Java Interview Questions

By

Praveen Oruganti
Linktree: https://linktr.ee/praveenoruganti
Email: praveenorugantitech@gmail.com

Java Principal Architect interview @ Aristocrat
1.What is Apache Kafka? When it is used? What is the use of Producer, Consumer, Partition, Topic, Broker and Zookeeper while using Kafka?

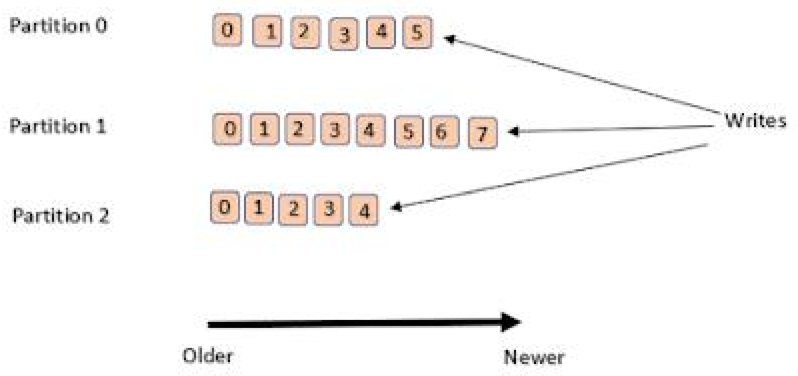
Apache Kafka is a distributed publish-subscribe messaging system.

It is used for real-time streams of data and used to collect big data for real-time analysis.

Topic: A topic is a category or feed or named stream to which records are published. Kafka stores topic in logs file.

Broker: A kafka cluster is a set of servers , each of which is called a broker.

Partition: Topics are broken up into ordered commit logs called partitions. Kafka spreads those log's partitions across multiple servers or disks.



Producer: A producer can be any application who can publish messages to a topic. The producer does not care what partition a specific message is written to and will balance messages over every partition of a topic evenly.

Directing messages to a partition is done using message key and a partitioner. Partitioner will generate a hash of the key and map it to a partition.

Producer publishes a message in the form of key-value pair.

Consumer: A consumer can be any application that subscribes to a topic and consume the messages.

A consumer can subscribe to one or more topics and reads the messages sequentially.

The consumer keeps track of the messages it has consumed by keeping track on the offset of messages.

Zookeeper: This is used for managing and coordinating kafka broker.

2.What are the Kafka features?

Kafka features are listed below:

- **High throughput** : Provides support for hundreds of thousands of messages with modest hardware.
- **Scalability** : Highly scalable distributed systems with no downtime.

- **Data loss** : Kafka ensures no data loss once configured properly.
- **Stream Processing** : Kafka can be used along with real time streaming applications like Spark and Storm.
- **Durability** : Provides support for persisting messages to disk.
- **Replication** : Messages can be replicated across clusters, which supports multiple subscribers.

3.What are Kafka components?

Kafka components are:

- **Topic**
- **Partition**
- **Producer**
- **Consumer**
- **Messages**

A topic is a named category or feed name to which records are published. Topics are broken up into ordered commit logs called partitions.
Each message in a partition is assigned a sequential id called an offset.

Data in a topic is retained for a configurable period of time.

Writes to a partition is generally sequential , thereby reducing the number of hard disk seeks.

Reading messages can either be from beginning and also can rewind or skip to any point in partition by giving an offset value.

A partition is an actual storage unit of kafka messages which can be assumed as a kafka message queue. The number of partitions per topic are configurable while creating it.

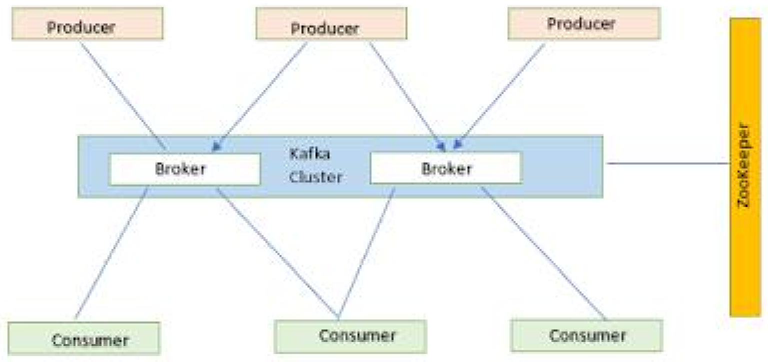
Messages in a partition are segregated into multiple segments to ease finding a message by its offset.

The default size of a segment is very high, i.e. 1GB, which can be configured.

Each segment is composed of the following files:

- Log: Messages are stored in this file.
- Index: stores message offset and its starting position in the log file.
- TimeIndex

4.What is a Kafka cluster?
Kafka cluster is a set of servers and each server is called a broker.

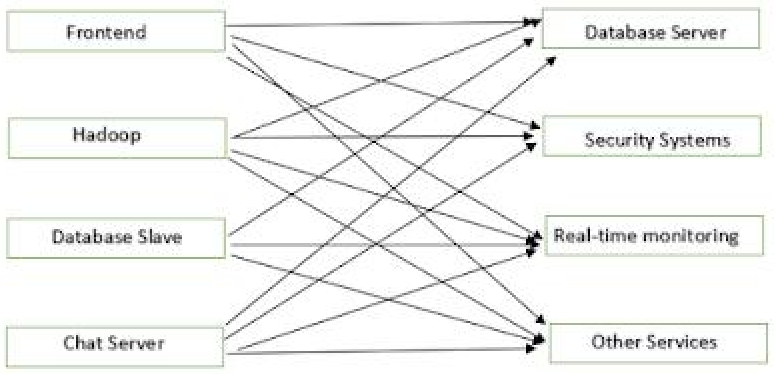


In the above diagram, Kafka cluster is a set of servers which are shown with Broker name. Producers publish messages to topics in these brokers and Consumers subscribe to these topics and consume messages.

