**TCS Interview**

1)Serialization - process of converting the state of an object into a byte stream. done using Serializable interface.

2)Fail-Fast and Fail-Safe Iterators

3)PriorityQueue vs TreeSet

4)How to make a child class non-serializable?

Answer: We need to override the writeObject method in the child class. Maybe we can throw NotSerializableException or any other way of handling

5)Equals and Hashcode overriding

6)final,finalize(),finally differences

7)Output:

public class Main

{

public static void main(String[] args) {

String name="mahesh";

String name1 = new String("mahesh");

String name2="mahesh";

System.out.println(name==name1); //false

System.out.println(name==name2); //true

}

}

8)Output:

class Base{

Base()

{

System.out.println("Base Class");

}

}

class Derived extends Base{

Derived()

{

System.out.println("Derived Class");

}

}

class DeriDerived extends Derived{

DeriDerived()

{

System.out.println("DeriDerived Class");

}

}

public class Main{

public static void main(String args[]){

DeriDerived dd = new DeriDerived();

}

}

Answer:Base Class, Derived Class, DeriDerived Class

9)Question:Non-repeating substring with maximum length (input="abcabcbb"), result should be "abc"

private static void findLongestNonRepeatingSubString(String input){

String longestSubstring="";

List<Character> chars = new ArrayList<>();

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

while(chars.contains(input.charAt(i))){

chars.remove(0);

}

chars.add(input.charAt(i));

String currentSubstring = chars.stream().map(String::valueOf).collect(Collectors.joining(""));

if(currentSubstring.length()>longestSubstring.length()){

longestSubstring=currentSubstring;

}

}

System.out.println(longestSubstring);

}

--------------------------------------------------------------------------------------------------------------------------------------------

**Cognizant Interview**

Introduction, projects worked upon

Java 8 features, java 8 vs 17

Given a list of strings find out the strings starting with the letter 'a'

Stream operations

@EnableAutoConfiguration

How to connect to a database from a spring boot application

@ConfigurationProperties

@Component, @Controller, @Service, @Repository differences

How to call a REST API? Differences between RestTemplate and WebClient?

Is it better to have multiple databases in microservices design?

What's the use of actuators?

OAuth Security implementation?

**Cognizant Interview (TR2)**Questions related to Agile Methodology, Deployment Processes, Project developments, Application Security, Public and Private Certificates

and all general stuff like Kafka, AWS, JIRA tool, Docker, Jenkins etc...

---------------------------------------------------------------------------------------------------------------------------

**EPAM Systems Interview (TR1)**2 coding questions

Given a list of sentences, find the duplicate strings from those sentences

input -> int[] arr={2,8,5,4,9,1} result-> {8,9,9,9,-1,-1}

Asked about custom linked list

Stream operations

Memory management in Java

Garbage collector, Mark and Sweep Algorithm

Collections, Data Structures

Question about generics, How to create a list or collection of objects without all classes like List<A>, List<B>, List<C>

class A{

}

class B extends A{

}

class C extends A{

}

Answer: List<? extends A> list = new ArrayList<>(); //wildcard ? extends A

Time Complexities

ArrayList vs LinkedList -> which is used for insertion and deletion, searching/retrieving and why?

HashSet and HashMap

--------------------------------------------------------------------------------------------------------------------------------------------

**Netcracker Technology Interview (TR1)**Majorly on Core Java and SQL Queries and concepts

Java 8 features including Optional Class, Stream API operations

JDK,JRE,JVM

OOPS Concepts and explanation

String class in Java and the methods within it.

