JAVA INTERVIEW

1) OOPsp principles

2) Method overloading vs method overriding (Explain rules).

3) Why string objects are immutable?4) StringBuffer vs StringBuilder

5) Comparable vs Comparator. Advantages of Comparator

6) How put() method of HashMap works?

7) Importance of equals() and hashcode() method. What is a contract between equals() and hashcode() method.

8) Explain TreeSet.

9) Difference between ArrayList and LinkedList. Use cases of ArrayList and LinkedList.

10) Why Vector and Stack class are not used?

11) Difference between HashMap and Hashtable.

12) Why there is a category of Checked Exceptions? Why some exceptions are checked and others are unchecked? (Explain why)

12) Explain Java 8 new features.

13) Advantages of Lambda Expression. Before Java 8 what you were using??

14) What is a functional interface? In which package functional interfaces are introduced in Java 8? Explain categories of functional interfaces.

15) Explain use of Stream API.

16) Difference between map() and flatMap() method.

17) Explain intermediate and terminal operations in stream api.

18) Prepare most commonly used functional interfaces in Stream API.(hint: Interface and its abstract method. For eg. Predicate<T> interface has "public boolean test(T t) method.)

19) Explain default and static methods of interface.

20) Difference between throw and throws. New features added in Java 7 related to exceptions (Hint: try with resources and multicatch block)

21) How to create custom exception?

22) Explain exception chaining. (Hint - initCause() method)

