Interview Questions

JAVA

1.Output of below code:

String str1 = new String("java");

        String str2 = new String("java");

        if(str1 == str2)

        System.out.println("if block");

        }else{

        System.out.println("else block");

        }

2. Sort the below list based on Age of Emp class

List <Employee> list=new Arraylist<Employee>();

List.add(new Employee(3,”c”));

List.add(new Employee(2,”b”));

List.add(new Employee(1,”c”));

Comparator<Employee> cmp = Comparator.comparing(Employee::getEmpid);

Collection.sort(List,cmp);

Sort with multiple fields

    Comparator<Employee> compareByName = Comparator

                        .comparing(Employee::getFirstName)

                        .thenComparing(Employee::getLastName);

   List<Employee> sortedEmployees = employees.stream()

                    .sorted(compareByName)

                    .collect(Collectors.toList());

3. write a singleton class with thread safe

public class ASingleton **implements** Serializable, Cloneable {

private static volatile ASingleton instance;

private static Object mutex = new Object();

private ASingleton() {

}

public static ASingleton getInstance()

{

ASingleton result = instance;

if (result == null)

{

synchronized (mutex)

{

result = instance;

if (result == null)

instance = result = new ASingleton();

}

}

return result;

}

**protected** Object readResolve()

{

**return** instance;

}

 @Override

**protected** Object clone() **throws** CloneNotSupportedException

 {

**return** **super**.clone();

 }

}

public class CassandraAstyanaxConnection {

private CassandraAstyanaxConnection(){ }

private static class Holder {

private static final CassandraAstyanaxConnection INSTANCE = new CassandraAstyanaxConnection();

}

public static CassandraAstyanaxConnection getInstance() {

return Holder.INSTANCE;

}

// rest of class omitted

}

4. Find duplicates from List or Array

input -----> {1, 2, 3, 4, 2, 7, 8, 8, 3}.

Set s = new Hashset()

for (i=0; i< input.length();i++)

{

for (j=i+1; j<input.length;j++)

{

if (input[i] = input [j])

{

s.add(Integer.valueof(input[i]));

}

}

}

for(Integer n : s)

system.out.println(n);

ARRAY

for (String name : names) {

if (set.add(name) == false) {

// your duplicate element

}

Find duplicates in a LIST:

List list = {1, 2, 3, 4, 2, 7, 8, 8, 3}

For(int I : list)

{

If(list.contains(l))

Sout(l);

}

Set<T> items = new HashSet<>();  
return list.stream()  
 .filter(n -> !items.add(n)) // Set.add() returns false if the element was already in the set.  
 .collect(Collectors.*toSet*());

4. Find even numbers

int arr[]= {1,2,3,4,5,6};

Arrays.asList(arr).stream.filter(a-> a%2==0).foreach(a->System.out.println(a));

5.disadv of microservices

# **Advantages of Microservices**

* Microservices are self-contained, independent deployment module.
* The cost of scaling is comparatively less than the monolithic architecture.
* Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.
* It is possible to change or upgrade each service individually rather than upgrading in the entire application.
* Microservices allows us to develop an application which is organic (an application which latterly upgrades by adding more functions or modules) in nature.
* It enables event streaming technology to enable easy integration in comparison to heavyweight interposes communication.
* Microservices follows the single responsibility principle.
* The demanding service can be deployed on multiple servers to enhance performance.
* Less dependency and easy to test.
* Dynamic scaling.
* Faster release cycle.

## **Disadvantages of Microservices**

* Microservices has all the associated complexities of the distributed system.
* There is a higher chance of failure during communication between different services.
* Difficult to manage a large number of services.
* The developer needs to solve the problem, such as network latency and load balancing.
* Complex testing over a distributed environment.

custom thread pool

circular dependency -

@postconstruct, @Lazy, setter injection

prototype into singleton -

ObjectFactory<Object> , @Lookup,Provider<Object>

transactions in spring

Hibernate openSession() vs getCurrentSession()

|  |  |  |
| --- | --- | --- |
| **Parameter** | **openSession** | **getCurrentSession** |
| Session object | It always create new Session object | It creates a new Session if not exists , else use same session which is in current hibernate context |
| Flush and close | You need to explicitly flush and close session objects | You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally |
| Performance | In single threaded environment , It is slower than getCurrentSession | In single threaded environment , It is faster than getOpenSession |
| Configuration | You do not need to configure any property to call this method | You need to configure additional property “hibernate.current\_session\_context\_class” to call getCurrentSession method, otherwise it will throw exceptions. |

blocking and non blocking get in future

The Java Future object is used to get the result of asynchronous computation which is performed by a parallel thread(Executors). We call Future.get() method and wait until result is ready.

Non blocking get is supported with [CompletableFutures](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html).

put and patch

**Difference between PUT, POST, GET, DELETE and PATCH IN HTTP Verbs:**

The most commonly used HTTP verbs POST, GET, PUT, DELETE are similar to CRUD (Create, Read, Update and Delete) operations in database. We specify these HTTP verbs in the **capital** case. So, the below is the comparison between them.

1. create - POST
2. read - GET
3. update - PUT
4. delete - DELETE

**PATCH:** Submits a partial modification to a resource. If you only need to update one field for the resource, you may want to use the PATCH method.