How to create an immutable class

How to write method overriding and method overloading in java code

Output of programs for below:

String s = null;

int i=0;

for(;i<10;){

s+="adding-"+i+++";";

}

System.out.println(s);

List<String> d= Arrays.asList("a", "b","c","B");

for(int i=0; i< d.size();i++){

if(d.get(i).equals("b"))

d.remove(i);

}

System.out.println(d);

Non repetitive characters:

List<String> d= Arrays.asList("a", "b","c","b");

d.stream().distinct().collect(Collectors.toList()); //a,b,c

d.stream().collect(Collectors.groupingBy(word->word,Collectors.counting())); //this approach is tried and interviewer suggested to use below way

d.stream().filter((i->Collections.frequency(d,i)==1)).collect(Collectors.toList());

SQL Commands - DDL,DML Operations

Differences between Delete,Drop and Truncate

<https://onecompiler.com/oracle/437439dwq> - Opened this and asked SQL Queries below

-- create

CREATE TABLE EMPLOYEE (

empId NUMBER PRIMARY KEY,

name VARCHAR2(15) NOT NULL,

dept VARCHAR2(10) NOT NULL,

Salary NUMBER NOT NULL,

ManagerId NUMBER

);

CREATE TABLE BONUS (

empId NUMBER ,

AMOUNT NUMBER NOT NULL

);

-- insert

INSERT INTO EMPLOYEE VALUES (1, 'Ram', 'IT', 1400, NULL);

INSERT INTO EMPLOYEE VALUES (2, 'Sam', 'IT', 2000, 1);

INSERT INTO EMPLOYEE VALUES (3, 'Sergey', 'Sales', 1000, 1);

INSERT INTO EMPLOYEE VALUES (4, 'Trump', 'Accounting', 1200, 2);

INSERT INTO EMPLOYEE VALUES (5, 'Mike', 'Accounting', 1500, 4);

INSERT INTO EMPLOYEE VALUES (6, 'Dev', 'Sales', 1000, 5);

INSERT INTO BONUS VALUES (1, 100);

INSERT INTO BONUS VALUES (4, 200);

INSERT INTO BONUS VALUES (5, 400);

Find the department with highest salary?

select sum(salary),dept from employee group by dept order by 1 desc fetch first row only;

Find the employee having highest overall salary(including bonus)?

select e.empId,e.salary+NVL(b.amount,0) as total from employee e left outer join bonus b

on e.empId=b.empId order by total desc fetch first row only;

--------------------------------------------------------------------------------------------------------------------------------------------

**EPAM (TR1)**1)Memory management in Java -> Heap Memory, Stack Memory usages

2)What is Mark and Sweep Algorithm

3)Java 8 features

4)Detailing about Functional Interface, Stream API

5)Coding question on stream

Given a list of student objects->

i)Find the student with second rank

ii)using flatmap list out all the contact numbers

iii)find out the department and the students in each department. Result should be Map<String,List<String>> deptStudents

iv)find out all the students in karnataka

6)Given a list of integers. Find out all the numbers whose second digit is 1

i) using string

ii)without using string

7)SQL-> ACID properties

8)Query using employees and departments table, find out the department with sum of employee salary as 50000 or more.

9)ConcurrentHashMap usage

10)Optional class

11)Junit annotations -> @Test,@Mock,@Spy, ArgumentCaptor

12)How to test a void method

13)If we want to run the test case multiple times what to do?

**EPAM (TR2)**1)Java -> In case of ConcurrentHashMap is it possible to read and write at once?

2)Spring Core -> How to lazily intiliaze a bean

3)SOLID Principles, Just tell about L in it :)

4)Microservices design patterns

5)Microservices communication - RestTemplate and WebClient in detail

6)@Transactional annotation in detail

7)Default bean scope in spring

if a singleton bean has a reference to a prototype bean and we are calling singleton bean multiple times how many prototype beans will be created?

8)In case of an immutable class , lets say person it has a List<Address> parameter, how to handle it? Is it a deep copy?

9)Let us say there is a restaurant , there are starters,main course and desserts? Which design pattern is used? Answer: Builder

10)How to identify performance bottlenecks?

11)What is API Gateway and its uses?

12)How to externalize application properties? Config server in detail

13)SQL - ACID properties

13)Indexes - Clustered and Non clustered

14)In which case we go for SQL and in which case we choose NoSQL Databases?

15)Have you been part of data migration activities? Explain

16)Have you created architecture/design diagrams? Idea about LLD(Low Level Design) and HLD (High Level Design)? Explain

17)What is nativequery in spring data jpa and how to implement it?

18)What are the possible SQL related exceptions in spring boot?

19)How to troubleshoot API requests with a timed out response?

