**Java 8**

* The main Objective of java8 is to reduce the number of lines of code by implementing functional programming.
* The main features include:

1. Lambda expressions
2. Functional Interface
3. Default methods and static methods
4. Predefined Functional interfaces like

Predicate

Function

Consumer

Producer etc

1. Double colon operator(::)

Method reference

Constructor reference

1. Streams
2. Date and Time API
3. Optional Class
4. Nashorn javascript

**Lambda Expressions:**

1. It is already available in LISP, Python, C, C++, Ruby, Scala. Finally Java also implemented this feature.
2. It is an anonymous function, without return type, without modifiers.
3. Public void m1()

{

System.out.println(“hello”);

}

1. As it doesn’t have name, modifier, return type remove them. But to convey that it is lambda expression, we need to use “->” symbol.

() -> System.out.println(“hello”);

1. Compiler gets the data type automatically sometimes. In that case we can remove the datatype.

(a,b) -> System.out.println(a+b);

1. If the function body contains only one line and it is return stmt, u can remove the keyword return. Below are the examples.

n->n\*n;

s->s.length();

**Functional Interfaces:**

* An interface contains only single abstract method is called functional interface.
* Collections.sort(lst, (emp1,emp2) -> (emp1.eno<emp2.eno)?-1:(emp1.eno>emp2.eno)?1:0);
* Collections.sort(lst, (e1,e2)->e1.ename.compareTo(e2.ename));

**Default Methods and Static Methods:**

* Without effecting implementation classes if we want to add new methods to the interface(extending interface functionality), we use default methods.
* Also default methods avoid the burden of implementing all unnecessary methods of interface in the implementation class. I will just implement whatever I need in my class.
* We can’t make Object class methods as default methods in our interfaces.
* Interface static methods can only be called using Interface name even if it is implementation class. By default these methods won’t be available in implementation classes.
* In interface you can define, public abstract or default or static methods only. All other methods are invalid.
* If same default method is available in two different interfaces and if we want to implement both of them in a single class, it is not possible. So we need to create one more interface which extending both of these and override this methods by calling A.super.m1() and then implement.
* From 1.8 onwards, default methods and public static methods are also allowed. Public static methods are useful for common utilities.

**Predefined Functional Interfaces:**

* These will be useful to use lambda expression in our daily programming. They include:

1. Predicate
2. Function
3. Consumer
4. Supplier

* Two argument Predefined Functional Interfaces:

1. BiPredicate
2. BiFunction
3. BiConsumer

* Primitive Functional Interfaces

1. IntPredicate
2. IntFunction
3. IntConsumer

**Predicate:**

* It is a functional interface contains only one public abstract boolean test(T t) abstract method.
* It is useful in lambda expressions to replace the conditional check code.

Ex:

Predicate p<Integer> p = i -> i>10;

sop(p.test(15)); -> true

**Function:**

* It is a functional interface contains only one public abstract <T2> apply(T1 t1, T2 t2) abstract method.
* It is useful in lambda expressions to chk and return some value.

Function<String, Integer> f = s1->s1.length();

Sop(f.apply(“string”));

**Consumer:**

* If we need to give input and perform some operations and don’t expect any output then we should go for Consumer.
* It contains void accept(T t) method.

Consumer<String> c = s1 -> sop(s1);

c.accept(“hello”);

**Supplier:**

* It won’t accept any argument but return some value.
* It contains R get() method.

Supplier<String> s = () -> {some random logic

return “output”

}

Sop(s.get());

* If we use Predicate functional interface in case if we are dealing with primitive data types, there would a performance issue with autoboxing and autounboxing. To avoid this we have primitive types of Predicate, Function etc. For BiPredicate there is no such facility.
* IntFunction<Integer> intFun = (i) -> i\*i;
* LongToIntFunction, ToDoubleFunction etc.
* IntToDoubleFunction intToDblFun = (i) -> Math.*sqrt*(i);
* System.***out***.println(intToDblFun.applyAsDouble(5));
* If both the input and return types are same in Function we can directly use UnoryOperator. Primitive version also applicable in this.
* IntUnaryOperator intUnoFun = (i) -> i\*i;
* For BiFunction, if both input types and return type are same, then we can go for BinaryOperator.
* BinaryOperator<String> binOp = (fname, lname) -> fname.concat(" ").concat(lname);
* Primitive types also available in this. IntBinaryOperator, DoubleBinaryOperator, LongBinaryOperator.

**Method reference and Constructor reference:**

* Instead of writing code for lambda expression always, we can create a static or instance method implementing the same logic and we can refer that using method reference. This is useful for code reusability.
* For static method, we can use <class\_name>::<method\_name>
* For instance method, we have to create the object and use <object\_ref>::<method\_name>
* In case the abstract method implementation has to return a new object, in that case we can go for constructor reference.
* Interf i = <Class\_name>::new
* <Class\_name> ref = i.get();

**Streams:**

* To process the data present in the collections then Streams will be useful.
* filter method is used to filer some data. Eg., to get only even numbers from the list.
* map method is used to apply some function on every object. Eg., to convert all strings to upper class.

**Anonymous inner classes vs Lambda expressions:**

* Anonymous inner class is a class which doesn't have name and either extends parent class or implements parent interface.
* Lambda expression can replace inner classes only in case if it is Functional interface.

**Default and Static methods in interfaces:**

* We can’t add Object class methods as default methods as there is no necessity of these methods as they would be already available in any class, so JVM won’t allow.
* As interfaces from 1.8 having the default methods, we would encounter with diamond problem in multiple inheritance when two interfaces having the same default method. In this case, we need to override such method inside the implementation class. We can call interface default method inside it as below.

InterfaceName.super.methodName();

* Default methods are by default available inside the implementation class but the static methods will never directly available inside the implementation class. We need to call them only by interface name only like below.

intefaceName.methodName();

* We can write public static void main methods inside the interface as well from 1.8 and can run the interface.
* As static methods doesn’t need any object to be created, we can write them in interfaces and can call directly with interface name without creating the class which is costly operation.
* Static methods are introduced in interfaces to write the common utility methods which can be used in various classes. No extra class needed for these utility methods.

**Spring Boot**

* how do I enable auto reload of spring boot changes? -> By using spring boot developer tools. Add spring-boot-devtools dependency and restart ur app.
* How do u configure two different envs? -> using profiles.

Eg., application-prod.properties and application-dev.properies then use spring.profiles.active = prod in application.properties.

* Web.xml, dispatcher-servlet.xml, applicationContext.xml

**Spring boot Features:**

* **Auto configure**  - Detects and automatically configures the Spring applications based on the added dependencies
* **CLI** - Command Line Interface to start, test and stop Spring Boot applications from command prompt (Not discussed in this course)
* **Actuator** - Enables enterprise features and gives the insight of the application
* **In built servers**
* **Spring boot starters**
* **Spring Initializer – which gives us the spring boot project structure to start our business logic without much effort.**
* **Provides in-memory database like h2 for testing.**

**Session:**

Assessment:

<https://lex.infosysapps.com/toc/lex_6783733217754302000/about>

Course: <https://lex.infosysapps.com/toc/lex_6783733217754302000/about>

Assessment : <https://lex.infosysapps.com/viewer/lex_6783733217754302000/lex_15407698985702472000>

Settings.xml  : <https://infygit.ad.infosys.com/Rasmi_G/Demos/blob/master/settings/Settings.zip>

@PropertySource(value = {classpath:config.properties})

**Spring microservices:**

* Microservice is an architecture to provide solution for the issues with monolithic applications.
* SOA also does the same but Microservices having easy way of implementations from various vendors like springs, netflix.
* SOA uses ESB(routing technology) to solve the above problems.
* Bootstrap.properties
* Spring-cloud-config-server dependency makes our application a cloud.
* Client side load balancing. By using ribbon we can do client side load balancing. We can write code to choose the server on which it has to be run.
* To get ribbon functionality we need to add Ribbon dependency.
* Eureka will monitor all the servers and provide only healthy state server details back to ribbon.
* <https://lex.infosysapps.com/viewer/lex_4795770912300205000/lex_29215824857341200000> -> assessment link.
* <https://lex.infosysapps.com/viewer/lex_auth_0126193469141893122170> -> spring microservices playground.
* Add Eureka dependency and add @EnableEurekaServer annotation below the @SpringBootApplication.
* @EnableDiscoveryClient in client spring boot application.
* Hysterix -> add this dependency to your application to get fallback feature into your application. Eg., in case a service is not available in b/n a transaction, Hysterix helps us to complete that transaction. It gives the failed requests to fall back code and stop sending requests to that failed microservice after a threshold limit reaches. It waits some time and again start sending reqs to that micro service, if still failing, the cycle repeats.

**OAuth authentication:**

* Register your microservice with oAuth provider.
* Provide the token id and secure in application.yml file which are given by oAuth provider.
* Include security dependencies in the pom.xml.

**API Gateway:**

**Feign:** is used to call a microservice from other MS very easily. It is an alternate for Rest template. It enables easy communication b/n MSs.

**Int 1:**

1. Difference b/n log4j and logback?
2. Why spring use logback instead of log4j?
3. How to change .jar to .war in spring boot?

* **Ans**: Add spring-boot-starter-tomcat dependency into pom.xml file.
* change packaging to war instead of jar in pom.xml file.
* Go to your Spring Boot main class and extends SpringBootServletInitializer abstract class.
* Override the SpringApplicationBuilder method of  
  SpringBootServletInitializer abstract class .
* Now just clean and build your project you will get your war file inside target folder of your project.