**Note:**  
**Since POST, PUT, DELETE modifies the content, the tests with Fiddler for the below url just mimicks the updations. It doesn't delete or modify actually. We can just see the status codes to check whether insertions, updations, deletions occur.**

**URL:** <http://jsonplaceholder.typicode.com/posts/>

1) **GET:**

GET is the simplest type of HTTP request method; the one that browsers use each time you click a link or type a URL into the address bar. It instructs the server to transmit the data identified by the URL to the client. Data should never be modified on the server side as a result of a GET request. In this sense, a GET request is read-only.

**Checking with Fiddler or PostMan:** We can use Fiddler for checking the response. Open Fiddler and select the Compose tab. Specify the verb and url as shown below and click Execute to check the response.

**Verb:** GET

**url:** <http://jsonplaceholder.typicode.com/posts/>

**Response:** You will get the response as:

"userId": 1, "id": 1, "title": "sunt aut...", "body": "quia et suscipit..."

In the “happy” (or non-error) path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

**2) POST:**

The POST verb is mostly utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource.

On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

**Checking with Fiddler or PostMan:** We can use Fiddler for checking the response. Open Fiddler and select the Compose tab. Specify the verb and url as shown below and click Execute to check the response.

**Verb:** POST

**url:** <http://jsonplaceholder.typicode.com/posts/>

**Request Body:**

data: { title: 'foo', body: 'bar', userId: 1000, Id : 1000 }

**Response:** You would receive the response code as 201.

If we want to check the inserted record with Id = 1000 change the verb to Get and use the same url and click Execute.

As said earlier, the above url only allows reads (GET), we cannot read the updated data in real.

**3) PUT:**

PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.

**Checking with Fiddler or PostMan:** We can use Fiddler for checking the response. Open Fiddler and select the Compose tab. Specify the verb and url as shown below and click Execute to check the response.

**Verb:** PUT

**url:** <http://jsonplaceholder.typicode.com/posts/1>

**Request Body:**

data: { title: 'foo', body: 'bar', userId: 1, Id : 1 }

**Response:** On successful update it returns 200 (or 204 if not returning any content in the body) from a PUT.

**4) DELETE:**

DELETE is pretty easy to understand. It is used to **delete** a resource identified by a URI.

On successful deletion, return HTTP status 200 (OK) along with a response body, perhaps the representation of the deleted item (often demands too much bandwidth), or a wrapped response (see Return Values below). Either that or return HTTP status 204 (NO CONTENT) with no response body. In other words, a 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses.

**Checking with Fiddler or PostMan:** We can use Fiddler for checking the response. Open Fiddler and select the Compose tab. Specify the verb and url as shown below and click Execute to check the response.

**Verb:** DELETE

**url:** <http://jsonplaceholder.typicode.com/posts/1>

**Response:** On successful deletion it returns HTTP status 200 (OK) along with a response body.

**Example between PUT and PATCH**

**PUT**

If I had to change my firstname then send PUT request for Update:

{ "first": "Nazmul", "last": "hasan" } So, here in order to update the first name we need to send all the parameters of the data again.

**PATCH:**

Patch request says that we would only send the data that we need to modify without modifying or effecting other parts of the data. Ex: if we need to update only the first name, we pass only the first name.

Please refer the below links for more information:

Circuit breaker

The circuit breaker is fault-tolerance technique that monitors and detects when a service is behaving abnormally. It temporarily rejects those calls until the service becomes healthy again.

Netflix Hystrix is an open source library which provides this solution

<https://javatechonline.com/how-to-implement-hystrix-circuit-breaker-in-microservices-application/>

Hashmap working

1) If two objects are equal, then they must have the same hash code.  
2) If two objects have the same hash code, they may or may not be equal.

Pessimistic vs optimistic locking in hibernate

In pessimistic locking, the object is locked when it is initially accessed for the first time in a given transaction. The lock then is released only when the transaction completes; the object is not accessible for any other transactions during the transaction.

In optimistic locking, the object is not locked when it is accessed for the first time in the transaction. Instead, its state (generally the version number) is saved. When other transactions that are accessing the same object try to modify the state of the object, the present state and the saved state are compared. If the states differ, then it’s a clear indication of a conflicting update and the transaction will be rolled back.

The difference between @Inject vs. @Autowire vs. @Resource?

**@Autowired:** spring propriety annotation (as opposed to @Inject and @Resource) that inject a resource by-type, i.e. by the class of by the interface of the annotated field or contractor. In case we have few implementation of an interface or a subclass we can narrow down the selection using the @Qualifier annotation to avoid ambiguity. For a fallback match, the bean name is considered a default qualifier value. Although you can use this convention to refer to specific beans by name, @Autowired is fundamentally about type-driven injection with optional semantic qualifiers.

