Hi Good morning sir/mam this is Amritanshu here I am currently working with Infosys and having experience of 1.5 years in IT industry.

Here I am working as a full stack developer with a team of 4 developers 2 testers and 1 scrum master and 1 business architect and 1 product owner from client side here created multiple APIs for an E commerce application using Springboot and front-end components using Angular framework .

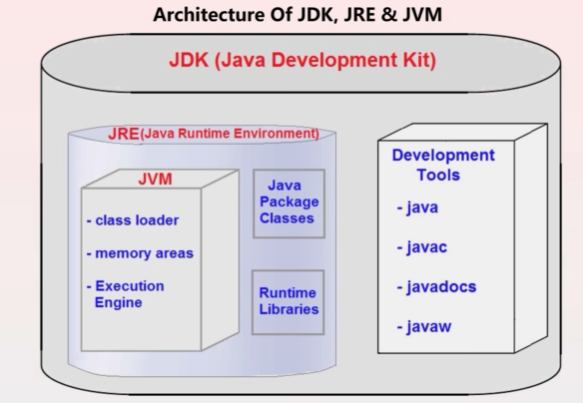
Currently from past 4 months am working with Bank client from Europe here I have worked on creating Statement component for bank using Tibco Enterprise integration tool here also the task was to develop backend services on Tibco BW6 and the front component is Tibco AMX BPM.

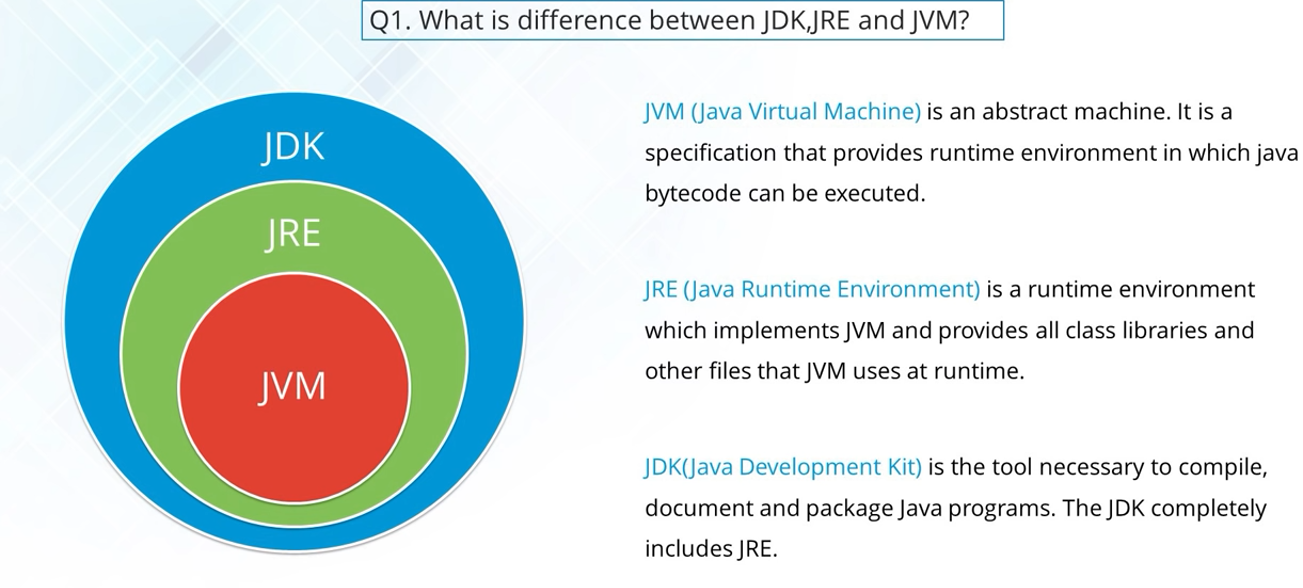
Also at the end ask feedback from interviewer what needs to be excelled

Tasks:

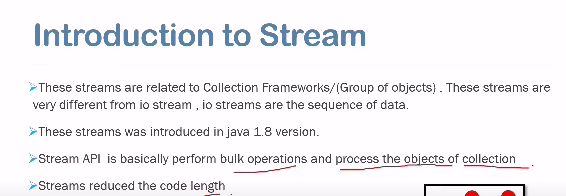
Developing APIs, testing and deploying those APIs .