1. How to configure multiple datasources in spring boot?
2. What is parent tag in pom.xml?
3. **Diff b/n process and thread?**

* **Ans:** A Program in the execution is called the process whereas; A thread is a subset of the process
* Processes are independent whereas threads are the subset of process.
* Process have different address space in memory, while threads contain a shared address space.
* Context switching is faster between the threads as compared to processes.
* Inter-process communication is slower and expensive than inter-thread communication.
* Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.

1. **What is inter thread communication?**

* **Ans:** The process of communication between synchronized threads is termed as inter-thread communication.
* Inter-thread communication is used to avoid thread polling in Java.
* The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.
* It can be obtained by wait(), notify(), and notifyAll() methods.
* The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.
* public final void wait()

1. **Why wait() method should be called from synchronized block?**

**Ans:** Wait method is meant for inter thread communication b/n synchronized methods. If we call it from out side synchronized block,  it will throw **java.lang.IllegalMonitorStateException** exception. Moreover, we need wait() method for inter-thread communication with notify() and notifyAll(). Therefore It must be present in the synchronized block for the proper and correct communication.

1. **What are the states of thread?**

* **Ans: New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
* **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
* **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
* **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
* **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.

1. **Diff b/n preemptive and time slice scheduling?**

**Ans:** Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

1. **What is context switching?**

**Ans:** In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resumed from the same point later. Context switching enables the multiple processes to share the same CPU.

1. **Diff b/n Thread class and Runnable interface?**

* **Ans:** By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).
* By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
* Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

1. **What join() method does/**

**Ans:** The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

1. **Diff b/n wait() and sleep()?**

**Ans:** wait method is for inter thread communication b/n synchronized methods, so it releases the lock on the obj while sleep method sleeps certain time and gives time to other priority task and it doesn’t release the lock.

wait method is in Object class and sleep method is in Thread class.

Wait method resumes on notify or notifyAll() methods whereas sleep method resumes after the time lapses.

1. **Can we make the user thread as daemon thread if the thread is started?**

Ans: No, if you do so, it will throw IllegalThreadStateException. Therefore, we can only create a daemon thread before starting the thread.

1. **What is shutdown hook?**

Ans: The shutdown hook is a thread that is invoked implicitly before JVM shuts down. So we can use it to perform clean up the resource or save the state when JVM shuts down normally or abruptly. We can add shutdown hook by using the following method:

1. **public** **void** addShutdownHook(Thread hook){}
2. Runtime r=Runtime.getRuntime();
3. r.addShutdownHook(**new** MyThread());

Some important points about shutdown hooks are :

* Shutdown hooks initialized but can only be started when JVM shutdown occurred.
* Shutdown hooks are more reliable than the finalizer() because there are very fewer chances that shutdown hooks not run.
* The shutdown hook can be stopped by calling the halt(int) method of Runtime class.

1. **When should we interrupt a thread?**

Ans: We should interrupt a thread when we want to break out the sleep or wait state of a thread. We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

1. **How is the safety of a thread achieved?**

Ans: If a method or class object can be used by multiple threads at a time without any race condition, then the class is thread-safe. Thread safety is used to make a program safe to use in multithreaded programming. It can be achieved by the following ways:

* Synchronization
* Using Volatile keyword
* Using a lock based mechanism
* Use of atomic wrapper classes

1. **What is race-condition?**

Ans: A Race condition is a problem which occurs in the multithreaded programming when various threads execute simultaneously accessing a shared resource at the same time. The proper use of synchronization can avoid the Race condition.

1. **What do you understand by thread pool?**

* Java Thread pool represents a group of worker threads, which are waiting for the task to be allocated.
* Threads in the thread pool are supervised by the service provider which pulls one thread from the pool and assign a job to it.
* After completion of the given task, thread again came to the thread pool.
* The size of the thread pool depends on the total number of threads kept at reserve for execution.

The advantages of the thread pool are :

* Using a thread pool, performance can be enhanced.
* Using a thread pool, better system stability can occur.

1. **What is BlockingQueue?**

Ans: The java.util.concurrent.BlockingQueue is the subinterface of Queue that supports the operations such as waiting for the space availability before inserting a new value or waiting for the queue to become non-empty before retrieving an element from it. Consider the following example.

1. <https://www.javatpoint.com/java-multithreading-interview-questions>
2. **How to create immutable class in java?**

Ans: create the class as final so that there won’t be any child classes for it.

Declare all data members as final inside the class, so that we can’t change their value after object creation.

One parameterized constructor.

Create only getter methods **not** the **setter** methods.

1. **Find out the combination of numbers having sum as 5 in a given array in java**



1. **JVM Memory management:**

**Ans:** Java memory divided into two parts stack and heap. Heap stores the objects.

Heap again divided into two parts, young memory and old memory.

Young memory again divided into 3 parts, Eden, Survivor memory(S0), Survivor memory(S1)

All newly created objects will be stored in Eden memory part of young memory.

In case Eden memory was filled, Minor GC will be performed and few of the objs from Eden will move to S0. If S0 fills, moves to S1. This is very minor and GC1, GC2, GC3…GCn will be called in short span.

During this Minor GC, it check which objs are important for survival and JVM moves them to survivor memory. Also it removes the objs which doesn’t have reference.

When survivor memory gets filled, java creates new objects for these in old generation memory with new references.

In case this old memory gets filled Major GC will be called. Major GC takes longer time. Major GC degrades the performance hence our code should be in such way that, Major GC shouldn’t called every time.

There is one more kind of memory Permanent generation which is not part of Heap memory and it will be generated during run time and will store all meta data of our classes and methods.

This perm generation contains a part called Method area in which all static variable, constant variable, complete class structure will be saved.

In old generation, we have one more area called Constant Pool which stores all the immutable objs including String objs, String pools.

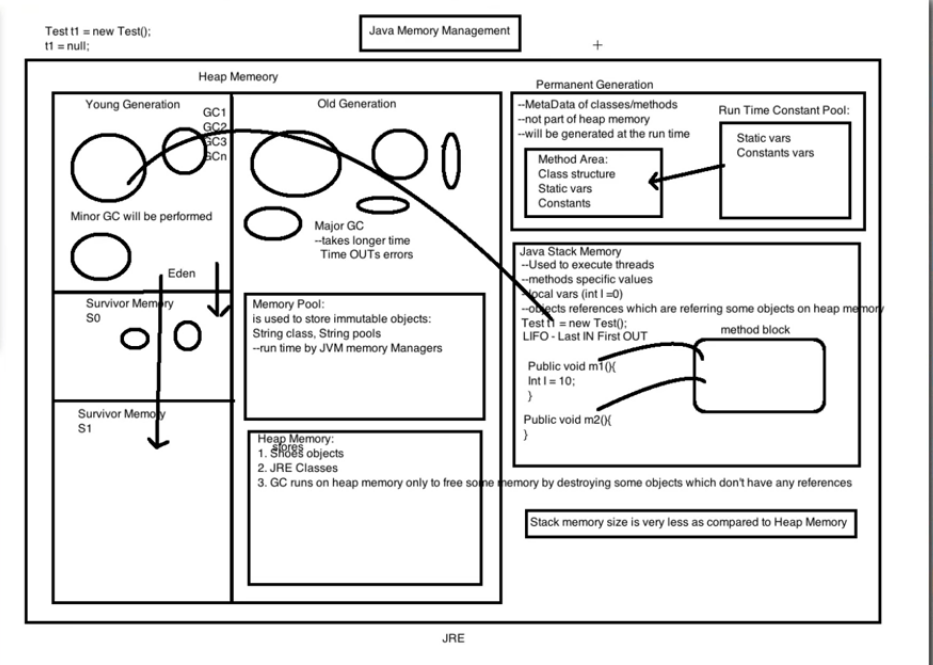
This will be defined at run time by JVM memory managers.

Permanent Generation also consists of Run time Constant Pool, saves constant variable and static vars.

One more memory area is Java Stack memory which is out of heap memory, used to execute threads, method specific values, local vars, obj references of some objs present in heap.

If we write Test t1 = new Test(); obj will be saved in heap and t1 ref will be saved in stack memory.

Whenever we create a method, a block will be created in stack memory and given to that method. When the execution of that method completes, this memory will be given to another method.



1. **ACID:**

* Atomicity -> commit or rollback
* Consistency -> if database is consistent before a transaction, it should be consistent after the transaction as well. Transactions shouldn’t break the consistency of DB.
* Isolation -> DB should be in a position to persist the latest committed transactions even if the DB down after commit operation. The DB should hold this modified data and persist this data in disk after it is up.
* Durability -> if more than one transaction is happening in parallel, one transaction shouldn’t impact the other. These should complete just like only one transaction is happening at a time.

1. **SOLID principles:**

* Single responsibility -> A class should have single responsibility.
* Open for Extension Closed for modification -> we stop ourselves from modifying existing code and causing potential new bugs in an otherwise happy application. Of course, the one exception to the rule is when fixing bugs in existing code.
* Liskov substitution -> if class A is a subtype of class B, then we should be able to replace B with A without disrupting the behavior of our program.
* Interface Segregation -> larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.
* Dependency Inversion -> The principle of Dependency Inversion refers to the decoupling of software modules. This way, instead of high-level modules depending on low-level modules, both will depend on abstractions. In other words, instead of creating dependent objects inside the class with new keyword, we have to inject them by using constructor or setter.

1. Implement a binary search



1. Binary Search Tree:

Ans:

* While adding a new key, it always be added at leaf.
* We start searching key from root to till hit a leaf node.
* Once leaf node found, new node will be added as child.
* If the new node is bigger, it traverses right else left. Accordingly it will add.