**@Inject:** Annotation based on JSR-330 (Dependency Injection for Java) identifies injectable constructors, methods, and fields. This annotation is an almost complete drop-in replacement for Spring’s @Autowired annotation. So, instead of using the Spring-specific @Autowired annotation, you might choose to use @Inject. One of the differences between @Autowired and @Inject is that @Inject does not have the required field so in case we fail to find a suitable object to injected it will fail while @Autowired can used required=false and allow null able field (only if required!). Advantage of @Inject annotation is that rather than inject a reference directly, you could ask @Inject to inject a Provider. The Provider interface enables, among other things, lazy injection of bean references and injection of multiple instances of a bean. In case we have few implementation of an interface or a subclass we can narrow down the selection using the @Named annotation to avoid ambiguity. @Named annotation works much like Spring’s @Qualifier

**@Resource:** annotation based on JSR-250. @Resource is quite similar to @Autowired and @Inject, but the main difference is the execution paths taken to find out the required bean to inject. @Resource will narrow down the search first by name then by type and finally by Qualifiers (ignored if match is found by name). @Autowired and @Inject will narrow down the search first by type then by qualifier and finally by the name.

Many to one relationship

1. @ManyToOne(cascade=CascadeType.ALL)

**private** Address address;

1. @OneToOne(cascade=CascadeType.ALL)
2. **private** Employee employee;

How to load beans conditionally

@ConditionalOnProperty(prefix = "notification", name = "service")

@Conditional(IsDevEnvCondition.class)

@ConditionalOnBean(CustomLoggingConfiguration.class)

@ConditionalOnClass(CustomLogger.class)

@ConditionalOnJava(JavaVersion.EIGHT)

How to handle exception globally and separately in spring boot

# **Spring Boot Global Exception Handling**

**CustomizedExceptionHandling.java class:** This class is annotated with @ControllerAdvice which means that this class will be able to handle the exceptions occured from any of the **Controllers**(not only limited to the **ResourceController.java**). It extends the class ResponseEntityExceptionHandler which exposes the functionality required to implement this feature. The @ExceptionHandler annotation is required to handle the type of exceptions which are thrown and the corresponding method will be able to handle the exceptions.

@ControllerAdvice

public class CustomizedExceptionHandling extends ResponseEntityExceptionHandler {

@ExceptionHandler(TeacherNotFoundException.class)

public ResponseEntity<Object> handleExceptions( TeacherNotFoundException exception, WebRequest webRequest) {

ExceptionResponse response = new ExceptionResponse();

response.setDateTime(LocalDateTime.now());

response.setMessage("Not found");

ResponseEntity<Object> entity = new ResponseEntity<>(response,HttpStatus.NOT\_FOUND);

return entity;

}

}

Separately handling Exception:

@ResponseStatus(value = HttpStatus.NOT\_FOUND) **public** **class** **MyResourceNotFoundException** **extends** **RuntimeException** { **public** **MyResourceNotFoundException**() { super(); }

How to load Properties

BST tree

node, left, right

if(node ==null)

return false;

if(left!=null&& node.data<=left.data)

return false;

if ()

if(isBst(node.left, left , node )||isBst(node.right, node, right))

5. Sum of even elements from list

list.streams.filter(l->l%2!=0).map(m-> m\*m).collect(Collectors.summingDouble());

6.comparable and comparator

7. Cows and bulls

class Solution {

public String getHint(String secret, String guess) {

int[] secretCounts = new int[10];

int[] guessCounts = new int[10];

int numA = 0;

int numB = 0;

for(int i = 0; i < secret.length(); i++){

char s = secret.charAt(i);

char g = guess.charAt(i);

if(s == g){

numA++;

}else{

if(guessCounts[s - '0'] > 0){

numB++;

guessCounts[s - '0']--;

}else{

secretCounts[s - '0']++;

}

if(secretCounts[g - '0'] > 0){

numB++;

secretCounts[g - '0']--;

}else{

guessCounts[g - '0']++;

}

}

}

return numA+"A"+numB+"B";

}

}

find the index of an array that gives sum of target.

Array {1, -3, 6, 2, 9} , target = 0

answer. = 0,1,3

default = -1,-1,-1

Custom array list

class ArrayList{

int list[] = null;

int capacity=0;

int size=0;

public ArrayList(int capacity)

{

list[] = new int[capacity];

this.capacity=capacity;

}

void add(int element)

{

if(size==capacity)

{

print(list is full)

}

else

{

int i=0;

boolean flag = false;

while(i<size) // 1,2,3

{

if(list[i]==element)

{

flag=true;

break;

}

i++;

}

if(!flag)

{

list[size]=element;

size++;

}

}

Find the occurrence of each element in a list

List<String> items = Arrays.asList("apple", "apple", "banana", "apple", "orange", "banana", "papaya");

Output:

apple -3

banana-2

orange-1

papaya-1

list.stream().filter(i -> Collections.*frequency*(list, i) > 1)  
 .collect(Collectors.*toSet*());

Print unique numbers in the list

List<Integer> list = Arrays.asList(5,3,2,1,2,3,4,1,2,3,4,1,2,3,1,9,12);

Output = 5,3,2,1,4,9,12

       List<Integer> UniqueNumbers

            = Numbers.stream().distinct().collect(

                Collectors.toList());

Print sorted binary in an array

arr = 1,0,3,0,0,2

int prev;

for(int i=0; i<arr.length()-1;i++)

{

if(a[i] == 0)

countinue;

else

if(a[i+1]==0)

prev = a[i];

a[i]=a[i+1];

a[i+1]=prev;

Optimal platform required to depart the train

arr[] = {7:00,8:10,9:00, 9:40, 9:50, 11:00, 15:00, 18:00}

dep[] = {8:20,8:40,9:10, 12:00, 11:20, 11:30, 19:00, 20:00}

**static** **int** findPlatform(**int** arr[], **int** dep[], **int** n)

    {

        // Sort arrival and departure arrays

        Arrays.sort(arr);