Attending daily standup calls and scrum meetings for sprint planning





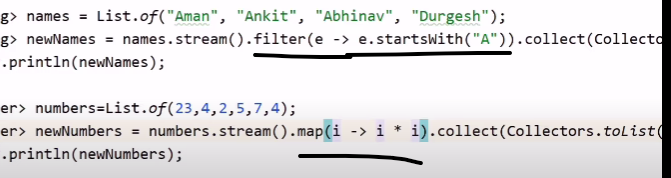
* Stream API (from Java 8 onwards): filter , map – present in java.util.stream



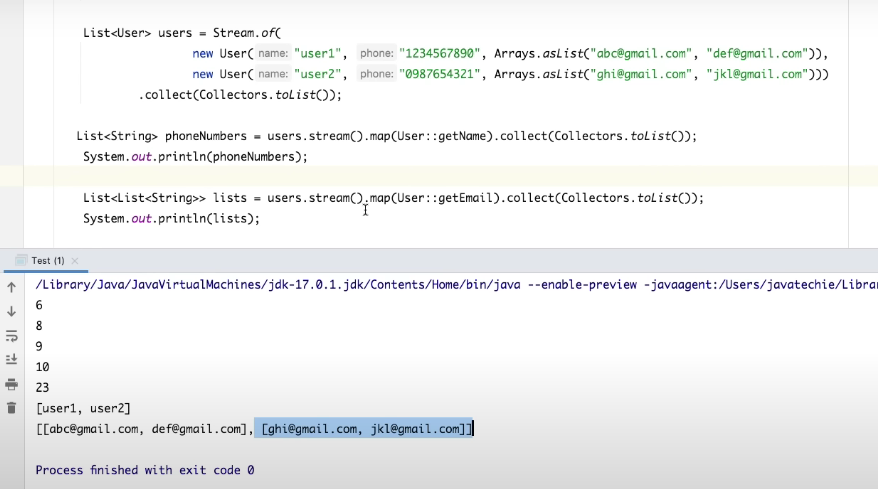
Order for implementing on collections / grp of Objects –

Collections -> convert to stream() -> filter( ) -> collect()

* filter() – Parameters (are Predicate)is lambda expression for it ,will be boolean condition i.e if condition is true means satisfying the condition then it will be further passed else ignored.
* map() – parameter lambda expression is the actual function that return the actual value.

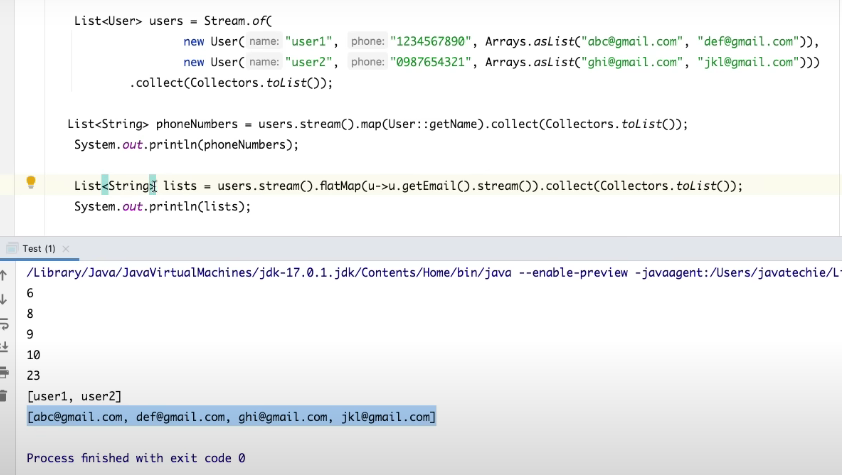


* flatMap() – parameter lambda expression is the actual function that return the actual value as a stream.
* Map vs FlatMap : it is recommended to use map if we have to fetch single value from stream like string or integer but in case we have to fetch a list of object then it is recommended to use flatMap .

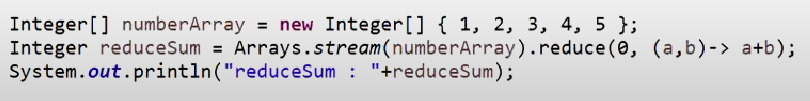


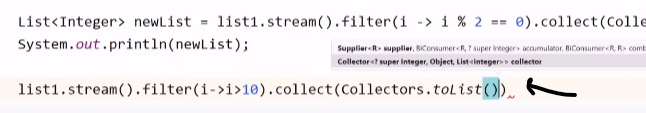
Above is the case if we are using map for fetching a list of email which is giving us output as list of lists.