20)Coding question

package com.maheshkumarvaka.epam.interview;

import java.util.\*;

import java.util.stream.Collectors;

import java.util.stream.\*;

class Movie {

String title;

String director;

Movie(String title, String director) {

this.title = title;

this.director = director;

}

public String getTitle() {

return title;

}

public String getDirector() {

return director;

}

}

public class Main {

public static void main(String[] args) {

List<Movie> movies = Arrays.asList(

new Movie("Jaws", "Steven Spielberg"),

new Movie("Jurassic Park", "Steven Spielberg"),

new Movie("Jackie Brown", "Quentin Tarantino"),

new Movie("Pulp Fiction", "Quentin Tarantino"),

new Movie("Jumanji", "Joe Johnston")

);

/\*

\* find the number of movies of each director where Movie starts with letter J

\*

\* Steven Spielberg 2

\* Quentin Tarantino 1

\* Joe Johnston 1

\*/

// write code below

Map<Object, Long> movieDirs = movies.stream().filter(m->m.getTitle().startsWith("J")).map(movie->movie.getDirector()).collect(Collectors.groupingBy(m->m,Collectors.counting()));

System.out.println(movieDirs);

}

}

21)Kafka questions-> Kafka architecture, what's pub sub model? Topic and Broker?

22)Detailing about SDLC?

23)How to achieve reactive programming? Spring WebFlux?

24)Concurrency and ExecutorService?

25)Can we have multiple terminal operations using stream api? why or why not?

26)How does parallel streams work?

27)map and flatmap difference?

28)All of a sudden you're seeing API latencies? What are the possible scenarios and how to troubleshoot?

29)Can we have GetMapping with a request body?

30)Authentication and Authorization related?If the token is timed out during an api call to a microservice, what will happen? How to do token renewals?

31)Difference between Optional.of and Optioanl.ofNullable

32)Can we implement our own health checks on applications, just like actuators?

33)How to create a custom configuration class, which is a part of auto configuration?

34)There are two classes which are @Component and another is @Configuration? What happens if we interchange? Any errors?

35)Global Exception Handling - @ControllerAdvice

36)Explanation about Spring AOP

37)Distributed Tracing in microservices

38)CompletableFuture and its usage

39)How to achieve scalability for applications?

40)How to handle incosistency in multi thread environments?

**Notes Prepared:**

<https://spring.io/guides/topicals/spring-security-architecture>

<https://docs.spring.io/spring-cloud-config/reference/>

<https://www.baeldung.com/java-blogs>

<https://stackify.com/java-blogs-for-programmers-of-all-levels/>

<https://www.baeldung.com/spring-boot-context-path>

All possible autoconfiguration properties in spring boot :

[https://docs.spring.io/spring-boot/docs/2.0.9.RELEASE/reference/htmlsingle/#common-application-properties](https://docs.spring.io/spring-boot/docs/2.0.9.RELEASE/reference/htmlsingle/" \l "common-application-properties)

<https://www.youtube.com/watch?v=gvQGKRlgop4> - Multithreading perfectly explained

<https://www.youtube.com/watch?v=ePJrt5-G8eM> - functional interface, lambda expressions and stream api functions

<https://www.geeksforgeeks.org/jvm-works-jvm-architecture/?ref=lbp> - JVM Architecture

Java 8 streams:

<https://javaconceptoftheday.com/java-8-interview-sample-coding-questions/>

<https://medium.com/@mehar.chand.cloud/java-stream-coding-interview-questions-part-1-dc39e3575727>

<https://www.digitalocean.com/community/tutorials/java-programming-interview-questions>

<https://blog.devgenius.io/java-8-coding-and-programming-interview-questions-and-answers-62512c44f062>

Interview Preparation:

1)Diamond Problem in Java

It is about multiple inheritance in Java i.e. two parent classes cannot be extended by a single class.

Hence Multiple Inheritance is possible through Interfaces.