        Arrays.sort(dep);

        // plat\_needed indicates number of platforms

        // needed at a time

**int** plat\_needed = 1, result = 1;

**int** i = 1, j = 0;

        // Similar to merge in merge sort to process

        // all events in sorted order

**while** (i < n && j < n) {

            // If next event in sorted order is arrival,

            // increment count of platforms needed

**if** (arr[i] <= dep[j]) {

                plat\_needed++;

                i++;

            }

            // Else decrement count of platforms needed

**else** **if** (arr[i] > dep[j]) {

                plat\_needed--;

                j++;

            }

            // Update result if needed

**if** (plat\_needed > result)

                result = plat\_needed;

        }

**return** result;

    }

Find the missing number in an array . (data range should be 0 to m)

data={0,1,2,4,5,11}

n=6 size of array

m=14 arbitrary number

n<m;

data[i]<= m-1

m=0;

for(int i=0;i<data.length-1;i++)

{

if(data[i]!==m)

{

print (m)

break;

}

m++; }

Request should go 3:1 . Logic need to use for R1

APAC (75%)

Stream of data packets -> R1->

EMEA (25%)

Generate random number between 0 to 2. 0 and 1 will go APAC and 2 will go EMAE

Print the salary of employee whose age is >25

list<employee> find the salary for an employee age >25

listsal=list.streams.filter(l-> l.age>25).map(l->l.salary).collect(Collector.toList());

listsal.foreach(system.out::prinln);

Singleton Class

class SingletonExample implements Cloneable,Serialisable

{

private static SingletonExample obj = null;

private SingletonExample()

{

}

public static SingletonExample getObj()

{

if(obj ==null)

{

obj = new SingletonExample();

}

return obj;

}

public SingletonExample clone() throws ClonenotsupportException{

throw new ClonenotsupportException

readresolve ()

{

}

}

Immutable class in java

Why mark all fields final as well?

Mark all the fields final so even we will not be able to change the fields outside of the constructor.

public final class FinalClassExample {

private final int id;

private final String name;

private final HashMap<String,String> testMap;

public int getId() { return id; }

public String getName() { return name; }

/\*\* \* Accessor function for mutable objects \*/

public HashMap<String, String> getTestMap() { //return testMap;

return (HashMap<String, String>) testMap.clone();

}

/\*\* \* Constructor performing Deep Copy \* @param i \* @param n \* @param hm \*/

public FinalClassExample(int i, String n, HashMap<String,String> hm)

{

System.out.println("Performing Deep Copy for Object initialization"); this.id=i;

this.name=n;

HashMap<String,String> tempMap=new HashMap<String,String>();

String key;

Iterator<String> it = hm.keySet().iterator();

while(it.hasNext()){ key=it.next();

tempMap.put(key, hm.get(key));

}

this.testMap=tempMap;

}

Custom Exception

public class CustomException extends Exception

{

public CustomException(String s)

{

super();

sout(s)

}

Output for Below snippet:

public int getNumber() {

try {

int n=3/0;

System.out.println("try");

return 1;

} catch(Exception ex) {

System.out.println("catch");

return 0;

} finally {

System.out.println("finally");

return 2;

}

System.out.println("test");

}

catch

public int getNumber() {

try {

int n=3/0;

System.out.println("try");

return 1;

} catch(Exception ex) {

System.out.println("catch");

} finally {

System.out.println("finally");

return 2;

}

}

catch

finally - 2

public int getNumber() {

try {

int n=3/0;

System.out.println("try");

return 1;

} catch(Exception ex) {

System.out.println("catch");

return 0;

} finally {

System.out.println("finally");

return 2;

}

}

catch -

finally 2

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

A subarray is a contiguous part of an array.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Example 2:

Input: nums = [1]

Output: 1

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

public static void maxSubarrays(int arr[])  
{  
int max = Integer.MIN\_VALUE;  
  
for(int i = 0 ; i<arr.length;i++)  
{  
int temp=arr[i];  
  
for(int j=i+1;j<arr.length;j++)  
{  
if(temp>max)  
{  
max= temp;  
}  
  
temp=temp+arr[j];  
  
}  
if(temp>max)  
{  
max= temp;  
}  
  
}  
  
System.out.println(max);  
  
  
}

swap without temp

a= 10;

b= 20;

a = a+b; // 30

b = a-b; // 10

a = a-b; // 20

Exception in Inheritance

class Athena {

void m1() throws IOException {

System.out.println("In Athena");

}

}

class Epocrates extends Athena {

void m1() throws Exception {

System.out.println("In Epocrates");

}

}

public class Test {

public static void main(String[] args) {

Athena obj = new Epocrates();

try {

obj.m1();

} catch (IOException e) {

e.printStackTrace();

}

}

}

List of list (size k) we have following elements and find the combination of those elements

a b c

d e f

g h i

list of list

adg

Check two strings have all the characters

s1 - "mary"

s2 - "army"

char[] c1 = s1.toCharArray();

char[] c2 = s2.toCharArray();

s1.length() != s2 .length()

return false;

int i = 0;

while(i<s1.length()-1)