1. Red Black tree:

Ans: It’s a self balanced binary tree.

* Every node has a color either red or black
* Root node is always black
* There are no two adjacent red nodes
* Every path from root to null, the no. of black nodes are equal.
* Operations include change of nodes color left shift or right shift.
* Nodes will be added as binary tree and to fulfill above properties, mentioned operations will be done.
* A new node will be added as a red node.
* If uncle is red, we do recoloring. If uncle is black, we do rotations or and recoloring.
* If both uncle and parent are in red, change both to black and grand parent to red.
* Make grand parent as new node and repeat the above step.
* If uncle is black and parent is red, then 4 cases based on where the new element is being added.
  + Left Left -> G right shift and parent grand parent color exchange.
  + Left Right -> parent left shift, G right shift, swap colors of G and new node.
  + Right Right -> G left shift, parent grand parent color exchange
  + Right Left -> parent right shift, G left shift, swap colors of G and new node.

1. Logic to find first repeated letter in "Hyderabad" (Ans is : 'd')

****

1. What is strategy design pattern?
2. How can you break the Singleton design pattern?

Ans:

* 1. To overcome this issue, override clone() method and throw an exception from clone method that is CloneNotSupportedException.
  2. We need to stop creating an object from outside. For this use private constructor. Call this from a public static getInstance() method which checks for existence of object already and create one by calling this private constructor if obj is already not there.
  3. We need to stop creation of obj from reflection API. For this add a condition in private constructor to check the existence of obj already, if exists throw a run time exception.
  4. We need to stop creation of obj simultaneously by more than one thread. For this declare the obj as volatile. Add synchronized block in public methods while creation of obj.
  5. We need to stop creation of obj from deserialization. For this, we need to override readResolve() method to call getInstance() method.



<https://codippa.com/how-to-break-singleton-in-java/>

<https://medium.com/@kevalpatel2106/how-to-make-the-perfect-singleton-de6b951dfdb0>

1. What is the difference between MVC 1 and MVC 2?

Ans: In MVC 1, controller and model both are JSP. While in MVC2 controller is servlet and model is java class. In MVC1 there is tight coupling between page and model as data access is usually done using Custom tag or through java bean call.  
In MVC2 architecture there is only one controller which receives all the request for the application and is responsible for taking appropriate action in response to each request.

1. What is SQL injection?

Ans: SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

eg 1:

txtUserId = getRequestString("UserId");

txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

If webuser provides the txtUserId as "105 OR 1=1" the sql looks like,

txtSQL = SELECT \* FROM Users WHERE UserId = 105 OR 1=1;

which returns all the user details present in the database.

eg 2:

SELECT \* FROM Users WHERE Name ="John Doe" AND Pass ="myPass" can modify to

SELECT \* FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

eg 3:

105; DROP TABLE Suppliers can convert our sql to

SELECT \* FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

1. What is session failover?

Ans: **System Failover:**

In this chapter, system failure refers to a hardware or process failure at the OpenSSO Enterprise server, at the Policy Agent, or at a load balancer. Hardware fails due to a mechanical problem or power outage. A web container application crashes causing OpenSSO Enterprise to become inaccessible. These are examples of system failure. Whenever possible, you should install redundant OpenSSO Enterprise servers, OpenSSO Policy Agents, and load balancers to serve as backups, or to fail over to, in the event of a system failure. This helps to ensure that no single point of failure exists in your deployment. Load balancers distribute the workload among OpenSSO Enterprise servers. If a Policy Agent fails, requests are redirected to another Policy Agent. If server hardware fails, requests are routed to other server hardware. Without system failover, a single hardware failure or process failure can cause OpenSSO Enterprise downtime.

**Session Failover:**

Session failover ensures that session data remains accessible to OpenSSO Enterprise servers and OpenSSO Enterprise Policy Agents. Service requests are routed to a failover server, the user's session continues uninterrupted, and no user data is lost. The OpenSSO Enterprise Session Service maintains authenticated session states and continues processing new client requests subsequent to the failure. In most cases, without session failover, after system failure and subsequent service recovery, the user would have to re-authenticate.

Session failover is critical when end-users' transactions involve financial data or other sensitive information that is difficult to recover when a system failure occurs. With session failover, when a system failover occurs, the user's transaction can proceed uninterrupted. Session failover is less important if end-users are, for example, reading but not writing data.

**OpenSSO Enterprise Sites:**

The most basic OpenSSO Enterprise site consists of two or more OpenSSO Enterprise servers and one or more load balancers. When you configure all the components in the site to work under a single site identifier, or name, all components in the site act as one unit. The load balancers in the site are associated with a site identifier. When a component such as a Policy Agent accesses a site, it communicates through the load balancer associated with that site, instead of directly accessing individual OpenSSO Enterprise servers in the site. All the client requests are passed through the load balancer to the OpenSSO Enterprise servers located behind a firewall. Individual OpenSSO Enterprise servers are never directly exposed to entities outside the firewall. The only client that can access the OpenSSO Enterprise servers is a load balancer.

1. What are the different types of data sources supported by the Websphere Admin Console?
2. Differentiate between a socket and a ServerSocket?
3. How GC works?

Ans: <https://stackify.com/what-is-java-garbage-collection/>

1. What is class loader?

Ans: <https://www.geeksforgeeks.org/classloader-in-java/>

The **Java ClassLoader** is a part of the [**Java Runtime Environment**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/) that dynamically loads Java classes into the [**Java Virtual Machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). The Java run time system does not need to know about files and file systems because of classloaders.

[Java classes](https://www.geeksforgeeks.org/classes-objects-java/) aren’t loaded into memory all at once, but when required by an application. At this point, the **Java ClassLoader** is called by the **JRE** and these ClassLoaders load classes into memory dynamically.

Not all classes are loaded by a single ClassLoader. Depending on the type of class and the path of class, the ClassLoader that loads that particular class is decided. To know the ClassLoader that loads a class the [***getClassLoader()***](https://www.geeksforgeeks.org/java-lang-class-class-java-set-1/) method is used. All classes are loaded based on their names and if any of these classes are not found then it returns a [**NoClassDefFoundError**](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/) or [**ClassNotFoundException**](https://www.geeksforgeeks.org/classnotfoundexception-vs-noclassdeffounderror-java/).

A Java Classloader is of **three types**:

1. **BootStrap ClassLoader:** A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java ClassLoader. Bootstrap ClassLoader loads classes from the location ***rt.jar***. Bootstrap ClassLoader doesn’t have any parent ClassLoaders. It is also called as the **Primodial ClassLoader**.
2. **Extension ClassLoader:** The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from ***jre/lib/ext*** directory or any other directory pointed by the system property ***java.ext.dirs***.
3. **System ClassLoader:** An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable ***CLASSPATH, -classpath or -cp command line option***. The Application ClassLoader is a child class of Extension ClassLoader.

**Note**: The ClassLoader Delegation Hierarchy Model always functions in the order Application ClassLoader->Extension ClassLoader->Bootstrap ClassLoader. The Bootstrap ClassLoader is always given the higher priority, next is Extension ClassLoader and then Application ClassLoader.

1. **Delegation Model**: The Java Virtual Machine and the Java ClassLoader use an algorithm called the **Delegation Hierarchy Algorithm** to Load the classes into the Java file.

The ClassLoader works based on a set of operations given by the delegation model. They are:

* + ClassLoader always follows the **Delegation Hierarchy Principle**.
  + Whenever JVM comes across a class, it checks whether that class is already loaded or not.
  + If the Class is already loaded in the method area then the JVM proceeds with execution.
  + If the class is not present in the method area then the JVM asks the Java ClassLoader Sub-System to load that particular class, then ClassLoader sub-system hands over the control to **Application ClassLoader**.
  + Application ClassLoader then delegates the request to Extension ClassLoader and the **Extension ClassLoader** in turn delegates the request to **Bootstrap ClassLoader**.
  + Bootstrap ClassLoader will search in the Bootstrap classpath(JDK/JRE/LIB). If the class is available then it is loaded, if not the request is delegated to Extension ClassLoader.
  + Extension ClassLoader searches for the class in the Extension Classpath(JDK/JRE/LIB/EXT). If the class is available then it is loaded, if not the request is delegated to the Application ClassLoader.
  + Application ClassLoader searches for the class in the Application Classpath. If the class is available then it is loaded, if not then a **ClassNotFoundException** exception is generated.

1. **Visibility Principle**: The **Visibility Principle** states that a class loaded by a parent ClassLoader is visible to the child ClassLoaders but a class loaded by a child ClassLoader is not visible to the parent ClassLoaders. Suppose a class GEEKS.class has been loaded by the Extension ClassLoader, then that class is only visible to the Extension ClassLoader and Application ClassLoader but not to the Bootstrap ClassLoader. If that class is again tried to load using Bootstrap ClassLoader it gives an exception ***java.lang.ClassNotFoundException***.
2. **Uniqueness Property**: The **Uniquesness Property** ensures that the classes are unique and there is no repetition of classes. This also ensures that the classes loaded by parent classloaders are not loaded by the child classloaders. If the parent class loader isn’t able to find the class, only then the current instance would attempt to do so itself.

**Methods of Java.lang.ClassLoader**

After the JVM requests for the class, a few steps are to be followed in order to load a class. The Classes are loaded as per the delegation model but there are a few important Methods or Functions that play a vital role in loading a Class.