But instead of that if we use flatMap below is the result.

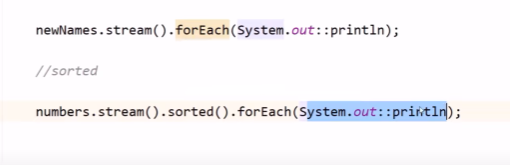


* Collect() – is used for collecting filtered data into the same form of collection.
* Reduce() – is k/a terminal operator coz thy are always present at the end of stream chain



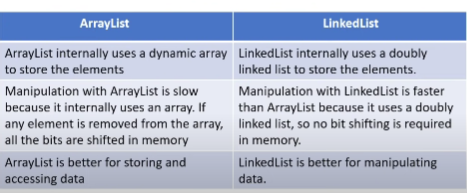


* Similarly sorted() and foreach() is called



* Arraylist vs Linked list:

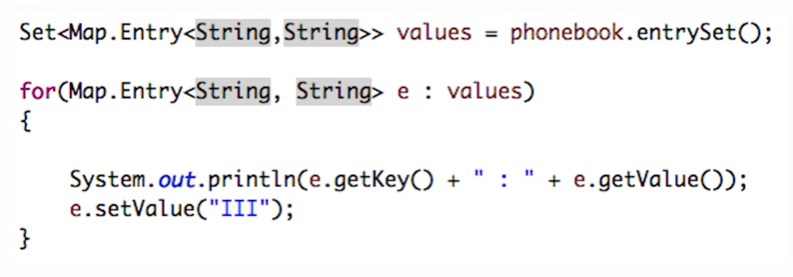
|  |  |
| --- | --- |
| Arraylist | Linkedlist |
|  |  |
| Best choice -> retrieval of data from any place is frequent | Best choice-> Insertion/deletion operation in middle is frequent |
| worst choice-> Insertion/deletion operation in middle is frequent | worst choice -> retrieval of data from any place is frequent |

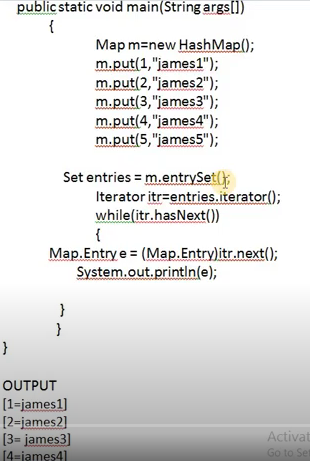


* Hash map vs Linked hashmap vs Tree map:

|  |  |  |
| --- | --- | --- |
| Hash map | Linked hashmap | Tree map |
|  |  |  |
| Doesn’t maintain insertion order | Will Maintain insertion order | Maintain order in sorted order wrt Keys. |

EntrySet() stores both key and value as single entry in set





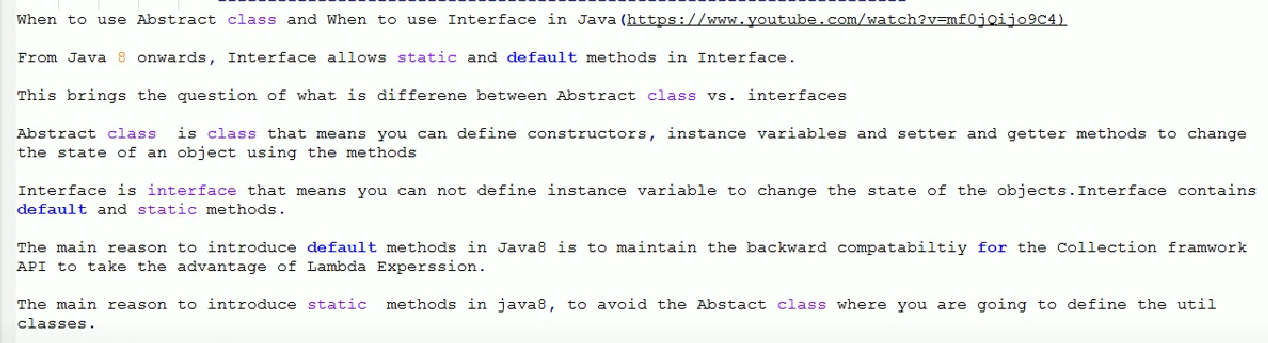
* Multiple inheritance is not allowed in java because if we have a method which is defined in both the parent and now the implemented class is calling it from base class so now the JRE will get confused which parent method to call .