{

if(Integer.valueof(c1[i])> Integer.valueof(c1[i+1]))

{

char temp = c1[i];

c1[i] = c1[i+1];

c1[i+1]=temp;

}

if(Integer.valueof(c2[i])> Integer.valueof(c2[i+1]))

{

char temp = c2[i];

c2[i] = c2[i+1];

c2[i+1]=temp;

}

}

If(new String(c1).equals(c2))

{

Print equal

}

1. Array reverse, Expression tree and Binary tree. Simple In- Memory cache.

2. Merge 2 sorted arrays into a single one. Find duplicates no. in array of natural no.

3. Function to return true if array contains three elements whose sum is X

4. When do you use HashMap vs TreeMap vs LinkedHashMap?

5. ArrayList vs LinkedList. How HashMap works internally.

6. Reverse a stack using just push and pop no extra space.

7. Design LLD for Instagram/ photo sharing app, Design A Cron Job scheduler.

8. Design HLD for Twitter system/ automated parking lot to minimize the entry and exit times.

next palindrome number

int n = 9;

int reminder = 0;

int que = 0;

String sum = "";

for

while (n>0)

{

reminder = n % 10;

sum = sum + String.valueOf(que); // sum\*10+reminder;

que = n/10;

}

if(n == Integer.parseInt(sum))

{

print palindrome

}

Btree:

Class Btree{

Node root;

public void insert(int data)

{

Node node = new Node(data)

if(root == null)

{

root = node;

}

else if (root.data>node.data)

{

root.right = node;

}

else

{

root.left = node;

}

}

Override ‘equals()’ to avoid same key irrespective of case

Map<k,v> map = new HashMap<>();

map.put("ABC","xyz");

map.put("abc","Test");

Use a [TreeMap](http://java.sun.com/javase/6/docs/api/java/util/TreeMap.html) which is constructed with [String#CASE\_INSENSITIVE\_ORDER](http://java.sun.com/javase/6/docs/api/java/lang/String.html" \l "CASE_INSENSITIVE_ORDER).

Map<String, String> map = new TreeMap<String, String>(String.CASE\_INSENSITIVE\_ORDER);

map.put("FOO", "FOO");

System.out.println(map.get("foo")); // FOO

System.out.println(map.get("Foo")); // FOO

System.out.println(map.get("FOO")); // FOO

Print output of below format.

int a[] = {3,1,2,3}

o/p = 6,18,9,6

int l[] = new int[a.length];

int r[] = new int[a.length];

int p[] = new int[a.length];

l[0] = 1;

r[a.length-1] =1

for(int i=1;i<a.length;i++)

{

l[i] = a[i-1]\*l[i-1]; // a[0] \* l[0] 2\*3 1,3,3,6

}

for(int j=a.length-1;j>=0;j--)

{

r[i] = a[i+1]\*r[i+1]; //6,6,3,1

}

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

{

p[i] = l[i] \* r[i];

}

What is the o/p of below code:

Double d = 1/2;

sout (d);

0.0

What is the o/p of below code

double d1 = DOuble.NAN;

double d2 = Double.NAN;

sout(d1 == d2) //false

(String.valueof(d1)).equals(String.valueof(d2)) // true

Well, log(-1) gives NaN, and acos(2) also gives NaN. Does that mean that log(-1) == acos(2)? Clearly not. Hence it makes perfect sense that NaN is not equal to itself.

Remove element from a list not using index but value;

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

list.add(1);

list.add(2);

list.add(3);

list.remove(2);

Integer i = new Integer(2);

list.remove(Integer.valueOf(2));

1 2

Print Output of below code

class A{

static void print (){

print(“hello”);

}

}

A a = null;

sout(a.print);

O/P -> hello

Print Output of below code:

class B{

static void b(int i){

sout(static)

}

void display()

{

sout(normal)

}

}

class A extends B{

}

A a = new A();

b.b(5)

b.display();

static

child plain

Sort a list twice based on name and age:

List<Student> list =

Comparator<Student> comp = Comparator.comparing(Student::getage);

Collections.sort(list,comp);

list.stream().sorting(1->l.name).thencomparing(l)

WALMART

Online round: [60 mins]

1. given a number N. print in how many ways it can be represented as N = a+b+c+d , 1< =a< =b< =c< = d; 1<=N< = 5000

2. given two number l and r (l<=r<=10^6) find most frequent digit among all prime numbers between l and r inclusive. if frequency is same print highest digit.

Round 1:

1. difference between wait() and sleep() in java.

2. write code to print 1 to 20 using 2 threads. one thread will print only even and another one only odd.

3. Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words. print all possible sequence(Word Break Problem). how will you implement dictionary.

4. discussion on REST services.

5. discussion on garbage collector algorithms and different heap generations.

6. some example on sql and he asked me to write sql code to get desired results. discussion on SQL CASE expressions.

Round 2:

1. discussion on different type of object relations (aggregation, association etc).

2. difference between NoClassDefFoundError and ClassNotFoundException in java.

|  |  |
| --- | --- |
| **ClassNotFoundException** | **NoClassDefFoundError** |
| It is an exception. It is of type java.lang.Exception. | It is an error. It is of type java.lang.Error. |
| It occurs when an application tries to load a class at run time which is not updated in the classpath. | It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at run time. |
| It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass(). | It is thrown by the Java Runtime System. |
| It occurs when classpath is not updated with required JAR files. | It occurs when required class definition is missing at runtime. |

3. difference between wait(seconds) and sleep() and their uses.