This code will through a compilation error.

public class Main extends Abc,Def {

public static void main(String[] args) {

Main main = new Main();

main.commonMethod();

System.out.println("Hello World");

}

}

class Abc {

public void commonMethod(){

System.out.println("Abc Method");

}

}

class Def {

public void commonMethod(){

System.out.println("Def Method");

}

}

Below code works well.

public class Main implements Abc,Def

{

public static void main(String[] args) {

Main main = new Main();

main.commonMethod();

System.out.println("Hello World");

}

@Override

public void commonMethod() {

System.out.println("common");

}

}

interface Abc {

public void commonMethod();

}

interface Def {

public void commonMethod();

}

2)Java 8 Interface:

It has default methods, default methods can have body with some implementation logic. default keyword must be used.

It has static methods, defined using static keyword. These are similar to static methods in classes.

3)Functional Interface in Java 8:

A functional interface is an interface that has exactly one abstract method. It can have multiple default or static methods.

Functional interfaces are used extensively with Java 8’s lambda expressions and method references.

The @FunctionalInterface annotation is optional but helps enforce the functional interface rule.

Java's functional interfaces: Runnable(run method),Callable(call method),Comparable(compareTo method),Comparator(compare method),

Consumer(accept method),Predicate(test method),Supplier(get method)

@FunctionalInterface

interface MyFunctionalInterface {

void execute(); // Abstract method

default void defaultMethod() {

System.out.println("This is a default method.");

}

}

// Using lambda expression

MyFunctionalInterface myFunction = () -> System.out.println("Executing...");

myFunction.execute(); // Output: Executing...

4)Java 9 Interface:

It has private methods, these can be used to share common code between default methods in the same interface.

This code is not accessible outside interface.

interface MyInterface {

private void helperMethod() {

System.out.println("Helper method");

}

default void defaultMethod() {

helperMethod(); // calling private method within the interface

}

}

5)Method References:

Method references provide a shorthand for calling methods through lambda expressions, and they can be applied to different types of methods.

For static methods:

class MyClass {

static void greet() {

System.out.println("Hello from static method!");

}

}

// Using method reference to call the static method

Runnable r = MyClass::greet;

r.run(); // Output: Hello from static method!

For instance methods:

class MyClass {

void greet() {

System.out.println("Hello from instance method!");

}

}

MyClass obj = new MyClass();

// Using method reference to call the instance method

Runnable r = obj::greet;

r.run(); // Output: Hello from instance method!

For constructors:

class MyClass {

MyClass() {

System.out.println("MyClass instance created!");

}

}

// Using constructor reference to create an instance of MyClass

Supplier<MyClass> supplier = MyClass::new;

supplier.get(); // Output: MyClass instance created!

6)What happens if the Test class with null reference call the static method in it?

class Test{

static void greet(){

System.out.println("greeting");

}

}

Test test = null;

test.greet(); //it will print greeting regardless of the reference

7)Equals and hashcode contract:

import java.util.\*;

import java.util.stream.\*;

class Main

{

public static void main(String[] args) {

List<String> argsList = List.of(args);

argsList.stream().forEach(System.out::println);

Student s1=new Student(30,"Mahesh");

Student s2=new Student(30,"Mahesh");

System.out.println(s1.equals(s2));

System.out.println(s1.hashCode());

System.out.println(s2.hashCode());

}

}

class Student{

private int age;

private String name;

Student(int age,String name){

this.age=age;

this.name=name;

}

public int getAge(){

return age;

}

public String getName(){

return name;

}

public void setAge(){

this.age=age;

}

public void setName(){

this.name=name;

}

@Override

public boolean equals(Object o){

if(this==o) return true;

if(o==null||getClass()!=o.getClass()) return false;

Student s = (Student)o;

return age==s.getAge()&&name.equals(s.getName());

}

@Override

public int hashCode(){

return Objects.hash(age,name);

}

}

8)What will be the result?

Set<String> tree = new TreeSet();

tree.add("ABC");

tree.add(null);

System.out.println(tree);

Since TreeSet internally uses TreeMap it throws a null pointer exception during runtime;

9)Serialization:

When we need to convert object's state into a byte stream we implement Serializable interface. It is required in use cases like

saving the object's state to a file, sharing a data over network or in web application sessions where the object data has to be maintained.

public class MyClass implements Serializable {

private static final long serialVersionUID = 1L; // Unique identifier

String name;

int age;

}

10)Use of serialVersionUID

When an object is serialized, its state (i.e., its fields) is written to a stream.