## Java 8 New Features:

1. Lambda Expression

2. Stream API

3. Default and static methods in Interfaces

4. Functional interfaces

5. Optional class

6. New Date and Time API

7. Functional Programming & Why Functional Programming?

8. Method reference, Constructor reference##

Collections:Collection => Main interfaceList, Set, Map, Queue => Child interfaces of Collection interface.## List => ArrayList, LinkedList, Vector, Stack.We can have duplicate elements.Insertion order is maintained.It allows null values.Duplicate elements are differentiated by means of index.## ArrayList:Internal data structure => Growable array.Use case => Search Operations => ArrayList implements RandomAccess interfaceWorst case => Insertion / Deletion in the middle positions. => Shift operations are costly.## LinkedList:Internal data structure => Doubly linked list.Use case => Insertion / Deletion in the middle positions.Worst case => Search Operations => Sequential search.## Stack:Q. Why we don't use Stack class?## Set => HashSet, LinkedHashSet, TreeSetDoes not allow duplicates.Insertion order is not maintained.One one null value is allowed.##HashSet:Underlying data structure is hashtable.##LinkedHashSet:Underlying data structure hashtable + linkedlistInsertion order is maintained.##TreeSet:Elements are sorted (Natural sorting order(Ascending) or customized sorting order)TreeSet employees = new TreeSet<>(); // Employee objects are comparable.// Employee objects are comparable means Employee class should implement Comparable interface.package java.lang;public interface Comparable {public int compareTo(T t);}public class Employee implements Comparable {private int id;private String name;private double salary;// setXxx() and getXxx()public int compareTo(Employee e2) {// Employee e1 = this; // We can directly use "this".// Comparison logic.}}If a class is not implementing from Comparable interface, we will get an exception "ClassCastException".## Ascending:ife1 < e2 return -vee1 > e2 return +vee1 == e2 return 0## Descending:If e1 < e2 retrun +vee1 > e2 return -vee1 == e2 return 0## Comparator interface:package java.util;public interface Comparator {public int compare(T t1, T t2);}Q. Advantages of Comparator interface over Comparable?class EmployeeNameComparator implements Comparator {public int compare(Employee e1, Employee e2) {// Logic of comparison}}TreeSet employees = new TreeSet<>(new EmployeeNameCompartor());##Map:6Collection of key-value pair (Entry).Keys are unique, values can be duplicated.##Map => HashMap, LinkedHashMap, TreeMap, Hashtable, PropertiesQ. How put() method of map works?Q. When we deal with Set or Map collections, it is recommended to override equals() and hashcode() method, why so?##Functional Interfaces:Interface contains only one abstract method. They may have default and static methods.Java 8 has introduced new annotation @FunctionalInterface.Java 8 has also introduced new package => java.util.functionThis package contains 44 functional interfaces. These functional interfaces are used in Stream API.public interface Calculator {public int calculate(int x, int y);}##1st Waypublic class CalculatorImpl implements Calculator {@Overridepublic int calculate(int x, int y) {return x + y;}}##2nd Way => Anonymous Inner class.Calculator c = new Calculator() {@Overridepublic int calculate(int x, int y) {return x + y;}}##Lambda Expression:A lambda expression is a concise representation of an anonymous function that can be passed around, it does not have a name, it has list of parameters, a body, a return type and possibly a list of exceptions that can be thrown.# Anonymous => Its anonymous because it doesn't have an explicit name.# Function => We say function because a lambda isn't associated with a particular class like a method is. But like a method, a lambda expression has list of parameters, return type and a list of exceptions that can be thrown..# Passed around => A lamda expression can be passed as an argument to a method or stored in a variable.# Concise => You don't need to write a boilerplate code like we do for anonymous [classes.java](http://classes.java/" \o "http://classes.java/" \t "_blank).util.function package interfaces are categorized into 4:1. Predicate2. Consumer3. Supplier4. Function++++++++++++++++++++++++++++++++++++++++++++++public interface Consumer {public void accept(T t);}public interface Predicate {public boolean test(T t);}public interface Supplier {public T get();}public interface Function {public R apply(T t);}## Stream APICollection or Array => Main purpose => To store the data.SetSetFind all employees of HR department having salary > 30000.SELECT \\* FROM employeesWHERE department='HR' AND salary > 30000Declarative way => What you want.Imperative way =>-- Streams lets you manipulate collections of data in a declarative way (you express a query rather than code and adhoc implementation for it).1. A stream does not hold any data. It pulls the data it processes from source (collection/Array/File).2. A stream does not modify the data it processes.3. The source may be unbound-- It is not finite.-- But most of the time, it only means that size of the source is not known at build time.4. One Thread => One TaskSecond Thread => Second TaskMost of the computers have multiple cores (dual, quad, octa)Parallel streams => It uses multiple cores of your computer.Q. What is Stream?=> From technical point of view: Stream is a typed interface.public interface Stream extends BaseStream> {}-- We also have IntStream, LongStream, DoubleStream.Q. How to build Streams?==>1. From collection objectList menu = new ArrayList<>();Stream stream = menu.stream();2. Empty streamStream stream = Stream.empty();3. of() method that accepts single parameter.Stream stream = Stream.of("Anna");4. of() method that accepts multiple parameters.Stream stream = Stream.of("Anna", "Alex", "Bob", "Peter");5. A stream on the lines of text files.Stream stream = Files.lines(path);6. Create a stream from [array.int](http://array.int/) arr[] = { 10, 20, 30, 40, 10, 20, 60, 80, 90, 30};IntStream stream = Arrays.stream(arr);stream.distinct().forEach(n -> System.out.println(n));7. Create a stream with infinite data.+++++++++++++++++++++++++++++++++++++++++++++++++++++++++## Optional<T> class: Added in Java 8. Available in java.util package.It is used to avoid to null reference checks.## Creating Optional Object:1. Empty OptionalOptional<Car> car = Optional.empty();2. Optional From non-null valuesOptional<Car> car = Optional.of(car);// If car object is null, NullPointerException will be thrown immediately once you try to acess properties of Car.3. Optional from null.Optional<Car> car = Optional.ofNullable(car);// If car object is null, then it will return empty Optional.## Method Reference:There are 3 different kinds of method references:1. A method reference to a static method.Integer::parseIntStringUtils::capitalizeList<String> names = Arrays.asList("Anna", "Peter", "Alex", "George");// names.forEach(name -> StringUtils.capitalize(name));names.forEach(StringUtils::capitalize);2) A method reference to an instance method of arbitrary object.For eg. Dish::isVegeterianString::length3) A method reference to an instance method of an existing object.For eg. System.out::println