Abstract class vs Interface:

* After Java 8 :

Previously only abstract classes had both concrete and abstract methods but now from java 8 interfaces also can have abstract as well as concrete method.

Interfaces in java 8 can have static and default methods which can have body.

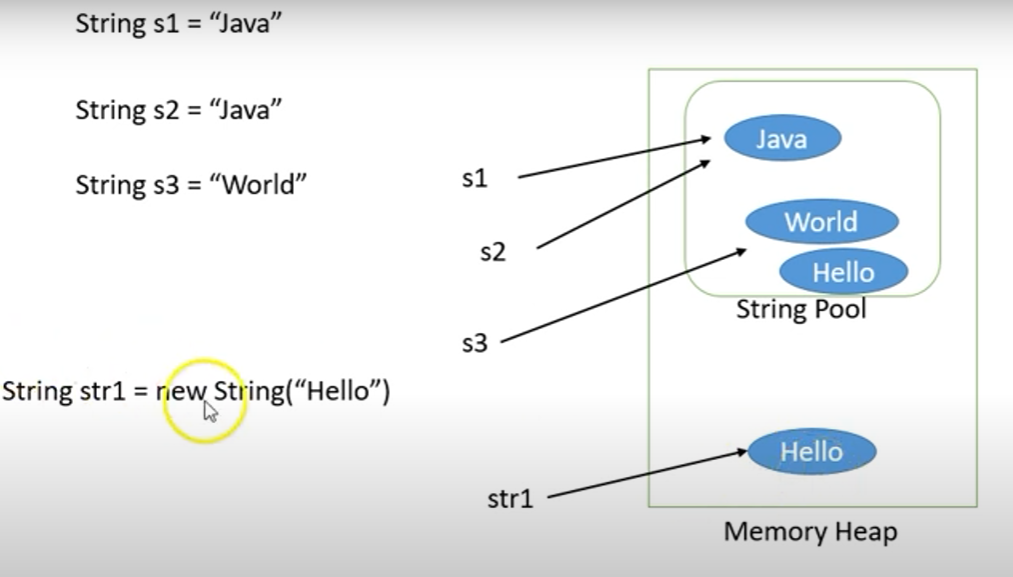


Also by providing body to any new functionality added into the interface we need all its implementations to give body those functionalities So now onwards we can write default methods in interface which can have body.

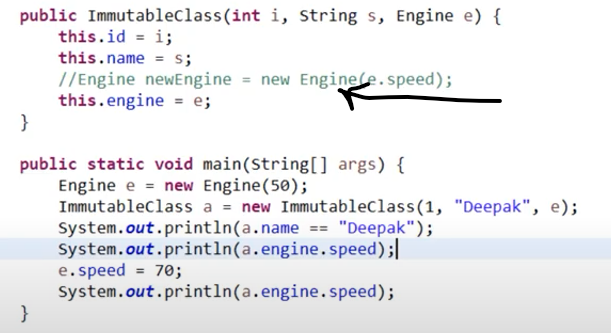
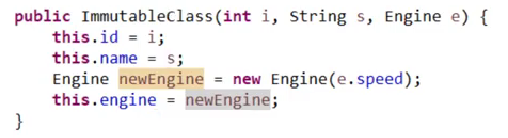
* But now we can face same ambiguity issue as we were facing for multiple inheritance i.e if we are defining method with same name in multiple interface then while calling it from implemented class we will get error (Duplicate default methods) so to overcome that we will need to override that method in implemented Class also.
* Also now if we are extending a class and implementing an interface both having method with same name so here the child class will call the method from the class not from the interface since class has more priority than interface.
* Third rule - if we try overriding any method in interface by making it default already present in Object class (Ex : equals() ) so now it will give error.

Mutable vs Immutable class :

Strings and Wrapper classes are immutable by default in java.



* For making class as immutable make:
* Class final
* Variables private and final
* No setters method
* Make deep copy of object : i.e if using any different class object in immutable class initialize it using new keyword instead of reference initializing.

Since String is immutable in Java, whenever we do String manipulation like concatenation, substring, etc. it generates a new String and discards the older String for garbage collection.

Strings are immutable but it can be made mutable using String builder and String buffer:

* Stringbuffer : are thread safe i.e if one thread is accessing stringbuilder then other will wait.