If you later change the class (for example, by adding, removing, or renaming fields), the serialized object may no longer be compatible

with the modified class.

serialVersionUID is a version identifier that Java uses to ensure that the class definition matches the serialized version of the object

during deserialization. If there’s a mismatch between the serialVersionUID of the class in the code and the serialVersionUID in the serialized object,

a InvalidClassException will be thrown.

11)Spring boot vs Spring

i)Spring pom.xml for dependencies needs versions to be provided and corresponding dependency compatibilities to be maintained.

In spring boot we have starter dependencies , which manage all of the required versions.

ii)Spring has XML based, Java based and annotation based configurations but those have to be configured manually.

In spring boot,auto configuration and component scanning can be provided through annotations.

iii)Configurations in xml context files for datasources, transaction managers,sessions etc are configured in spring.

In spring boot, we get rid of xml files and configurations provided in application.properties or application.yml files.

iv)Spring applications need external tomcat server, spring boot has embedded tomcat server for running the applications.

v)Spring needs external or third party dependencies for health metrics, spring boot has actuators to provide health metrics.

12)ApplicationContext vs BeanFactory

<https://www.geeksforgeeks.org/spring-difference-between-beanfactory-and-applicationcontext/>

13)How does a spring boot application work internally?

Spring boot autoconfiguration happens through the jar org.springframework.boot:spring-boot-autoconfigure.

There is a path META\_INF/spring.factories,it has all the auto configuration beans.

@SpringBootApplication->@Configuration+@EnableAutoConfiguration+@ComponentScan

The main method will do the same as executing the executing file.

SpringApplication.run method is the way to kick start the application context.

The run method contains steps below:

Application Context is started->@Configuration classes are searched->Beans in configuration classes are initialized->Those are stored in IOC container

->Dispatcher Servlet followed by default handler mappings, message converters etc..

14)SOLID Principles:

Single Responsibility Principle - each class should have one and only responsibility, reason for changes should be single

Open Closed Principle - Each class should be open for extension, closed for modification

Liskov Substitution Principle - We should be able to change the object reference from subclass to parent class , without any problem.

Interface Segregation Principle - We should not any force any class to implement methods not required from the interface, if needed segregate the interface into smaller interfaces

Dependency Inversion Principle - Dependency on objects i.e. tight coupling should be avoided, though we cannot achieve zero dependency we will have to make sure we are using interface(parent) references.

15)Application Security:

DAST(Dynamic Application Security Testing (DAST) is the process of analyzing a web application through the front-end to find vulnerabilities

through simulated attacks) - Threat Modeling, DAST+IMPT, Thin Client Testing

DevSecOps - SAST, SCA

Annual Compliance Scans - Annual DAST scans, PII(Personal Identifiable Information) Scans

Results Management - Remediation Assitance, Solution Review, Retest, Custom Filters, Results Dashboard

16)Gradle commands for dependency-tree

<https://blog.droidchef.dev/mastering-the-gradle-dependency-tree/>

./gradlew :{moduleName}:dependencies

./gradlew :{yourModuleName}:dependencies --configuration {yourConfigurationName}

17)Output of below program:

public class Main

{

public static void main(String[] args) {

final Integer i=10;

i=11;

System.out.println(i);

}

}

Above program will throw a compile time error. final variable cannot be re-assigned.

18)Output of below program:

public class Main

{

public static void main(String[] args) {

int i=10;

printNumber(i);

}

private void printNumber(final int x){

System.out.println(x);

}

}

Above program will throw a compile time error. non static method reference in static main method.

19)synchronization? race condition?