3. discussion on mutex and semaphore. discussion on lock and synchronization.

4. given linkedlist and a number k. reverse linkedlist in a group of k.

5. there are n houses in a city. all in a single row. we can color a house with any one of three colors (R,G,B). cost of coloring each house is given for each color. no two neighbour house can have same color. print min cost to color all house.

6. Given a wiki page, implement find all and replace all with limited regex support (? and \*). e.g. of search : ‘?\*?th’

Round 3(Manager):

1. why walmart, discussion on my current project.

2. discussion on sql indexing and joins.

3. discussion on java class initialization and constructors.

4. discussion on garbage collector algorithms.

5. 2 mathematical puzzle. first one was something related to counting numbers using fingers of one hand in alternate order. given a large number which finger will be the last finger when counting ends. second one was related to time velocity concept.

6. about my weakness and how will i improve it.

7. what was the last feedback you got from your manager.

Round 4 (HR):

1. tell me abt yourself, why walmart?

2. where u see yourself after 5 yesrs.

3. what i like and dislike about my current manager.

4. my expectation from walmart.

Round 1:

1. project role, and responsibility.
2. How to detects loops in graphs.
3. How to find the shortest path.
4. Application of graphs and whether I have implemented them in my previous experiences
5. data structures and algorithms, arrays, trees, sorting concepts.
6. move all negative integers in the array to the left and all the positive integers to the right and in between 0s.
7. I a 2-dimensional matrix find the minimum path starting from the first element to the last element.
8. Count the number of nodes at a particular level in a tree.
9. Copy from Geeksforgeeks, Print all distinct characters of a string in order.
10. Given a string, find the all distinct (or non-repeating characters) in it. For example,

“Geeks for Geeks”-> ‘for’;

“Geeks Quiz” -> ‘GksQuiz’;

1. Copy from Leetcode, 322 Coin Change.
2. You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount.
3. Input: coins = [1, 2, 5], amount = 11;

Output: 3;

Explanation: 11 = 5 + 5 + 1;

1. Given a vector of strings. He gave another string and asked to print all the anagrams in the order in which they appeared in the vector.
2. You have 2 not sorted integer Lists. Write a code to merge them with unique integers in sorted order.

For example 1, 4, 3 and 2, 4, 5 result will be 1, 2, 3, 4, 5

1. Write code for following programs, optimize and deduce the time complexities:

Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits.

We are given two arrays that represent the arrival and departure times of trains that stop.

Input:

arr[] = {9:00, 9:40, 9:50, 11:00, 15:00, 18:00}

dep[] = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}

Output:

3

Explanation: There are at-most three trains

at a time (time between 11:00 to 11:20)

Input:

arr[] = {9:00, 9:40}

dep[] = {9:10, 12:00}

Output: 1

Explanation: Only one platform is needed.

1. Given heights of buildings, find the water clogged above them. when it rains. Width of all are same and placed next to each other with no space in between them.

Example: heights are : 4,2.6 Water clogged will be 2 units above 2nd building.

1. Given n clusters of k machines, need to transfer one file to all of them. Find best way to do it.

Working of torrent as well.

1. Produce a mirror tree for an n-ary tree.
2. Given a number S. Given an array of n numbers. Find whether there are 2 elements in the array such that there sum is equal to S. if present return the indices. Expected time complexity O(nlogn)

Round 2:

1. basic Core Java Questions like Overloading, Collections, Design Pattern, Custom Objects, Types of Map, etc
2. Overloading
3. Comparator and Comparable
4. Singleton Design Pattern Write a program.
5. Immutable class other than string.
6. Design Patterns used.
7. Find a pivot index in an array
8. Java. What is the difference between Hashtable, HashMap and ConcurrentHashMap?
9. what are jax-WS, Jax-RS services
10. what is sonar code anaylaser
11. multhreading scenario in web services and application server tomcat
12. Implement multi-threading
13. linked list reverse, find mid of linked list, string reverse recursively
14. Hashmaps and internal working, hashcode
15. data structures, trees, complexities of algorithms and asked to reduce the complexities
16. Spring MVC, Dependancy Injection. Was asked to explain it with the help of a use case scenario
17. Asked about hibernate, persistence layer
18. What's singleton? How to design a singleton class with multi-threaded access?
19. What's LRU? Design/implement.
20. Design a complete employee management system using 2 microservices for updating reading and creating employee information.
    1. Define controllers/APIs for the same.
    2. What DB should be used? Why? Data types to be used in DB? Reasons? Define DB Schema. Write SQL queries for all the operations possible.
21. Internal working of HashMap. : (explained about the working and how equals() and hashCode() method play their role)
22. Followup Question: Implement hashCode for each employee entry in the above system so that there is no collision
23. Explain about ExecutorService in java. Explain thread pools and working.
24. How should we determine the number of threads to be used for different kinds of operations.
25. Explanation about CPU-intensive and io-intensive operations.
26. Explain all the new features of java 8.
27. (Functional interface – lambda, java streams, date-time APIs, default method in interfaces, etc)
28. Questions on java HashMap. How the internal working has changed in the latest version of java.
29. Project-related questions. Working on the projects. Tech stack used.
30. Questions related to java stack overflow and recursive constructors.
31. Behavioural questions.
32. then they asked me about some basic concept about Java, like interface, abstract class, super(), this, OOP. After one week I got the second phone screen, asked me two coding questions, one is string, the other is linked list, one week later
33. Implement cache with proper synchronization.
34. Leetcode Meeting Room II question
35. First asked about projects and internships. Then asked about JavaScript knowledge, design two classes, one inherits another; what is closure.
36. a simple print question. eg. given [1,2,3] print [1-1] [1-2] [1-3] [2-2] [2-3] [3-3]
37. This round was on the binary trees. How to check trees is balanced or not,
38. How you will balance a tree, and How you will implement a tree from scratch.
39. I was also asked about the AWL tree but only theoretical knowledge sufficed.