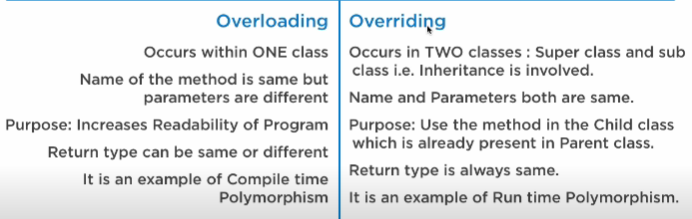
Since it is thread safe it is less efficient in performance i.e time complexity.

* Stringbuilder : are not thread safe. Since it is not thread safe it is more efficient in performance i.e time complexity.

== vs equals

* == is for reference comparison -> true if both object pointing to same reference/address.
* Equals is for content comparison -> true if content is same even if reference Is different .

Method overloading vs method overiding



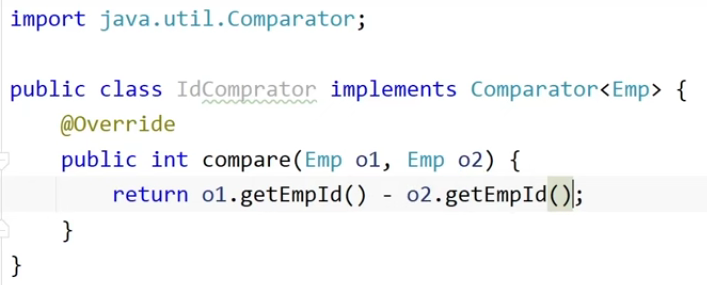
* Comparable and comparator : both interfaces are used to sort the collection elements/objects.

|  |  |
| --- | --- |
| **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. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

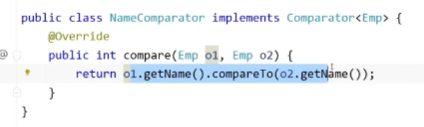
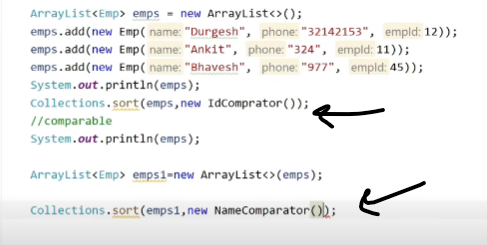
* Comparable interface : used for sorting collections on basis of single element by implementing comparable interface and defining compareTo() in it in the same class for which sorting needs to be done.



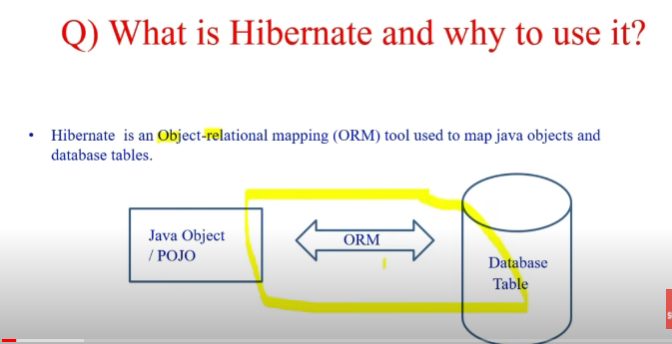
* Comparator interface : used for sorting collections on basis of multiple elements
* Using id