Due to race condition below i.e. threads switching between each other counter doesn't give the response as 20000.

public class Main{

private static int counter = 0;

public static void main(String args[]){

Thread t1= new Thread(()->{for (int i=0;i<10000;i++) {counter++;}});

Thread t2= new Thread(()->{for (int j=0;j<10000;j++) {counter++;}});

t1.start();

t2.start();

System.out.println(counter);

}

}

By using synchronized as below, counter can be properly incremented to 20000.

public class Main{

private static int counter = 0;

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

Thread t1= new Thread(()->{for (int i=0;i<10000;i++) {increment();}});

Thread t2= new Thread(()->{for (int j=0;j<10000;j++) {increment();}});

t1.start();

t2.start();

t1.join();

t2.join();

System.out.println(counter);

}

private synchronized static void increment(){

counter++;

}

}

It is not recommended to use synchronized at method level, instead use it for a block of code.

20)How to get the list of parameters in get request

21)JPA annotations

22) Stream API Intermediate and Terminal Operations:

Intermediate Operations : map(), flatMap(), filter(), distinct(), sorted(), limit(), skip()

Terminal Operations :forEach(), toArray(), reduce(), collect(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()

23)map vs flatmap

Both are intermediate operations in stream API.

map will take a stream as an input and returns another stream as output.

flatMap will take a stream of stream input and returns flattened stream as output.

Example:

import java.util.\*;

import java.util.Map.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class Main

{

public static void main(String[] args) {

Customer c1 = new Customer("Mahesh",1,Arrays.asList(8600291203L,9640214820L));

Customer c2 = new Customer("Ramesh",2,Arrays.asList(9603825275L,9490213180L));

Customer c3 = new Customer("Vaka",3,Arrays.asList(9603570560L,9849282259L));

List<Customer> customers = Arrays.asList(c1,c2,c3);

System.out.println(customers.stream().map(c->c.getId()).collect(Collectors.toList()));

System.out.println(customers.stream().flatMap(c->c.getMobileNumbers().stream()).collect(Collectors.toList()));

}

}

class Customer{

public String name;

public Integer id;

public List<Long> mobileNumbers;

Customer(String name,Integer id,List<Long> mobileNumbers){

this.name=name;

this.id=id;

this.mobileNumbers=mobileNumbers;

}

public String getName(){

return name;

}

public void setName(String name){

this.name=name;

}

public Integer getId(){

return id;

}

public void setId(Integer id){

this.id=id;

}

public List<Long> getMobileNumbers(){

return mobileNumbers;

}

public void setMobileNumbers(List<Long> mobileNumbers){

this.mobileNumbers=mobileNumbers;

}

@Override

public String toString(){

return name+"-"+id+"-"+mobileNumbers;

}

}

24) Optional Class in Java 8

Majorly used to handle NullPointerException cases.

methods in Optional class: empty, of, ofNullable

Optional<Object> stringOptional = Optional.empty();

System.out.println(stringOptional); //Optional.empty

Customer c1 = new Customer("mahesh",null,Arrays.asList(123L,345L));

//Optional<Integer> idOptional = Optional.of(c1.getId()); //this will throw nullpointer exception

Optional<Integer> idOptional = Optional.ofNullable(c1.getId());

//System.out.println(idOptional.get()); //will throw NoSuchElementException

if(idOptional.isPresent()){

System.out.println(idOptional.get());

}

25) map() reduce()

map is for transforming input stream, reduce is to do an aggregate operation

In case, we need sum or average or multiplication or max value of a list of elements we can use reduce.

List<Integer> numbers = Arrays.asList(1,6,3,2,0);

Integer sum = numbers.stream().reduce(0,(a,b)->a+b); //sum of numbers

Integer highest = numbers.stream().reduce(0,(a,b)->a>b?a:b); //highest number

Integer lowest = numbers.stream().reduce(0,(a,b)->a<b?a:b); //lowest number

Integer multiplied = numbers.stream().reduce(0,(a,b)->a\*b); //multiplication result

List<String> moviesList = Arrays.asList("bahubali","eega","rrr","simhadri");

List<String> companiesList = Arrays.asList("evoke","accenture","mahindra","satyam");

//Longest String in a list.