1. **loadClass(String name, boolean resolve)**: This method is used to load the classes which are referenced by the JVM. It takes the name of the class as a parameter. This is of type loadClass(String, boolean).
2. **defineClass()**: The defineClass() method is a *final* method and cannot be overriden. This method is used to define a array of bytes as an instance of class. If the class is invalid then it throws **ClassFormatError**.
3. **findClass(String name)**: This method is used to find a specified class. This method only finds but doesn’t load the class.
4. **findLoadedClass(String name)**: This method is used to verify whether the Class referenced by the JVM was previously loaded or not.
5. **Class.forName(String name, boolean initialize, ClassLoader loader)**: This method is used to load the class as well as initialize the class. This method also gives the option to choose any one of the ClassLoaders. If the ClassLoader parameter is NULL then Bootstrap ClassLoader is used.
6. All sorting and searching programs?
7. How to handle cyclic dependency in springs?

Ans: Except for constructor based dependency for all other injections, spring will take care of this. In case of constructor based injection, you can use @Lazy to one of those beans so that, it creates a proxy of that and injects into other bean. This proxy will be created fully when it was called first. Or you can go for setter injection so that spring will not inject until they are needed.

1. Sort an array of 0 and 1.

Ans: You can count number of 1’s or 0’s and based on that create a new array having those number of 0’s first followed by 1’s. complexity will be O(n)

Else you can have two indexes and traverse as below.

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

**byte**[] b = **new** **byte**[] {1,1,0,0,0,1,0,1,0,0,0};

**int** i = 0;

**int** j = b.length-1;

**while**(i<j) {

**if**(b[i] == 1) {

**if**(b[j] == 0) {

b[i]=0;

b[j]=1;

i++;

j--;

}

**else** {

j--;

}

}

**else** {

i++;

}

}

**for**(**byte** b1:b)

System.***out***.println(b1 +" ");

}

1. LinkedList internal working.

Ans: It is an implementation of double linked list. It keeps the first and last node transient vars and based on the method called like, addFirst, addLast, add, remove, removeFirst, removeLast, getFirst, getLast, get… it traverse the nodes. Also changes the next, first, last references accordingly. It also implements List, Serializable, Cloneable interfaces and extends AbstactSequentialList abstract class.

1. How to get second/third max salary from employee table?

Ans: **SELECT** **MAX**(Salary) **From** Employee **WHERE** Salary **<** ( **SELECT** **Max**(Salary) **FROM** Employee);

select unique(salary) from employee

ORDER BY salary DESC

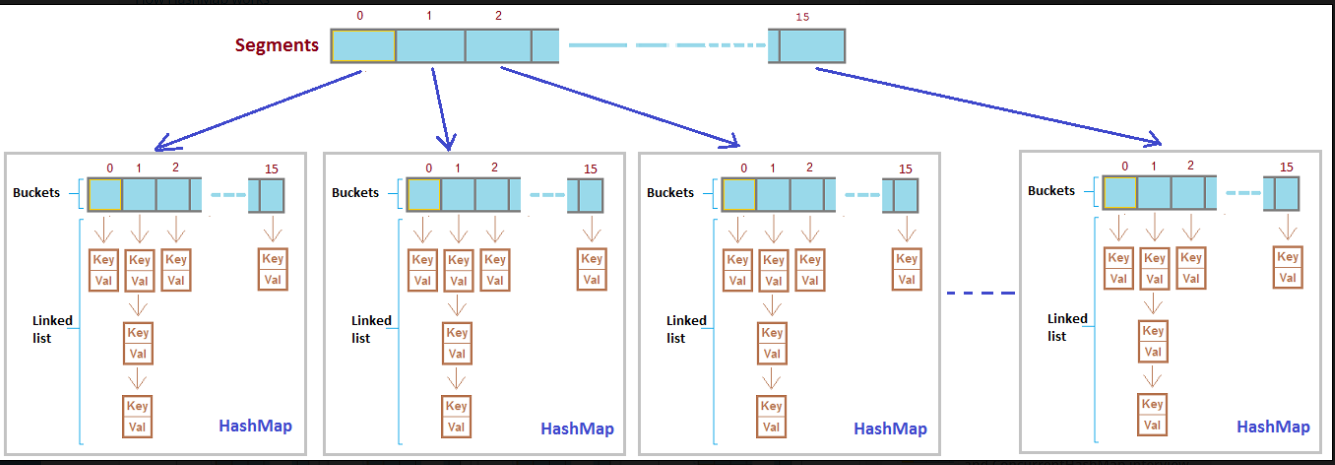
OFFSET 2 ROWS

FETCH NEXT 1 ROWS ONLY; -- 3rd highest salary

1. **What is concurrent hashmap and when u use it?**

* **Ans:** Prerequisites:Need of ConcurrentMap  
  ConcurrentHashMap ConcurrentHashMap class is introduced in JDK 1.5, which implements ConcurrentMap as well as Serializable interface also. ConcureentHashMap is enhancement of HashMap as we know that while dealing with Threads in our application HashMap is not a good choice because performance wise HashMap is not upto the mark.
* Key points of ConcurrentHashMap:
* The underlined data structure for ConcurrentHashMap is Hashtable.
* ConcurrentHashMap class is thread-safe i.e. multiple thread can operate on a single object without any complications.
* At a time any number of threads are applicable for read operation without locking the ConcurrentHashMap object which is not there in HashMap.
* In ConcurrentHashMap, the Object is divided into number of segments according to the concurrency level.
* Default concurrency-level of ConcurrentHashMap is 16.
* In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updation in object, thread must lock the particular segment in which thread want to operate. This type of locking mechanism is known as Segment locking or bucket locking. Hence at a time 16 updation operations can be performed by threads.
* null insertion is not possible in ConcurrentHashMap as key or value.
  1. **How concurrenthashmap contains segments?**

Ans: ConcurrentHashMap added one Array on top of it and each index of this additional array represents complete HashMap. Additional array is called Segment in ConcurrentHashMap.



* 1. **How concurrenthashmap decides in which segment data to be stored?**

**Ans:** Once we get the hashcode value with hashcode() method, this value will be sent to segments() method to identify the segment number.

The only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using Collections.synchronizedMap() method.  
  
  
Read more: <https://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html#ixzz5o6spdOlX>

* 1. **How hashset use hashmap where as hashset having only value but map having key, value pair? What is that value passed to map in case of hashset?**

**Ans:** A dummy Object class object(**PRESENT**) value to associate with an Object in the backing up for Hashmap value.

* 1. **What is the special advantage of in built in FunctionalInterface Predicate instead of if clause?**
  2. **How hashset identifies a duplicate value?**

**Ans:** Hashmap works on hashing technology. **hashCode**() method calls on key object and returns hashcode value. **Hash**() function of map returns the hash value and which is sent a input for **indexFor**() method which identifies the bucket where this value to be stored which actually the index of internal array(hashmap is an array of linked lists which stored Entry obj).

Hashmap keys are immutable because while retrieving the data, the hashcode of the same key should return same hashcode value. That is the reason, String and other wrapper classes are the best to use as keys. Also for these classes equals method and hashcode method are already overridden.

When two keys got same hashcode, this is called collision in hashmap and these will be added as linked list on the same bucket.

HashMap simply overrides the value if add an Entry with same key if and only if equals method could identify it.

The main point to notice is that put (key,value) will return  
  
1.  null , if key is unique and added to the map  
2.  Old Value of the key , if key is duplicate

public boolean add(E e) {  
            return map.put(e, PRESENT)==null;  
       }

* 1. **What is volatile?**

**Ans:** Using volatile is yet another way (like synchronized, atomic wrapper) of making class thread safe.

Suppose that two threads are working on **SharedObj**. If two threads run on different processors each thread may have its own local copy of **sharedVariable**. If one thread modifies its value the change might not reflect in the original one in the main memory instantly. This depends on the [write policy](https://en.wikipedia.org/wiki/CPU_cache#Write_policies) of cache. Now the other thread is not aware of the modified value which leads to data inconsistency.

**volatile vs synchronized:**  
Before we move on let’s take a look at two important features of locks and synchronization.

1. **Mutual Exclusion:** It means that only one thread or process can execute a block of code (critical section) at a time.
2. **Visibility**: It means that changes made by one thread to shared data are visible to other threads.

Java’s synchronized keyword guarantees both mutual exclusion and visibility.

Volatile variables have the visibility features of synchronized but not the atomicity features.

The values of volatile variable will never be cached and all writes and reads will be done to and from the main memory.

* 1. **What is serialization and what is problem if we don’t give unique serialVersionUID?**

**Ans:** Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the java.io.Serializable interface.

The ObjectOutputStream class contains writeObject() method for serializing an Object.

The ObjectInputStream class contains readObject() method for deserializing an object.

Only the objects of those classes can be serialized which are implementing java.io.Serializable interface.

**Points to remember**

* If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.
* Only non-static data members are saved via Serialization process.
* Static data members and transient data members are not saved via Serialization process. So, if you don’t want to save value of a non-static data member then make it transient.
* Constructor of object is never called when an object is deserialized.
* Associated objects must be implementing Serializable interface.
* serialVersionUID is needed to remember versions of the class. It should be same while serializing and deserializing. It is a good programming practice to provide this value rather than JVM assigning one(generally it is hash). It is not necessary for two classes to have unique values but two versions of same class should have unique version id.
  1. **What is transient key word?**

**transient** is a Java keyword which marks a member variable not to be serialized when it is persisted to streams of bytes. When an object is transferred through the network, the object needs to be 'serialized'. Serialization converts the object state to serial bytes. Those bytes are sent over the network and the object is recreated from those bytes. Member variables marked by the java transient keyword are not transferred; they are lost intentionally.

**transient and static**: Since static fields are not part of state of the object, there is no use/impact of using transient keyword with static variables. However there is no compilation error.