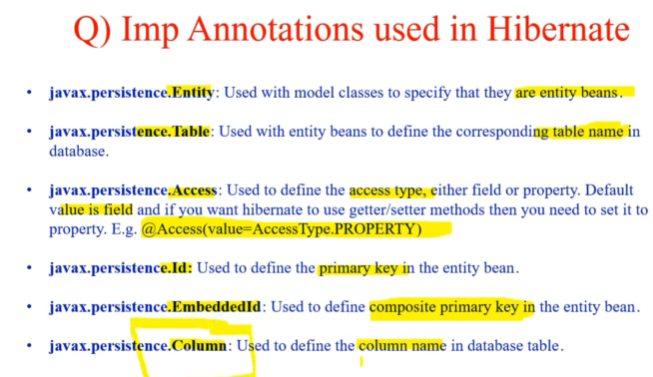
 

* Similarly we can do using name element

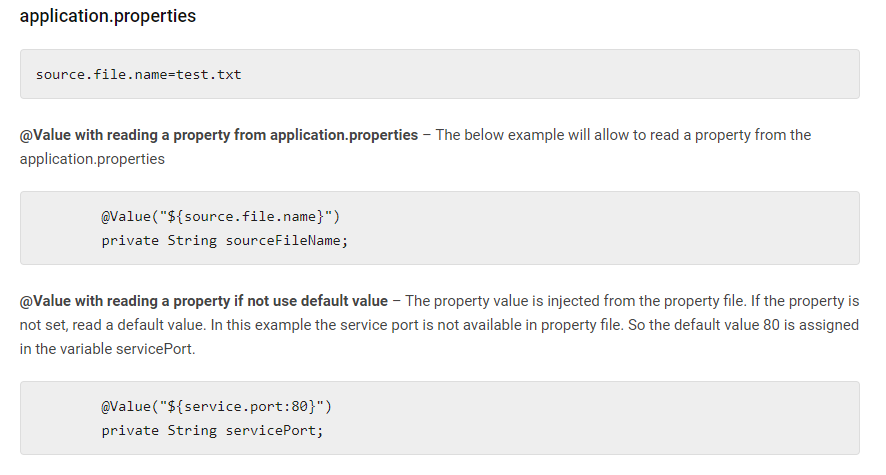
* Hibernate annotations and uses ex: @Entity,@Id ,@Pattern….





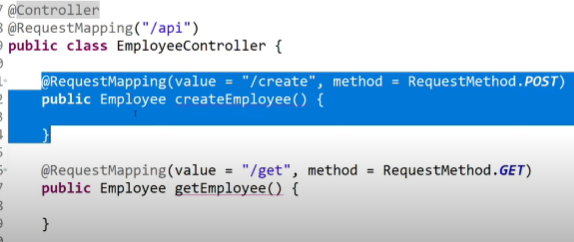
[*https://www.javatpoint.com/hibernate-with-annotation*](https://www.javatpoint.com/hibernate-with-annotation)

@Value – used to assign value to a variable in a class. Mainly used to assign properties value in the class variables.



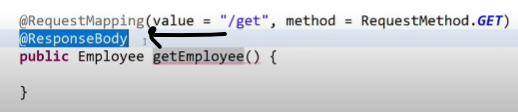
@Controller vs @RestController

* @Controller



Methods in normal controller class returns the View (Complete UI) that is the HTML/CSS/JavaScript…

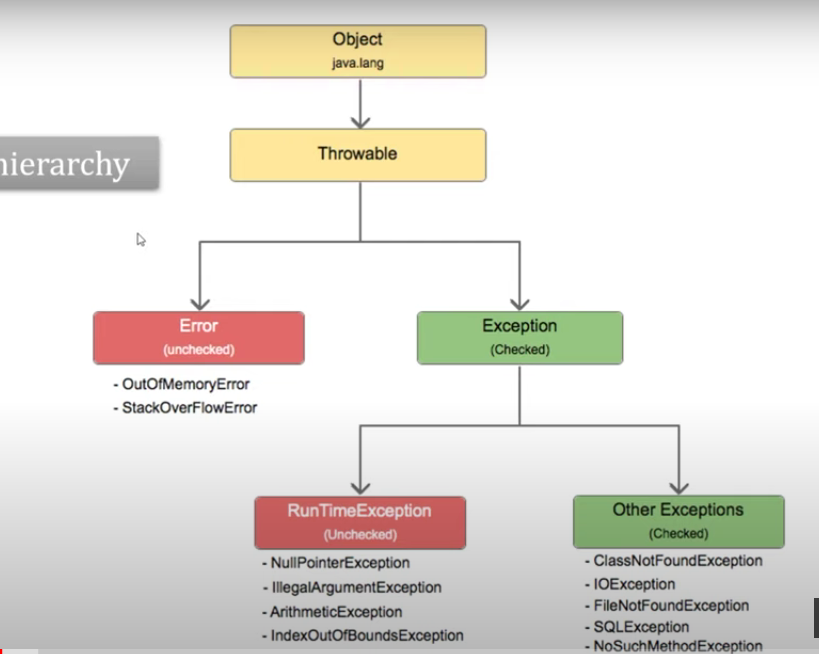
So if we want out method to send HTTP response for Rest API as JSON/XML so we will need to add one more annotation i.e @ResponseBody



* @RestController = @Controller + @ResponseBody

So if we use @RestController then we don’t need to explicitly add @ResponseBody in the API

* Exception handelling and types of exception:
* Exception : Abnormal condition in program which terminates application abruptly.
* While in case of inheritance if any method of child class is throwing an exception then its parent should also throw the same exception but the vice versa isn’t true i.e if a parent class method is throwing any exception then its child may or may not throw that exception.