String longMovie = moviesList.stream().reduce((m1,m2)->m1.length()>m2.length()?m1:m2).get();

String longCompany = companiesList.stream().max(Comparator.comparingInt(String::length)).get();

If there are multiple strings with equal length, below code will help to fetch all those strings

List<String> longestStrings = names.stream().collect(Collectors.groupingBy(String::length))

.entrySet().stream().max(Comparator.comparingInt(Map.Entry::getKey)).map(Map.Entry::getValue).orElse(Collections.emptyList());

System.out.println(longestStrings);

26)Multithreading

Thread LifeCycle - States-> New, Runnable, Running, Blocked, Terminated

Two ways to create a thread i.e. by implementing Runnable interface, by extending Thread class.

public class Main

{

public static void main(String[] args) {

Thread one = new Thread(new Thread1());

Thread two = new Thread(new Thread2());

one.start();

two.start();

}

}

class Thread1 implements Runnable{

public void run(){

for(int i=0;i<10;i++){

System.out.println("Thread1 iteration "+i);

}

}

}

class Thread2 implements Runnable{

public void run(){

for(int i=0;i<10;i++){

System.out.println("Thread2 iteration "+i);

}

}

}

join() method - it is to make sure the remaining threads wait and the current thread completes its execution and lets other threads to join later.

27)ArrayList vs Vector vs LinkedList

ArrayList uses array as a data structure to store elements. Faster in case of retrievals, slow in terms of insertions,updations and deletions.

LinkedList uses doubly linked list as a data structure to store elements. Fast in terms of insertions,updations and deletions , slow with retrieval

ArrayList is not thread safe (not synchronized), Vector is thread safe (synchronized).

28)String vs StringBuffer vs StringBuilder

String is immutable, StringBuffer and StringBuilder are mutable classes.

StringBuffer is thread-safe and synchronized whereas StringBuilder is not.

StringBuilder is more memory-efficient compared to String. Best used for single-threaded operations.

29)Reverse a string:

String name="Mahesh"

StringBuffer sb = new StringBuffer(name);

sb.reverse();

System.out.println(sb);

30)Create an immutable class

The class must be declared as final so that child classes can’t be created.

Data members in the class must be declared private so that direct access is not allowed.

Data members in the class must be declared as final so that we can’t change the value of it after object creation.

A parameterized constructor should initialize all the fields performing a deep copy so that data members can’t be modified with an object reference.

Deep Copy of objects should be performed in the getter methods to return a copy rather than returning the actual object reference)

import java.util.HashMap;

import java.util.Map;

// Class 1

// An immutable class

final class Student {

// Member attributes of final class

private final String name;

private final int regNo;

private final Map<String, String> metadata;

// Constructor of immutable class

// Parameterized constructor

public Student(String name, int regNo,

Map<String, String> metadata)

{

this.name = name;

this.regNo = regNo;

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

// Iterating using for-each loop

for (Map.Entry<String, String> entry :

metadata.entrySet()) {

tempMap.put(entry.getKey(), entry.getValue());

}

this.metadata = tempMap;

}

// Method 1

public String getName() { return name; }

// Method 2

public int getRegNo() { return regNo; }

// Note that there should not be any setters

// Method 3

// User -defined type

// To get meta data

public Map<String, String> getMetadata()

{

// Creating Map with HashMap reference

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

for (Map.Entry<String, String> entry :

this.metadata.entrySet()) {

tempMap.put(entry.getKey(), entry.getValue());

}

return tempMap;

}

}

31)Create a singleton class

class Singleton {

private static Singleton singleton = new Singleton();

private Singleton() {}

public static Singleton getInstance() {

return singleton;

}

public static void main(String[] args) {

Singleton instance = Singleton.getInstance();

System.out.println("Singleton instance obtained: " + instance);

}

}

**Overall practice programming concepts:**

import java.util.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class Main

{

public static void main(String[] args) {

int[] numbers = IntStream.range(1,5).toArray(); //initializes numbers array with 1,2,3,4

int[] numbersInclusive = IntStream.rangeClose(1,5).toArray(); //initializes numbers array with 1,2,3,4,5

for(int number:numbers){

System.out.print(number+" ");

}

String input="hi4you";

//remove digits in string input

System.out.println(input.chars().filter(Character::isLetter).mapToObj(d->String.valueOf((char)d)).collect(Collectors.joining()));

String [] names = new String[]{"how","are","you"};