**transient and final**: final variables are directly serialized by their values, so there is no use/impact of declaring final variable as transient. There is no compile-time error though.

To pass sensitive info over the n/w, use encryption and decryption.

* 1. **When we used to get object can’t be serialized exception?**

Ans: The NotSerializableException is thrown when attempting to serialize or deserialize an object that does not implement the java.io.Serializable interface. In case if we use any reference object inside an object to be serialize, that should be either implement Serializable or it should be declared as transient.

The exception hierarchy is as shown below.

java.lang.Object -> java.lang.Throwable -> java.lang.Error -> java.io.IOException -> java.io.ObjectStreamException -> NotSerializableException

* 1. **What are types of injections in spring? When you use what?**
  2. **Difference b/n look up and injection?**

**Ans:** Dependency lookup is when the object itself is trying to find a dependency, such as:

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("/application-context.xml");

MyBean bean = applicationContext.getBean("myBean")

Here, the class itself is initializing the ApplicationContext through an XML, and it is searching in the context for the bean called myBean in the ApplicationContext

The Dependency injection is when a property is automatically binded when an istance is initialized. For example:

in the application-context.xml, we have one line which initialize the bean and another to initialize the object of, let's say, MyClass:

<bean id="myBean" class="org.mypackage.MyBean"/>

<bean id="myClass" class="org.mypackage.MyClass"/>

Then in in MyClass, you have something like:

@Component

public class MyClass{

@AutoWire

MyBean myBean;

}

* 1. **What is exactly IOC in spring?**

**Ans:** All the responsibilities of a class apart from its core functionality should be delegated to some other class or something else so that our class concentrates on just its core responsibility.

**Eg.**, if class A depends on B, just use object of B for the purpose but don’t take the headache of creation and maintenance of object B. Factory pattern helps us here by giving the responsibility of object creation to another class based on an input string beanName here in case of springs. BeanFactory or ApplicationFactory are based on Factory pattern, creates the objects and provide back to the application based on beanName.

* 1. **byName, byType, constructor autowiring?**
  2. **How you inject a prototype bean into a singleton bean, what will be the behavior of such things?**

**Ans:**

* 1. by implementing ApplicationContextAware interface, we will implement its method( to get ApplicationContext. Using this context we will get the prototype obj inside singleton obj.
  2. By using <lookup-method> injection. Config file will be like below. This lookup method dynamically creates the object every time it calls instead of injecting the created object by the container. This is called as lookup method injection where as normal injection is dependency injection.

abstract RequestHandler getRequestHandler();

<bean id="requestManager" class="org.netjs.prog.RequestManager">

<lookup-method name="getRequestHandler" bean="requestHandler"/>

</bean>

<bean id="requestHandler" class="org.netjs.prog.RequestHandler" scope="prototype" />

* 1. Scoped Proxy: <aop:scoped-proxy/>

When a singletonBean instance invokes a method on the dependency-injected sessionBean object, it actually is invoking a method on the proxy. The proxy then fetches the real sessionBean object from (in this case) the HTTP Session, and delegates the method invocation onto the retrieved real sessionBean object.

* 1. **How you manage transaction management in spring, spring jpa/spring data?**

**Ans: Java Transaction API** will manage the transaction management in case if we go for ApplicationManagedEntityManager and inbuilt **JPA’s Entity transaction management** in case of ContainerManagedEntityManager.

* 1. **What is Optional class in java 8?**
  2. **Difference b/n abstract class and interface when comes to java8 as java8 interfaces can have default methods?**

difference is that an interface cannot have a constructor even in Java 8 but you may remember that [abstract class can also have a constructor](http://www.java67.com/2013/02/can-abstract-class-have-constructor-in-java.html) in Java. All methods of an interface were abstract but since Java 8 you can define non-abstract methods in the form of default and static methods inside interface in Java.  
  
Abstract class is commonly used to define a base class for a type hierarchy with default implementation, which is applicable to all child classes, but interface is a purest form of API. It defines what it will do but not how it will do.

Interface is to tell what a class contains, abstract class is to implement all those contained features which are common to all childs.

Now if we say we can’t create an instance of an abstract class then why do Java adds a constructor in the abstract class. One of the reasons which make sense is when any class extends an abstract class, the constructor of sub class will invoke the constructor of super class either implicitly or explicitly. This chaining of constructors is one of the reasons abstract class can have constructors in Java.

Interfaces support multiple inheritance but abstract classes can’t.

* 1. **What is the use of constructor in abstract class?**

**Ans:** A constructor in Java doesn't actually "build" the object, it is used to initialize fields.

Imagine that your abstract class has fields x and y, and that you always want them to be initialized in a certain way, no matter what actual concrete subclass is eventually created. So you create a constructor and initialize these fields.

Now, if you have two different subclasses of your abstract class, when you instantiate them their constructors will be called, and then the parent constructor will be called and the fields will be initialized.

If you don't do anything, the default constructor of the parent will be called. However, you can use the super keyword to invoke specific constructor on the parent class.

* 1. **How you can check the duplicity while inserting into ArrayList?**

**Ans:** using java 8 streams.

list.stream().distinct().collect(Collectors.toList());

* 1. **How the user inserted data filled in spring beans?**

**Ans:**

* 1. **How to use @Autowired for constructor dependency?**

**Ans:**

* 1. **How the UI form data will be injected into java class in Controller class?**

**Ans:**

* 1. **By default spring beans are singleton but how different users will get data differently?**

Ans: It is ideal to consider that Singleton scope is suitable for defining the stateless beans and Prototype scope is suitable for stateful beans.

* 1. **By default spring beans are singleton, is the singleton at container level or application level?**

**Ans:** Java singleton class is per classloader and Spring’s singleton is per application context(per container and per bean).

* 1. **How transaction management happens in spring microservices as project scatters in different MSs?**

Ans:

* 1. **In how many ways singleton can be implemented?**

**Ans:**

* 1. **How u add ur other project as a dependent in parent project pom.xml?**

**Ans:** <project>

...

<dependencies>

<dependency>

<groupId>yourgroup</groupId>

<artifactId>myejbproject</artifactId>

<version>2.0</version>

<scope>system</scope>

<systemPath>path/to/myejbproject.jar</systemPath>

</dependency>

</dependencies>

...

* 1. **Sort an array of zeros and ones in O(n) iterations?**

**Ans:**

* 1. **How Oracle dependencies will be added in pom.xml in MSs?**

**Ans:** 1. Download the oracle jar file and add the below dependency in pom.xml.

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0</version>

</dependency>`

1. Now install this jar file to maven repository

mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dpackaging=jar -Dversion=11.2.0.4.0 -Dfile=ojdbc6.jar -DgeneratePom=true

* 1. **Mention the areas where singleton has been implemented?**

**Ans:** Singleton pattern is used for logging, drivers objects, caching and thread pool.

Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.

Singleton design pattern is used in core java classes also, for example java.lang.Runtime, java.awt.Desktop.

* 1. **Spring version 4 features?**

**Ans:**

* 1. **Annotation in spring boot to stop auto configurations for a jar?**

**Ans:**@EnableAutoConfiguration/@SpringBootApplication(exclude=<class names not to be auto configured>

Otherwise you can use @ComponentScan and @Configuration instead of @SpringBootApplication.

* 1. **Known problems in Micro services?**

**Ans:**

* 1. **What is exactly inversed in IOC of spring?**

**Ans:**

* 1. **What is intermediate and end operations in Streams? Can an end operation become an intermediary?**
  2. **Can we apply Streams directly on collections without converting them to Streams?**
  3. [**What is difference between Lambda Expression and Anonymous class?**](http://www.java2novice.com/java_interview_questions/lamda-expression-anonymous-class/)
  4. [**What is HTTP basic authentication?**](http://www.java2novice.com/java_interview_questions/http-basic-authentication/)

**Ans:** First HTTP client makes a request to the web server. Request method doesn’t has to be GET it can be any method. If web server sees that the requested resource need authentication to access then it sends backs 401 Unauthorized status code along with WWW-Authenticate header. And then client displays a dialog box to take username and password as input. Once the credentials has been enter the client sends it using the Authorization header. If the credentials are correct then server responds with 200 status code and Authentication-Info header.

* 1. [**What is functional interface in java?**](http://www.java2novice.com/java_interview_questions/functional-interface/)
  2. [**What is the difference between HTTP methods GET and POST?**](http://www.java2novice.com/java_interview_questions/http-get-post/)
  3. **What are non-idempotent methods in rest?**

**Ans:** An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. Idempotency is important in building a fault-tolerant API. Suppose a client wants to update a resource through POST. Since POST is not a idempotent method, calling it multiple times can result in wrong updates. What would happen if you sent out the POST request to the server, but you get a timeout. Is the resource actually updated? Does the timeout happened during sending the request to the server, or the response to the client? Can we safely retry again, or do we need to figure out first what has happened with the resource? By using idempotent methods, we do not have to answer this question, but we can safely resend the request until we actually get a response back from the server.

POST and PATCH are non-idempotent methods where as PUT, GET, DELETE, OPTIONS, HEAD are idempotent. Out of which GET, OPTIONS, HEAD are the only safe methods.

* 1. [**What is difference between CountDownLatch and CyclicBarrier in Java?**](http://www.java2novice.com/java_interview_questions/countdownlatch-cyclicbarrier/)

**Ans:** As stated in the definitions, CyclicBarrier allows a number of threads to wait on each other, whereas CountDownLatch allows one or more threads to wait for a number of tasks to complete. In short, CyclicBarrier maintains a count of threads whereas CountDownLatch maintains a count of tasks.