All the exceptions weather it is checked or unchecked both occur at runtime only.

* Checked exception : are the exceptions which are checked by compiler at compile time only, and warns the developer that the exception may occur in future so it asks u to cover it with try catch or by throws.
* Unchecked Exception : are the exceptions which are ignored by compiler but it happens at run time. And then causes program to stop abruptly.

Multi threading in java

Abstraction in detail practical

Hibernate revision for Spring from end to end controller to repository

Programming logical questions on collections java 8

@RequestParam, @PathVariable … all controller class annotations and their uses

hashmap vs hash table

syncronization

map vs flatmap

collection vs collections

how hashmap internally works

stream.put

serializable

sleep

wait

Can static method be overridden or overloaded

String class - how it stores value -> String i='dsds'; i='dsa';

* Can we store again new string I if yes then how and where

 String S1="ABC";

String S2= S1+ "  "+ "PQR"; -> here hw many objects will be created and in which memory

* final Student s = new Student("a",100);

s.setName("b"); -> can we change name to ‘b’.

Polymorphism real time working

* in real time project we use RUNTIME POLYMORPHISM as in dependency injection u.e here we inject parent i.e interface and can all its implemetions methods from it.
* Also if in our interface for a method declaration we have return type as parent class then in its implementation we can also return its child class as return type it will also be run time polymorphism.

Try with resource

Java 8 core concept

Java Core

Microservices

Springboot

SQL database queries and joins

Logic for getting top 3 employees salary from a list having employee and salary

* In below code which method will be called

public class Test {​

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

method(null);

}​

public static void method(Object o) {​

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

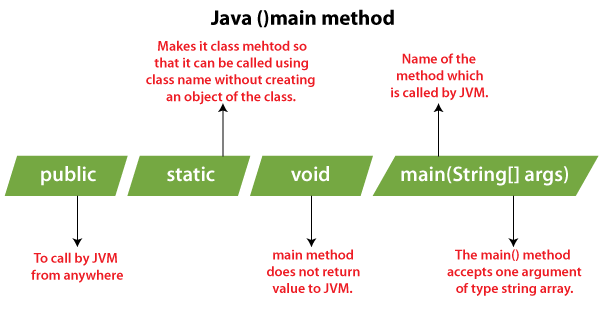
}​

public static void method(String s) {​

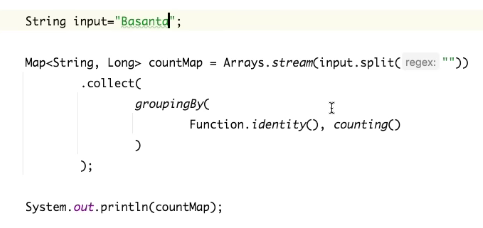
System.out.println("String method");