Arrays.sort(names); //sort string array in ascending order

System.out.println(Arrays.toString(names));

Arrays.sort(names,Collections.reverseOrder());//sort string array in descending order

System.out.println(Arrays.toString(names));

Company c1 = new Company("ABC",1994,20.3);

Company c2 = new Company("DEF",1995,28.3);

List<Company> companies=new ArrayList<>();

companies.add(c1);companies.add(c2);

//Collections.sort(companies, (o1,o2)->(int)o1.getEstYear()-o2.getEstYear()); //lambda expression for comparator

Collections.sort(companies, new CompanyComparator()); //comparator implementation

System.out.println(companies);

companies.stream().sorted(Comparator.comparing(Company::getName)).forEach(System.out::println);//sort list of companies by name using streams

List<String> names = Arrays.asList("mahesh","Mahesh","mahi","kumar","vaka");

List<String> mnames = names.stream().filter(name->name.toUpperCase().startsWith("M")).collect(Collectors.toList());

System.out.println(mnames);

MyFunctionalInterface fi = ()->System.out.println("Hello World");

fi.method();

Supplier<Main> m = Main::new;

m.get();

List<Integer> numbers = Arrays.asList(12,4,8,28,9,0,-1);

Collections.sort(numbers); //ascending order

Collections.reverse(numbers); //reversing the list

System.out.println(numbers);

numbers.stream().sorted().forEach(System.out::println);

numbers.stream().sorted(Comparator.reverseOrder()).forEach(System.out::println);

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

employees.put("mahesh",90000);

employees.put("kumar",80000);

employees.put("vaka",70000);

List<Entry<String,Integer>> entries = new ArrayList<>(employees.entrySet());

employees.entrySet().stream().sorted(Map.Entry.comparingByValue()).forEach(System.out::println); //sort map entries using streams

Collections.sort(entries,(e1,e2)->(int)e1.getValue()-e2.getValue()); //sort map entries using Collections

System.out.println(entries);

}

}

@FunctionalInterface

interface MyFunctionalInterface{

void method();

}

Main(){

System.out.println("constructor called");

}

}

class Company{

public String name;

public int estYear;

public double revenueInMillions;

Company(String name,int estYear,double revenueInMillions){

this.name=name;

this.estYear=estYear;

this.revenueInMillions=revenueInMillions;

}

public String getName(){

return name;

}

public void setName(String name){

this.name=name;

}

public int getEstYear(){

return estYear;

}

public void setEstYear(int estYear){

this.estYear=estYear;

}

public double getRevenueInMillions(){

return revenueInMillions;

}

public void setRevenueInMillions(double revenueInMillions){

this.revenueInMillions=revenueInMillions;

}

@Override

public String toString(){

return name+"-"+estYear+"-"+revenueInMillions;

}

}

class CompanyComparator implements Comparator<Company>{

public int compare(Company c1, Company c2){

return c1.getName().compareTo(c2.getName());

}

}

Streams:

Longest String in a string array

List<String> strings = Arrays.asList("apple", "banana", "cherry", "date", "grapefruit");

Optional<String> longestString = strings.stream().max(Comparator.comparing(String::length));

System.out.println(longestString.get());

Average Age of Persons in a list of objects

List<Person> persons = Arrays.asList(

new Person("Alice", 25),

new Person("Bob", 30),

new Person("Charlie", 35)

);

double averageAge = persons.stream().mapToInt(Person::getAge).average().orElse(0);

Anagrams program:

package com.maheshkumarvaka.practice;

import java.util.\*;

class Anagram {

static List<String> results = new ArrayList<>();

static int count = 0;

private static List<String> compute(String s1, String s2)

{

if (s2.length() <= 1)

{

count++; // no of combination words

results.add(s1.concat(s2));

}

else

{

for (int i = 0; i < s2.length(); i++)

{

String a = s2.substring(i, i + 1);

String b = s2.substring(0, i);

String c = s2.substring(i + 1);

compute(s1 + a, b + c); // recursive method

}

}

return results;

}

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

{

System.out.println("The Anagrams are : ");

System.out.println(compute("", "abc"));

}

}