

System.***out***.print(s.charAt(s.length()-1));

**return** reverse(s.substring(0,s.length()-1));

}

}

**Array to find second highest number**

**int** arry[] = {1,2,3,4};

**int** tmp;

**for**(**int** i=0 ;i<arry.length;i++)

{

**for**(**int** j=i+1;j>arry.length;j++)

{

**if**(arry[i]>arry[j])

{

tmp=arry[i];

arry[i]=arry[j];

arry[j]=tmp;

}

}

}

System.***out***.println(arry[arry.length-2]);

}

**Hashmap vs ConcurrentHashmap**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **HashMap** | **ConcurrentHashMap** |
| **Synchronization** | Non-synchronized | synchronized |
| **Thread-safety** | Not thread-safe | Thread-safe |
| **Iterator** | It is fail-fast and throws an exception during iteration | It is fail-safe and performs iteration by multiple threads |
| **Null Values** | It allows for storing null keys and values. | It does not allow to store null key/values. |
| **Performance** | faster | Slower than Hashmap |

**Failfast vs Fail safe Iterator**

|  |  |  |
| --- | --- | --- |
| **ase of Comparison** | **Fail Fast Iterator** | **Fail Safe Iterator** |
| **Exception** | It throws a ConcurrentModificationException in modifying the object during the iteration process. | It does not throw Exception. |
| **Clone Object** | No clone object is created during the iteration process. | A copy or clone object is created during the iteration process. |
| **Memory utilization** | It requires low memory during the process. | It requires more memory during the process. |
| **Modification** | It does not allow modification during iteration. | It allows modification during the iteration process. |
| **Performance** | It is fast. | It is slightly slower than Fail Fast. |
| **Examples** | HashMap, ArrayList, Vector, HashSet, etc | CopyOnWriteArrayList, ConcurrentHashMap, etc. |

**Java 8 Optional class Example**

**Optional class is used to avoid null pointer Exception**

**import** java.util.Optional;

**public** **class** OptionalExample {

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

        String[] str = **new** String[10];

        str[5] = "JAVA OPTIONAL CLASS EXAMPLE";// Setting value for 5th index

        Optional<String> checkNull = Optional.ofNullable(str[5]);

**if**(checkNull.isPresent()){  // It Checks, value is present or not

            String lowercaseString = str[5].toLowerCase();

            System.out.print(lowercaseString);

        }**else**

            System.out.println("String value is not present");

    }

}

**ArrayList to count the number of names**

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

list.add("sujith");

list.add("sujith");

list.add("canada");

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

**for** (String s : list) {

**int** count = 1;

**if** (map.containsKey(s)) {

count = map.get(s);

map.put(s, ++count);

} **else** {

map.put(s, count);

}

}

System.***out***.println("Print count of names in list::" + map.toString());

**HashMap sorting**

We need add hash map key set to tree map

Map<Integer, Employee1> map = **new** HashMap<>();

map.put(3, **new** Employee1("abcd",12));

map.put(1, **new** Employee1("sujith", 30000));

map.put(2, **new** Employee1("canada", 130000));

TreeMap<Integer, Employee1> tree = **new** TreeMap<Integer, Employee1>();

Iterator it = map.entrySet().iterator();

**while** (it.hasNext()) {

Map.Entry map1 = (Map.Entry) it.next();

tree.put((Integer) map1.getKey(), (Employee1) map1.getValue());

}

Iterator it1 = tree.entrySet().iterator();

**while** (it1.hasNext()) {

Map.Entry map2 = (Map.Entry) it1.next();

tree.put((Integer) map2.getKey(), (Employee1) map2.getValue());

System.***out***.println(map2.getKey() + "--" + map2.getValue());

}

**Spring vs SpringBoot**

**spring**

**-------**

**1. it is a framework which is used to develop java EE applications**

**2) it uses dependency injection**

**3) it contains boiler plate code**

**4) it does not support any in memory database**

**5) it doesn not contain any embedded server**

**6) manually add depencencies**

**springboot**

-----------

**1) it is a framework which is used to develop rest apis**

**2) it uses auto confirguration**

**3) it reduces boiler plate code(eg: @getter @setter, @tosring)**

**4) it suports inmemory database(h2)**

**5) it contains tomcat and jetty embedded server**

**6) springboot uses starter internally to download dependencies**

**SpringBoot starter**

**spring boot framer consists of spring starter where it provides us to add jars in the classpath(eg:jpa,starter web)**

**Logging in Spring boot**

**1)by default spring boot used logback**

**2) to get logs we have to add dependency like log4j2**

**3)private static final Logger LOGGER=LoggerFactory.getLogger(Application.class)**

**Synchronization in java**

**synchronization is used to restriccts multiple threads to access a shared resource**

**this we can achieve with synchronized keyword.**

**class** SynchronizationTest {

**void** test(**int** n) {

**for** (**int** i = 1; i < 5; i++) {

System.***out***.println(n \* i);

**try** {

Thread.*sleep*(400);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**class** Thread1 **extends** Thread {

SynchronizationTest b;

**public** Thread1(SynchronizationTest s1) {

b = s1;