* 1. [**Can Enum extend any class in Java?**](http://www.java2novice.com/java_interview_questions/can-enum-extend/)

**Ans:** No, because Enum already extends a base abstract class java.lang.Enum

* 1. [**Can Enum implements any interface in Java?**](http://www.java2novice.com/java_interview_questions/can-enum-implements/) **Ans: Yes**
  2. [**Can we have constructor in abstract class?**](http://www.java2novice.com/java_interview_questions/abstract-class-constructor/)
  3. [**What is MVC pattern?**](http://www.java2novice.com/java_interview_questions/mvc-pattern/)
  4. [**What is ActionServlet in struts?**](http://www.java2novice.com/java_interview_questions/struts-action-servlet/)
  5. [**What is the difference between servlet & Filter?**](http://www.java2novice.com/java_interview_questions/servlet-filter/)
  6. [**What is ActionMapping in struts?**](http://www.java2novice.com/java_interview_questions/struts-action-mapping/)
  7. [**What is the difference between application server and web server?**](http://www.java2novice.com/java_interview_questions/web-application-server/)
  8. [**What is the difference between JPA and Hibernate?**](http://www.java2novice.com/java_interview_questions/jpa-and-hibernate/)
  9. [**What is difference between the Value Object and JDO?**](http://www.java2novice.com/java_interview_questions/jdo-and-value-object/)
  10. [**How Struts control data flow?**](http://www.java2novice.com/java_interview_questions/struts-control-flow/)
  11. [**What is Spring?**](http://www.java2novice.com/java_interview_questions/what-is-spring/)
  12. [**What is Dependency Injection?**](http://www.java2novice.com/java_interview_questions/dependency-injection/)
  13. [**What are the different types of dependency injections in spring?**](http://www.java2novice.com/java_interview_questions/spring-dependency-injection-types/)
  14. [**What is BeanFactory in Spring?**](http://www.java2novice.com/java_interview_questions/spring-bean-factory/)
  15. [**What is difference between BeanFactory and ApplicationContext in spring?**](http://www.java2novice.com/java_interview_questions/beanfactory-applicationcontext/)
  16. [**How to make a bean as singleton in spring?**](http://www.java2novice.com/java_interview_questions/spring-singleton-bean/)
  17. [**What is IOC or inversion of control?**](http://www.java2novice.com/java_interview_questions/inversion-of-control/)
  18. [**What are different types of spring auto-wiring modes?**](http://www.java2novice.com/java_interview_questions/spring-autowire-modes/)
  19. [**What are the limitations and disadvantages of spring autowiring?**](http://www.java2novice.com/java_interview_questions/spring-autowire-limitations/)
  20. [**Is the spring singleton bean thread safe?**](http://www.java2novice.com/java_interview_questions/spring-singleton-thread-safe/)
  21. [**Why ConcurrentHashMap is faster than Hashtable in Java?**](http://www.java2novice.com/java_interview_questions/concurrenthashmap-faster-than-hashtable/)
  22. [**What is the difference between ConcurrentHashMap and Hashtable in Java?**](http://www.java2novice.com/java_interview_questions/difference-between-concurrenthashmap-and-hashtable/)
  23. [**Difference between ConcurrentHashMap and Collections.synchronizedMap(Map)?**](http://www.java2novice.com/java_interview_questions/concurrenthashmap-and-synchronizedmap/)
  24. [**What is the difference between ORM, JPA and Hibernate?**](http://www.java2novice.com/java_interview_questions/orm-jpa-hibernate/)
  25. [**What is stream pipelining in Java 8?**](http://www.java2novice.com/java_interview_questions/java-8-stream-pipelining/)
  26. [**What is interface default method in java 8?**](http://www.java2novice.com/java_interview_questions/java-8-interface-default-method/)
  27. [**Java-8: Interface with default methods vs Abstract class.**](http://www.java2novice.com/java_interview_questions/default-method-vs-abstract-class/)
  28. [**What is @SpringBootApplication annotation in Spring boot project?**](http://www.java2novice.com/java_interview_questions/springbootapplication-annotation/)
  29. [**What Embedded Containers Does Spring Boot Support?**](http://www.java2novice.com/java_interview_questions/spring-boot-supported-embedded-containers/)
  30. [**What are the advantages and disadvantages of Spring Boot?**](http://www.java2novice.com/java_interview_questions/spring-boot-pros-and-cons/)
  31. [**What is Spring Boot Actuator?**](http://www.java2novice.com/java_interview_questions/spring-boot-actuator/)
  32. **What is Spring Boot Initializr?**
  33. **How does Maven resolve version conflicts of dependencies?**
  34. **How to reload Spring Boot Application without restarting server?**
  35. **What is the difference between Spring Boot and the Spring framework?**
  36. **What are the key components of Spring Boot framework?**
  37. **What are the different types of bean scope in Spring framework?**
  38. **What are the standard Spring build-in events?**
  39. **What is Spring IoC container?**
  40. **Differences between BeanFactory and the ApplicationContext in Spring framework.**
  41. **Difference between constructor injection and setter injection in Spring.**
  42. **How Java-8 Streams differ from collections**
  43. **What are the various ways to obtain Streams in Java-8?**
  44. **List Java-8 Streams intermediate operations.**
  45. **List Java-8 Streams terminal operations.**
  46. **Can we reuse Java-8 Streams?**
  47. **Difference between map and flatMap methods in Java 8**
  48. **What is the difference between Closure and Lambda in Java 8?**
  49. **Does Java 8 Lambda supports recursive call?**
  50. **Can Java 8 default methods override equals, hashCode and toString?**
  51. **Hibernate Eager vs Lazy Fetch Type**
  52. **What is POJO?**
  53. **What is HQL (Hibernate Query Language)?**

<http://www.java2novice.com/java_interview_questions/spring-dependency-injection-types/>

**ANGULAR**

1. Class:
2. Constructor: only one can be created
3. Access modifier: public, private, protected
4. Public -> can be accessed outside the class and it is default, private -> can’t be used outside the class, protected -> can be used in inherited classes too.
5. Static variable is specific to the class but not to object.
6. Modularity: set of functionality
7. Import or export of a class or method for reusability
8. Type script course: <https://lex.infosysapps.com/viewer/lex_9436233116512678000/lex_auth_012712954616782848927>
9. Angular course: <https://lex.infosysapps.com/viewer/lex_20858515543254600000/lex_9393820929135647000>
10. https://lex.infosysapps.com/viewer/lex\_20858515543254600000/lex\_19957124143596520000
11. **npm install -g @angular/cli**
12. Angular was developed by google
13. Angular 1 to 7. From Angular 2, type script support introduced so we can design mobile apps.
14. Angular play ground

<https://lex.infosysapps.com/viewer/lex_auth_01257271100325068833>

<https://lex.infosysapps.com/viewer/lex_20858515543254600000/lex_30987236123954876000>

Features of Angular:

1. SPA
2. Modularity
3. Typescript support
4. Cross browser support
5. Web component support
6. Mobile app development
7. Easy to learn
8. Ide support
9. ECMA script 6
10. Performance

Ng new <MyApp> -> to create a new app.

Ng generate component <myComponent> -> to add a new component

ng serve - - open -> to start and open a project

**Template:**

1. Inline -> direct html code inside back ticks
2. External -> templateUrl -> include an external html file.

**Elements in template:**

1. Html, interpolation,{{}}, Template expressions – text inside interpolation -> {{expression}}, template statements (onclick events or etc.)
2. () -> one way databinding, html to TS

**Directives:**

1. Structural directives
2. Attribute directives

**ANGULAR**

1. Why type script instead of type script?

Ans: JS is interpreted not compiled, so we will get one by one and need fix one by one. Minimum obj orientation. JS is not data type specific, we will get errors at run time in case data type not matches.

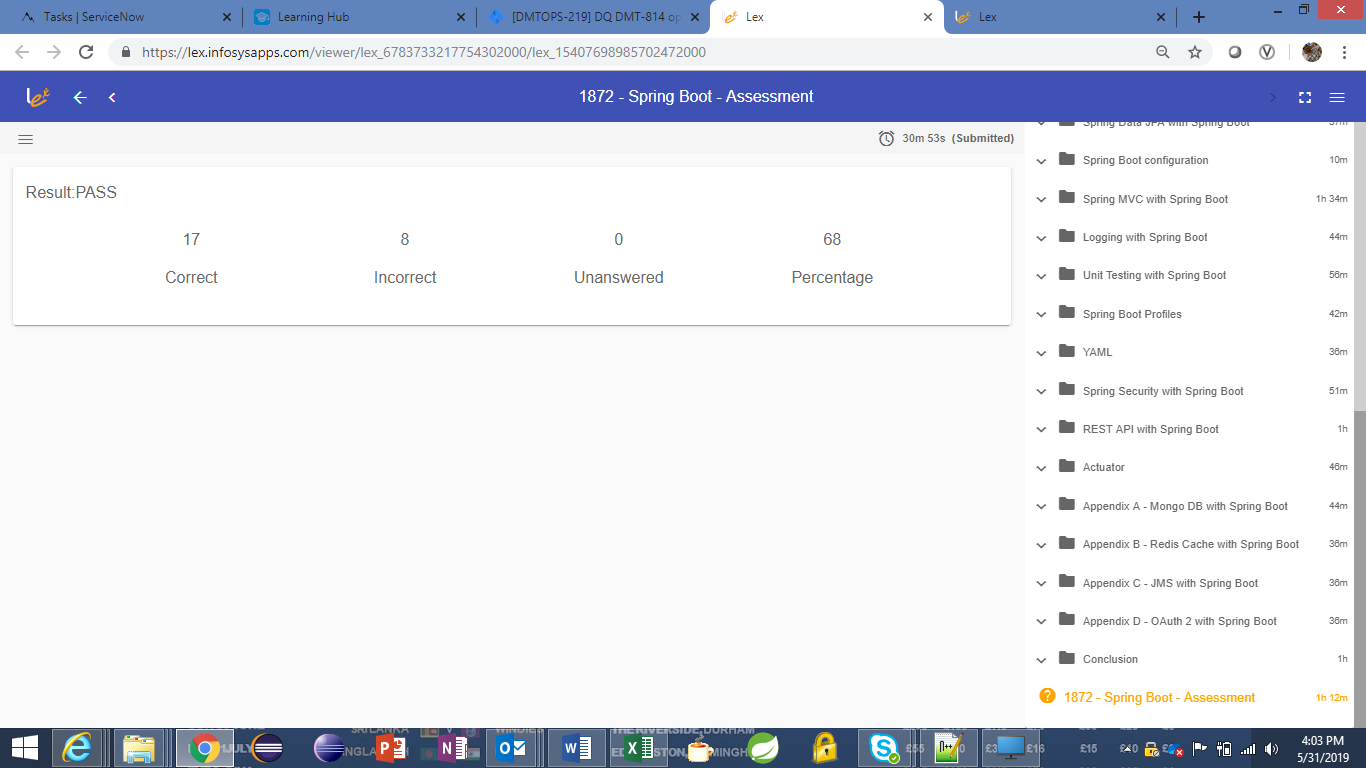
TS is extension of ECMA script. Any valid JS is also a TS. Finally TS need to transpiled in JS as browser doesn’t understand it.