}​

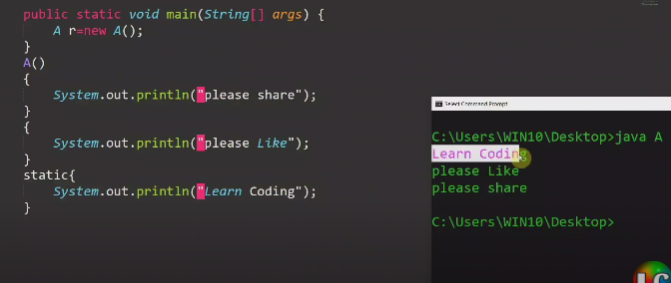
}​

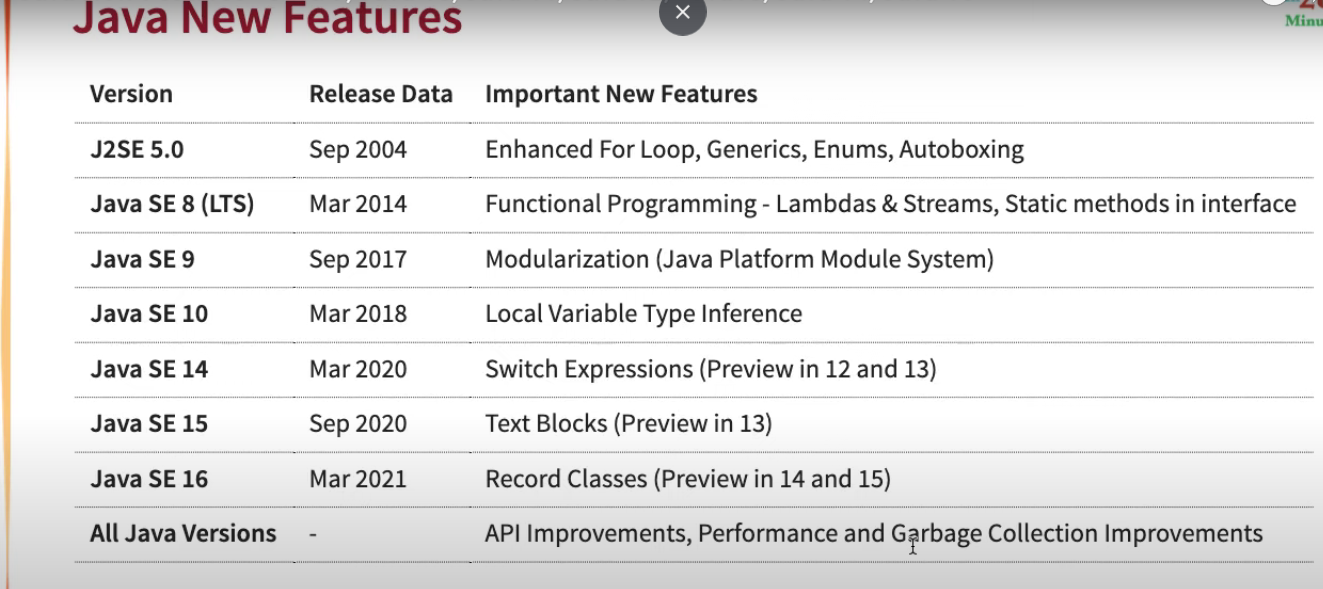


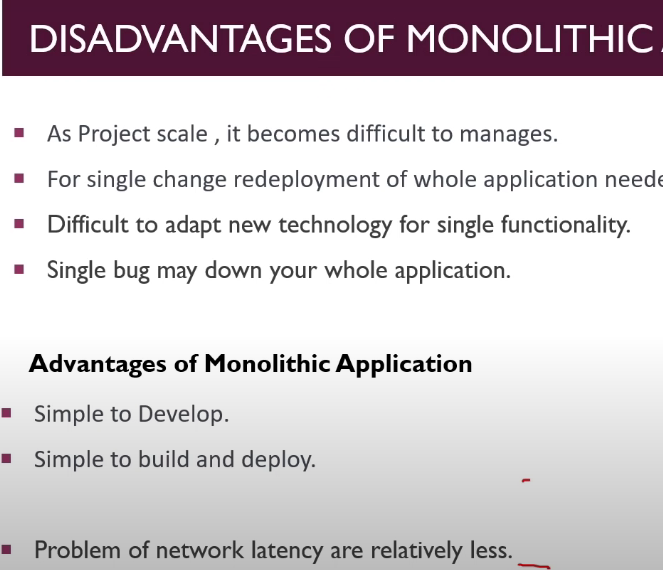
* Static Block get executed at the time .class file gets loaded into JVM even before object creation and constructor calling.
* Static variables can be declared at class level not at method level.
* Accessed using class name
* Execution order : 1. Static Block 2.Instance Block 3. Constructor



Above is the code for counting the frequency of each character in a string using stream form java 8.







Coding questions pending to do :

1. <https://www.youtube.com/watch?v=ItRZRx8kvwY&list=PLSIpQf0NbcCltzNFrOJkQ4J4AAjW3TSmA&index=4&ab_channel=JAVAAID-CodingInterviewPreparation>
2. <https://www.youtube.com/watch?v=pVkHLciuank&list=PLSIpQf0NbcCltzNFrOJkQ4J4AAjW3TSmA&index=6&ab_channel=JAVAAID-CodingInterviewPreparation>
3. <https://www.youtube.com/watch?v=hDhf04AJIRs&list=PLSIpQf0NbcCltzNFrOJkQ4J4AAjW3TSmA&index=10&ab_channel=JAVAAID-CodingInterviewPreparation>
4. <https://www.youtube.com/watch?v=PtFAJTCKNn8&list=PLSIpQf0NbcCltzNFrOJkQ4J4AAjW3TSmA&index=13&ab_channel=JAVAAID-CodingInterviewPreparation>