}

**public** **void** run() {

b.test(1);

}

}

**class** Thread2 **extends** Thread {

SynchronizationTest a;

**public** Thread2(SynchronizationTest s1) {

a = s1;

}

**public** **void** run() {

a.test(100);

}

}

**public** **class** Synchronization {

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

SynchronizationTest s1 = **new** SynchronizationTest();

Thread1 t1 = **new** Thread1(s1);

Thread t2 = **new** Thread2(s1);

t1.start();

t2.start();

}

}

**Spring Boot exception handling**

**Spring boot provides @RestControlleradvice to handle exceptions globally**

**@Exceptionhandler to handle exceptions specifically**

**Spring framework has   ResponseEntityExceptionHandler**

**we can extend this with @contolleradvice class**

**We don't need to catch any exception at each method or class separately instead you can just throw the exception from the method and then it will be caught under the central exception handler class annotated by @ControllerAdvidWe**

**What is spring bean?**

**Spring will create objects, manipulate that are defined with @Bean in configuration class in IOC container**

**@Bean is a annotation which is used to create a object in spring IOC container.**

**also we can create object in ioc with @component on top of the class.**

b

**spring bean IOC container**

**IOC CONTAINER**

**Spring Boot annotations**

**@Component**

**This is a generic stereotype spring annotation**

**@Controller**

**This is a specialized component which is used in Controller Class. Basically Dispathcer srver with the help request checks for this annotation**

**@Repository**

**This is used in DAO/jpa LAYER where it will handle/throw DB exceptions**

**@Service: It is also used at class level. It tells the Spring that class contains the business logic.**

**@Configuration:**

**It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions**

**@ComponentScan:**

 It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**Spring Data Rest**

**Spring data rest build on spring data project . it provides hypermedia driver rest web services.**

**It requires less code**

**To implememtnt we just need a Entity and A Repo class**

**With DB**

@RepositoryRestResource(collectionResourceRel = "users", path = "users") **public** **interface** **UserRepository** **extends** **JpaRepository**<**WebsiteUser**, **Long**> { List<WebsiteUser> **findByName**(@Param("name") String name); }

**How to sort List of Integers using Streams(descending order)**

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

list.add(1);

list.add(2);

list.add(3);

System.***out***.println(list.stream().sorted(Comparator.*reverseOrder*()).collect(Collectors.*toList*()));

**Sql query to fetch third highest salary**

SELECT sal

FROM (

SELECT DISTINCE sal

FROM emp

ORDER BY sal DESC )

WHERE rownum=3;

I am currently working in Mphasis as a Java Developer. Basically I have 3 years of relevant work experience in Core java, J2ee, Micro services, Restful Web Services, Spring Boot, My DataBasa, Spring Cloud, Git&SVN ,Junit-

**How to convert Employee List to HashMap**

List<Employee> employeelist = **new** ArrayList<>();

employeelist.add(e1);

employeelist.add(e2);

employeelist.add(e3);

HashMap<String,Integer> map = **new** HashMap<>();

employeelist.stream().forEach(x->

{

map.put(x.getName(),x.getSal());

});

**Functional Interface Examples in Java 8**

**1)Predicate**

**This will return Boolean type**

**2)Supplier**

**Eg:FindAny().orElse(“”)**

**This will return some informtion**

**3)Consumer**

**Ex:foreach()**

**This will output result**

**4)Bi Function-> it takes two arguments and returns result**

**Spring actuators Disable some Indicatos**

**To Disable specify name of the indicator and make it as enabled false in Property file**

**management.health.random.enabled=false**

**Default and Static Methods in Functional Interface**

**They will not be implemented in the implementation class**

@FunctionalInterface

**interface** TestFunctional {

**void** say();

**default** **void** tell() {

System.***out***.println("Default method");

}

**static** **void** tellstatic() {

System.***out***.println("this is static");

}

}

**class** Impl **implements** TestFunctional {

@Override

**public** **void** say() {

System.***out***.println("This is implemenmted clsass");

}

}

**public** **class** FunctionalInteface {

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

TestFunctional t = **new** Impl();

t.say();

t.tell();

TestFunctional.*tellstatic*();

}

}

**Thread Methods**

**Wait()-> it will wait until we use notify()or notifyAll()**

**Sleep()-> iT will wait for sometime as per The time**

**Bean Scopes**

* 1. **Singleton Scope**
  2. **Prototype Scope**
  3. **Request scope -> crate bean instance for every request**
  4. **Session scope \_>create single instance for multiple http request**

**Program to find the least positive integers**

**int** [] a = {-8,6,1,4,7};

//Make array sorted

**for**(**int** i=0;i<a.length;i++)

{

**if**(a[i]>0 && i!=a[i]) {

System.***out***.println("least positive numbr i s::"+i);

**break**;

}

}

}

}

**Stream to fetch the last element/ stream of elements**

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

// **TODO** Auto-generated method stub

ArrayList<String> list = **new** ArrayList<>();

list.add("1");

list.add("2");

list.add("3");

list.add("4");

list.add("5");

list.add("6");

list.stream().skip(5).forEach(x -> System.***out***.println(x));

}}