1. “let” keyword instead of “var” keyword.
2. <https://lex.infosysapps.com/viewer/lex_20858515543254600000>

**Creation of angular application:**

1. Install node js first
2. **npm install -g @angular/cli 🡪 install angular cli to provide basic structure of angular application**
3. ng new my-app 🡪 creates a new application
4. ng server - - open - - port 3000->
5. ng build
6. ng generate
7. Install Microsoft visual studio code from software center

Spring boot assessment:



**Java 8**

**Default methods:**

**Default methods**(those have **default** keyword) are quite useful to include new features in an interface without altering the implementations that exist for the interface. The default methods of interfaces have definitions that does not require to be redefined, it helps to manage the code.

Default methods also helps in avoiding utility classes, for example all Collections class methods provides default methods in the Collection interface.

**Design Patterns**

**Singleton:**

****

**Factory pattern:** Instead of creating objects everywhere, this pattern restricts the obj creation to a specific place called Factory based on an input type. This is useful for abstracting the actual things from user. User doesn’t know to which class we are creating the objects.

If input type is “add”, we will create obj for Add, if type is “multiply” we will create object of Multiply. This will be done in CalculateFactory class based on reflections concept. Uses instanceof method



**Template method pattern:** Sub classes decide how to implement steps in an algorithm. If we have common functionalities, we need to implement those in Abstract class and the unique methods should kept as abstract methods and implementation will be done in child classes.

**Front Controller pattern:**

**Strategy pattern:**

* Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.
* Capture the abstraction in an interface, bury implementation details in derived classes.
* A shoe contains outsole, heel etc. Outsole and heel applicable for all the shoes but it very from different sets of shoes.
* Outsole made up of rubber or synthetic etc. so it can be a strategy for us. RubberOutsole and SyntheticOutsole.
* Heel can be BoostHeel or UltraBoostHeel.
* So instead of implementing Outsole and Heel in a single class, we can divide these into 4 or more strategies so that now at runtime 4 different shoes can be implemented by combining these strategies.

**Adapter/Wrapper pattern:**

Making two incompatible interfaces compatible. Façade is to hide complexity of several classes. Proxy is to stay in between the actual class for security/caching reasons. Decorator is to add behavior to classes without changing the class.

Adapter converts one kind of interface to other to be compatible with the client requirement. Takes one kind of input and give other kind of output.

Client.java:

ITarget target = new Adapter(new Adaptee());

target.getRequest();

interface ITarget.java:

Public void request();

Adapter.java implements ITarget:

Adaptee a;

Public Adapter(Adaptee a){this.a=a;}

Publc void request(){this.specialRequest();}

Public void request(){};

Adaptee.java:

Public void specificRequest(){};

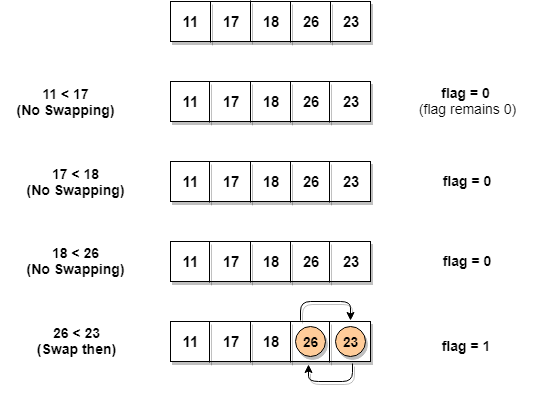
**Façade pattern:** sometimes, our client need to talk to a class A but A having the dependency on B and B on C, C again on A and D etc. so for client it is too complex to create all these dependencies. So Façade will be outer layer of this complexity which creates all these dependencies and client just interacts with this Façade.

Eg., Connection pooling. All internal complexity will be abstracted and provides the datasource for us.

**Sorting Algorithms**

**Bubble sort:** It is the simplest sort which performs sorting by moving the largest element to the highest index.





**Insertion Sort:** As the name suggests, insertion sort inserts each element of the array to its proper place. One element compares with all its previous elements.



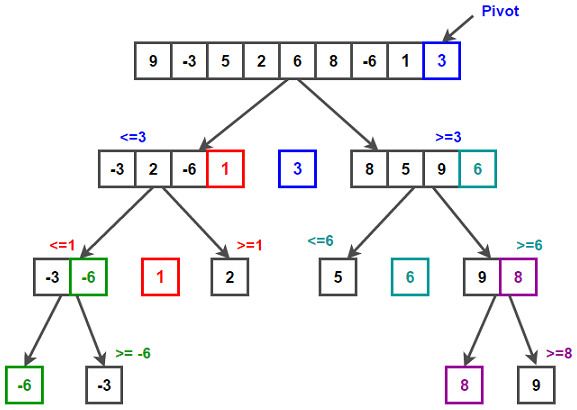


**Merge Sort:** Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.



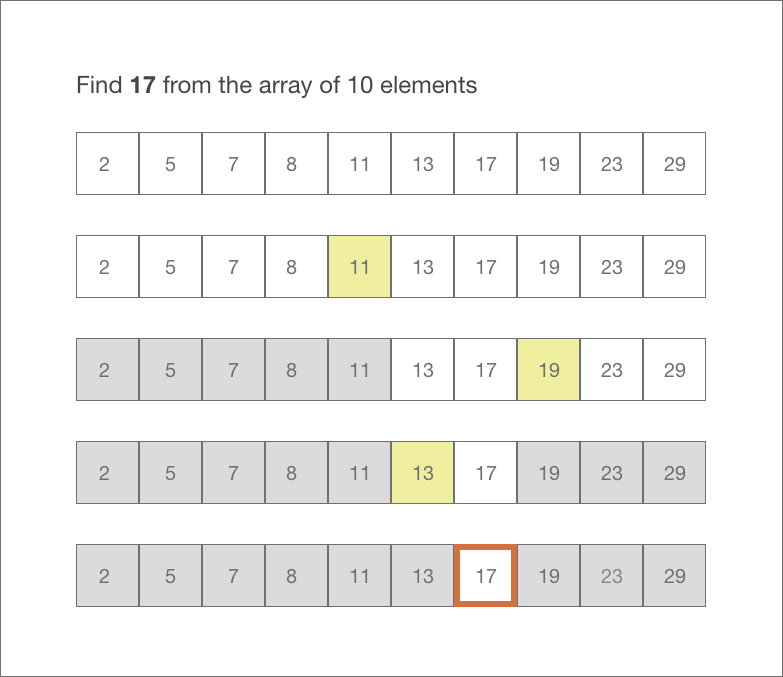


**Quick Sort:** Quick sort is based on the divide-and-conquer approach based on the idea of choosing one element as a pivot element and partitioning the array around it such that: Left side of pivot contains all the elements that are less than the pivot element Right side contains all elements greater than the pivot.



**Searching Algorithms**

**Binary Search:**



**Spring security:**

1. Add security dependency in your application.
2. Create a Config folder in java classes folder.
3. Create a java class with SecurityConfiguration name and extend with WebSecurityConfigurerAdapter. This class adapter will inject all spring security related injections.
4. Annotate this class with @Configuration and @EnableWebSecurity
5. Override a method protected void configure(HttpSecurity httpSecurty) method as below.