Round 3:

1. I mostly heard from others that this round mostly focuses on Java Multi-threading.
2. I was prepared for it but this round was on Garbage Collection both serial and parallel.
3. Design an elevator. OOP. Very famous and common question and you can find it and its answers everywhere on the internet. Design a parking lot.
4. Design a vending machine.
5. Design a restaurant reservation service. OOP, not system design. You can find this question on careercup and also in Grokking the Object Oriented Design Interview.
6. String/array manipulation, system design, backtracking algorithm, LRU cache, memory management

SQL

First Highest Salary

SELECT MAX(SALARY) FROM EMPLOYEE

Second Highest salary

SELECT MAX(SALARY) FROM EMPLOYEE

WHERE SALARY<(SELECT MAX(SALARY) FROM EMPLOYEE)

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

SELECT EmployeeId, Salary

FROM

(

Select EmployeeId, Salary, DENSE\_RANK() OVER(Order by Salary Desc) as Salary\_Order

from Employee

) DT

WHERE DT. Salary\_Order = N ;

Find the minimum number of groups who's sum of each group is at max 3, and every element must be in a group.  
Given an Array like: [1.01, 1.01, 3.0, 2.7, 1.99, 2.3, 1.7]  
return the minimum number of groups, in this case it would be 5 groups: (1.01 , 1.99), (1.01, 1.7), (3.0), (2.7), (2.3)  
Constraint: all elements are between 1.01-3 inclucsive, and each groups sum is at max 3

double[] arr = {1.01, 1.991, 1.32, 1.4};

Arrays.sort(arr);

int left = 0;

int right = arr.length - 1;

int count = 0;

while(left <= right){

if(left == right){

count++;

break;

}

if(arr[left] + arr[right] <= 3.0){

left++;

right--;

count++;

}

else{

right--;

count++;

}

}

System.out.println(count);

Merge two sorted list

public ListNode mergeTwoLists(ListNode l1, ListNode l2){

if(l1 == null) return l2;

if(l2 == null) return l1;

if(l1.val < l2.val){

l1.next = mergeTwoLists(l1.next, l2);

return l1;

} else{

l2.next = mergeTwoLists(l1, l2.next);

return l2;

}

}

# Find first and last positions of an element in a sorted array

|  |
| --- |
| **class** GFG {      // Function for finding first and last occurrence      // of an elements  **public** **static** **void** findFirstAndLast(**int** arr[], **int** x)      {  **int** n = arr.length;  **int** first = -1, last = -1;  **for** (**int** i = 0; i < n; i++) {  **if** (x != arr[i])  **continue**;  **if** (first == -1)                  first = i;              last = i;          }  **if** (first != -1) {              System.out.println("First Occurrence = " + first);              System.out.println("Last Occurrence = " + last);          }  **else**              System.out.println("Not Found");      }    **public** **static** **void** main(String[] args)      {  **int** arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };  **int** x = 8;          findFirstAndLast(arr, x);      }  } |

**Output:**

First Occurrence = 8nLast Occurrence = 9

Get Average age by gender

Map<Person.Sex, Double> averageAgeByGender =

roster.stream().collect(

Collectors.groupingBy(Person::getGender,

Collectors.averagingInt(Person::getAge)));

|  |
| --- |
| Map<String, Long> noOfMaleAndFemaleEmployees=  employeeList.stream().collect(Collectors.groupingBy(Employee::getGender, Collectors.counting())); |

Why Strings Immutable

### ****1 Performance****

### As we saw previously, String pool exists because Strings are immutable. In turn, it enhances the performance by saving heap memory and faster access of hash implementations when operated with Strings.

### ****2 Synchronization****

Being immutable automatically makes the String thread safe since they won't be changed when accessed from multiple threads.

Hence **immutable objects, in general, can be shared across multiple threads running simultaneously. They're also thread-safe** because if a thread changes the value, then instead of modifying the same, a new String would be created in the String pool. Hence, Strings are safe for multi-threading.

### ****3. Security****

The String is widely used in Java applications to store sensitive pieces of information like usernames, passwords, connection URLs, network connections, etc. It's also used extensively by JVM class loaders while loading classes.

### 4. Class Loading

As discussed in [Creating objects through Reflection in Java with Example](https://programmingmitra.com/2016/05/creating-objects-through-reflection-in-java-with-example.html), we can use Class.forName("class\_name") method to load a class in memory which again calls other methods to do so. And even JVM uses these methods to load classes.