Protected void configure(HttpSecurity httpSecurity) {

httpsSecurity.authorizeRequests()

//.anyMatchers(“\*\*/rest/\*”)

.anyRequest()

.permitAll()

.and().httpBasic();

httpSecurity.csrf().disable();

}

**Spring security:**

1. Create permission, role, user, assign\_role\_to\_user tables in the database.
2. User table contains, user\_name, email, password in bCrypt form.
3. Create a spring boot security project with the dependencies - spring web, spring jdbc, mysql, cloud oAuth2 dependencies.
4. In application.yml file provide database details.
5. Create public and private keys using Open SSL and provide these details as well in cofig file. Also provide username and pwd.
6. Private key shouldn’t be shared with anyone but public key should be shared among the microservices which use this security feature.
7. In application.yml provide clientid and clientSecret details which will be useful to fetch the token.
8. Also provide accesstoken\_validity and refreshtoken\_validity. Refresh token is used to send to server and get a new token when our access token got expired.
9. We will wrap these access tokens in Jwt token and send to authorization server, so that authorization server understands that the request came from a trusted client.
10. Write @EnableAuthorizationServer in main spring boot class which makes our application to act as a authorization server.
11. Now create a Model class for UserEntity (id, email, name, pwd, List<GrantedAuthority>).
12. Create a model class CustomUser which extends spring security core User class
13. Write a constructor with parameter as above UserEntity and call super constructor by calling all getter methods of userEntyty obj.
14. Also add other custom properties like id and name.
15. Create DAO service interface with method getUserDetails and write a class to implement this by calling DB to get list of authorities.
16. Now all the common set up has been completed to fetch data from ADFS or DB. Now it is the time to write security configuration files.
17. Create a config package and create a class CustomTokenEnhancer which extends JwtAccessTokenConverter which by default contains the user details like user name, list of GrantedAuthorities. We can add other custom needed attributes in this class.
18. We need to override a method enhance() method where we will take all extra info like id, name etc and set as additional info.
19. Now in security server project create a configuration class SecurityConfiguration which should extend WebSecurityConfigureAdapter class. This class should be annotated with @EnableWebSecurity, @Configuration, @EnableGlobalMethodSecurity(prePostEnabled = true)
20. Add a bean PasswordEncoder to convert user given plain password to encrypted password in the database.
21. Override a method configure.
22. Add one more bean AuthenticationManager and override authenticationManagerBean() method.
23. Now create a service package in authentication server and create a class CustomUserDetailsService and annotate with @Service. It should implement inbuilt spring security UserDetailsService class.
24. Implement loadUserbyUserName() method.
25. Autowire OAuthDAOService class and getuser details from this in the above method.
26. Create OAuth2Service class which should extend AuthorizationServerConfigurerAdapter.
27. Get all client id, client secret and other config elements from config file
28. Autowire PasswordEncripter, AuthenticationManager classes.
29. We need to override 3 configure methods for permitAll, access token details, end points authentication manager/tokenstore.
30. **Creation of rest services to use above authentication server**
31. Create a project with dependencies, spring web and oauth2 could dependencies.
32. Open main class and annotate with @EnableResourceServer
33. Create an application.yml file, add public key.
34. Create AccessTokenMapper with access\_token, id, username, name properties.
35. Create a SecurityConfiguration class file which should extend WebSecurityConfigurerAdapter class and add @EnableWebSecurity, @Configuration, @EnableGlobalMethodSecurity(prePostEnabled = true) annotatios.
36. Override configure() method.
37. We need to extract the Jwt token data for that we need to write a class.
38. Write a class JwtConverter which should extend DefaultAccessTokenConverter and implement JwtAccessTokenConverterConfigurer.
39. Override configure() method.
40. Override extractAuthentication() method.
41. Create rest controllers now.
42. On top of every resource annotate with @PreAuthorize(“hasRole(‘EDIT\_NOTE’)”)

<https://www.youtube.com/watch?v=l9chhjL7Kuk>

1. Difference b/n controller and rest controller

**Ans:** RestController is a combination of @Controller and @ResponseBody to use in Rest API. @RestController will return object simply either in JSON or XML format but @Controller is to create a Map of the model object.

1. Which design pattern u will use to call a 3rd party service which accepts only one id but our requirement is to get data for multiple ids.
2. What are the patterns u have used in REST API?
3. What are differences b/n SOAP and REST?
4. What is the significance of abstract class in java8?
5. How you handled the exception handling in your application?
6. Problem with cloneable while creating singleton?

**Ans:** Override clone() method and throw CloneNotSupportedException

1. Hashmap new implementation changes in java8?

Ans: In **Java 8**, **HashMap** replaces linked list with a binary tree when the number of elements in a bucket reaches certain threshold. While converting the list to binary tree, hashcode is used as a branching variable. ... This JDK 8 **change** applies only to **HashMap**, LinkedHashMap and ConcurrentHashMap.

1. Difference b/n Calleable and Runnable?

* Ans: A Callable needs to implement call() method while a Runnable needs to implement run() method.
* A Callable can return a value but a Runnable cannot.
* A Callable can throw checked exception but a Runnable cannot.
* A Callable can be used with ExecutorService#invokeXXX(Collection<? extends Callable<T>> tasks) methods but a Runnable cannot be.

1. Kstreams, ktable, flatMap in kafka

**Ans:** KStream is a data stream from a particular topic and Ktable contains only the data which satisfies the filter in the given predicate.

1. Java 8 Streams?

**Ans:** 1. Streams are useful to filter, sort, count the elements, iterate using forEach loop, apply map function.

* hasMap11.streams.sorted((e1, e2) -> e1.getId() - e2.getId());
* we can pipeline the streams. We can filter the data and that data we can send to sorted method to sort and then to forEach method to print.
* Collections are data structures which stores all data as inline memory where as streams are just flows which are computed on demand
* Collections are like a stored water tank(eagerly constructed) and Streams are like water flow through a pipe(lazy constructed)

1. How to avoid GIT merge conflicts?

Ans: there are two kinds of merge conflicts.

1. Is at the **beginning of the merge** if there are any changes identified in local or staging area with server. It is because of the pending changes in local. Then local state to be stabilized with git stash, git checkout, git commit, git reset.
2. **Conflict during the merge**, we have to cat the file content and check the conflicts manually and edit manually with proper data and then git add <file\_name> and then git commit.
3. in case if we are having more conflict the below commands would be useful.

* Git status -> helps to identify conflicted files
* Git log - -merge -> gives the list of commits that conflicts between the merging branches
* Git diff -> helps find differences between states of a repository/files. This is useful in predicting and preventing merge conflicts.
* Git reset - -mixes -> used to undo the changes in working or staging area.
* Git merge - -abort -> reverses the complete merge and sets back to previous state
* Git reset -> Git reset can be used during a merge conflict to reset conflicted files to a know good state

Black Knight:

1. What are the design patterns used by spring?
2. Internal implementation of LinkedList?

**Ans:** It is an implementation of doubly linked list. It contains node class which contains a value of Generic type, prev and next nodes of same reference.

1. Streams in java8
2. Try with resources

**Ans:** if there is any exception occurs, the close method of these resources will call and close them. Catch or Finally block will run after resources closed. The resources which implements AutoCloseable interface can only be used in this.

1. Difference b/n FileInputStream and FileReader?

**Ans:** 1. FileInputStream extends from InputStream where as FileReader is an extension of Reader

FileInputStream is used to read binary data where as FileReader is used to read text files of default encoding. FileReader automatically converts the raw bytes into characters using default encoding.

1. Diff b/n Const, let, var

**Ans:** var can be global/function scope. It can be redeclared or reassigned within the scope itself. Even if the var is reassigned in inner scope, the value would be updated globally which is a serious issue. It is assigned with undefined by default.

In other hand, let is blocked scope. You can reassign this within the scope but not redeclare within the same scope but you can redeclare in other scope but that would not impact the let declared in other scope. This won’t be assigned with any default value.

Const can’t be redeclared or reassigned. If const is a json object the data of the variables inside can be updated.

[**https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/#:~:text=var%20declarations%20are%20globally%20scoped%20or%20function%20scoped%20while%20let,be%20updated%20nor%20re%2Ddeclared**](https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/#:~:text=var%20declarations%20are%20globally%20scoped%20or%20function%20scoped%20while%20let,be%20updated%20nor%20re%2Ddeclared)**.**

1. Difference b/n @Qualifier and @Repository

**Ans:** There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the **@Qualifier** annotation along with **@Autowired** to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of @Qualifier annotation.

1. How to write setter and constructor based injection?

**Ans:** for constructor based injection you just need to use @Autowired, but for setter based injection you need to use @Autowired on top of setter method.

Also for constructor based injection, if there is only one constructor you can omit @Autowired also from spring 4.3

1. Steps to follow spring jpa or spring transaction management?

**Ans:** @EnableTransactionManagment on configuration class. Spring boot will do it for us. This step is not needed in spring boot.

@Transactional, will creates a transaction proxy for the repository to manage commit or rollback actions.

Autowire the repository and call save method.

* Required : create a new transaction or use if already existing
* Require\_new : a new transaction is needed for my method
* Not\_supported : don’t want any transaction even if there is a transaction present, suspend it.
* Never : don’t want any transaction, will cry if a transaction exists.
* Mandatory : an existing transaction is mandatory, I don’t open new transaction.

1. How you have handled exception handling in your application?
2. What is cross site scripting(XSS)?

**Ans:** Cross-site Scripting may also be used to [deface a website](https://www.acunetix.com/blog/news/full-disclosure-high-profile-websites-xss/) instead of targeting the user. The attacker can use injected scripts to change the content of the website or even redirect the browser to another web page, for example, one that contains malicious code.

[https://infosys.webex.com/meet/kranthirekha\_t](https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Finfosys.webex.com%2Fmeet%2Fkranthirekha_t&data=04%7C01%7Cvijayalakshmanakumar.mygapula%40externals.adidas.com%7C20fa3a7788d640658b0008d8841c927a%7C3bfeb222e42c4535aaceea6f7751369b%7C0%7C0%7C637404605619339433%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=zHRnLTz7BaN8Nf6P0oymhVPTezC8RaLbmQt6lvDGuP8%3D&reserved=0)