But if you see clearly all of these methods accepts the class name as a string object so Strings are used in java class loading and immutability provides security that correct class is getting loaded by ClassLoader.

Suppose if String would not have been immutable and we are trying to load java.lang.Object which get changed to org.theft.OurObject in between and now all of our objects have a behavior which someone can use to unwanted things.

Why Lambda Function

* **Enables support for parallel processing**: A lambda expression can also enable us to write **parallel processing**because every processor is a multi-core processor nowadays.

# Different ways to create objects in Java

1. Using new keyword
2. Using new instance
3. Using clone() method
4. Using deserialization
5. Using newInstance() method of Constructor class

Deployment Descriptor

Web.xml

Hystrix – Circuit breaker

Zuul – API gateway by Netflix

Second level cache in Spring boot

Caching strategies

Spring security in Spring boot

How to generate jwt token

How to communicate across microservices – Eureka server

You may use a discovery service which is just a spring-boot application with the annotation @EnableEurekaServer (You need to add the appropriate dependency in your pom) Now add to your first (primitive) Microservices the annotation @EnableDiscoveryClient

For sync services you can use either of these 3rd party tools:

1. Hashcorp Consul
2. Netflix Eureka [You may client load balancing using Netflix RIBBON ]
3. NATS etc

For Async you can use messaging solutions like:

1. Redis [use list/streams]
2. ActiveMQ
3. RabitMQ
4. Kafka
5. NATS etc

# HashMap Performance Improvements in JAVA 8 using TREEIFY\_THRESHOLD

In Java 8, HashMap replaces the linked list with another useful data structure i.e. **binary tree** on breaching a certain threshold, which is known as TREEIFY\_THRESHOLD. Once this threshold is reached the linked list of Entries is converted to the TreeNodes which reduces the time complexity from O(n) to O(log(n)).

| Singleton Pattern | Static Class |
| --- | --- |
| Singleton is a design pattern. | Static classes are basically a way of grouping classes together in Java. |
| Memory is allocated once the object is created. | Memory is allocated immediately after any of the class members is accessed. |
| Singleton implementation can either have static members or instance members. | Static classes can contain static members only. |
| It can implement any other interface or base class is required. | It cannot implement the interface or any other base class. |
| Singleton classes can be used as a method parameter. | Static class cannot be used as a method parameter. |
| Singleton pattern uses Heap memory. | Static classes use stack memory. |
| It works within the scope of Garbage Collector as it uses Heap memory. | Out of scope for Garbage Collector as it uses stack memory. |
| It supports Dependency Injection (DI) implementation as Singleton follows OOPS concepts. | It cannot implement Dependency Injection (DI) as DI is Interface-driven. |
| Singleton is an architectural pattern and not a language feature. | Static is a language feature and not an Architectural pattern. |
| Disposal of objects is possible. | It cannot dispose of the static class as there is no instance created. |

### 1. What is the difference between Collection and Stream? ([answer](https://javarevisited.blogspot.com/2014/03/2-examples-of-streams-with-Java8-collections.html))

The main difference between a Collection and a Stream is that Collection contains its elements, but a Stream doesn't. Stream work on a view where elements are actually stored by Collection or array, but unlike other views, any change made on Stream doesn't reflect on the original Collection.  
  
Read more: <https://www.java67.com/2018/10/java-8-stream-and-functional-programming-interview-questions-answers.html#ixzz7NQVzvzur>

### 6. What is the difference between intermediate and terminal operations on Stream? (answer)

The intermediate Stream operation returns another Stream, which means you can further call other methods of Stream class to compose a pipeline.  
  
For example after calling map() or flatMap() you can still call filter() method on Stream.  
  
On the other hand, the terminal operation produces a result other than Streams like a value or a Collection.  
  
Once a terminal method like [forEach()](https://javarevisited.blogspot.com/2015/09/java-8-foreach-loop-example.html#axzz5Hvw8WGhy) or [collect()](https://javarevisited.blogspot.com/2018/05/java-8-filter-map-collect-stream-example.html) is called, you cannot call any other method of Stream or reuse the Stream.  
  
Read more: <https://www.java67.com/2018/10/java-8-stream-and-functional-programming-interview-questions-answers.html#ixzz7NQW5F6lJ>

Java Misc:

[**https://www.javatpoint.com/java-main-method**](https://www.javatpoint.com/java-main-method)

**Internal Working of TreeMap**

[**https://www.dineshonjava.com/internal-working-of-treemap-in-java/**](https://www.dineshonjava.com/internal-working-of-treemap-in-java/)

The **TreeMap** compares each element during insertion in TreeMap. In this case, **TreeMap** uses **comparable** to compare the objects during an insert operation. When we will add any element in **TreeSet** it internally put in **TreeMap**. The **put method** checks if the object of the **comparator** is null then it compares the element based on **comparable**.

# Reverse a LinkedList in Java

**public** **static** LinkedList<String> reverseLinkedList(LinkedList<String> llist)

    {

        LinkedList<String> revLinkedList = **new** LinkedList<String>();

**for** (**int** i = llist.size() - 1; i >= 0; i--) {

            // Append the elements in reverse order

            revLinkedList.add(llist.get(i));

        }

        // Return the reversed arraylist

**return** revLinkedList;

    }