Zookeeper is used to manage the coordination among kafka servers.

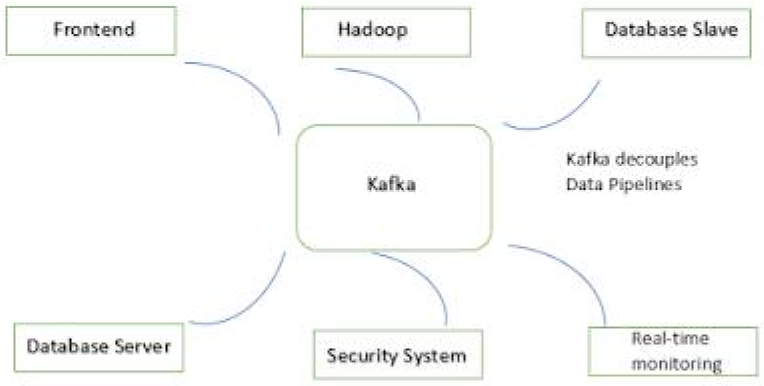
5.What problem does Kafka resolve?

Without any messaging queue implementation, what the communication between client nodes and server nodes look alike is shown below:



There is a large numbers of data pipelines which are used for communication. It is very difficult to update this system or add another node.

If we use Kafka, then the entire system will look like something:



So, all the client servers will send messages to topics in Kafka and all backend servers will consume messages from kafka topics.

6.What is the purpose of using @ServletComponentScan?

We add @ServletComponentScan to enable scanning for @WebFilter, @WebServlet and @WebListener.

It is used on the main SpringBootApplication.java class.

Embedded containers do not support @WebServlet, @WebFilter and @WebListener. That's why spring has introduced @ServletComponentScan annotation to support some dependent jars which use these 3 annotations.

To use @ServletComponentScan, we need to use spring boot with version 1.3.0 or above.

And we also need to add spring-boot-starter-parent and spring-boot-starter-web dependencies.

pom.xml file:

```
<parent>

   <groupId> org.springframework.boot </groupId>

   <artifactId> spring-boot-starter-parent</artifactId>

   <version> 1.5.1.RELEASE </version>

</parent>


<dependencies>

   <dependency>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-starter-web</artifactId>

      <version>1.5.1.RELEASE</version>

   </dependency>

</dependencies>
```

7.How does Servlet work and what are the lifecycle methods?

A servlet is a class that handles requests, processes them and reply back with a response.

e.g. we can use a servlet to collect input from user through an HTML form, , query records from a database and create web pages dynamically.

Servlets are under the control of another java application called servlet container. When an application running in a web server receives a request , the server hands the request to the servlet container - which in turn passes it to the target servlet.

Maven dependency for using servlet is given below:

```
<dependency>

   <groupId> javax.servlet</groupId>

   <artifactId> javax.servlet-api</artifactId>

   <version> 3.1.0</version>

</dependency>
```

Lifecycle methods of servlet are described below:

init(): The init method is designed to be called only once. If an instance of servlet does not exist, the web container does the following:

- Loads the servlet class
- Create an instance of the servlet class
- Initializes it by calling the init() method.

The init() method must be completed successfully before the servlet can receive any requests.
The servlet container cannot place the servlet in service if the init() method either throws a ServletException or does not return within a time period defined by the web server.

```
public void init throws ServletException{

     // code here

}
```

service(): This method is only called after the servlet's init() method has completed successfully.

The container calls the service method to handle the requests coming from the client, interprets the HTTP request type (GET, PUT, POST, DELETE etc.) and calls doGet(), doPut(), doPost() and doDelete() methods.

```
public void service(ServletRequest req, ServletResponse response) throws ServletException{

}
```

destroy(): It is called by the servlet container to take the servlet out of the service.

This method is only called after all the threads in the service have exited or a time period has passed.

After the container calls this method, it will not call the service method again on the servlet.

```
public void destroy(){

}
```

8.What application server have you used and what are the benefits of that?

I have used Weblogic application server.

Weblogic server provides various functionalities:

- Weblogic server provides support for access protocols like HTTP, SOAP etc.
- It also provides data access and persistence from database server.
- It also supports SQL transactions for data integrity.
- It also provides security.

So means, when we use Weblogic server, we do not have to care about protocol, security, database transactions, data integrity etc. All these are handled by Weblogic server. We can focus on business logic.

Java Interview @ Polaris
1.

```
public class Quiz23{

    public static void main(String[] args){

        int x = 0;
        int[] nums = {1,2,3,5};

        for(int i : nums){

            switch(i){

                case 1:
                    x += i;
                case 5:
                    x += i;
                default:
                    x += i;
                case 2:
                    x  += i;

            }
            System.out.println(x);
        }
    }
}
```

What will be the output of above code?

27

Explanation:
For case i = 1: All the cases will run.
For case i = 2: only case 2 will run.
For case i = 3: Default and case 2 will run.

For case i = 5: case , default and case 2 will run.

2.What is the difference between Map and FlatMap?

Map: Transforms the elements into something else. It accepts a function to apply to each element and returns a new Stream of values returned by the passed function.

It takes function object as the parameter e.g.: Function, ToIntFunction, ToIntDoubleFunction, ToLongFunction.

FlatMap: Combination of Map and Flat operations.So, we first apply map operation on each element and then flattens the result.

So, if function used by Map is returning a single value, then map is ok. But, if function used by Map operation is returning a stream of list or stream of stream, then we need to use flatmap to get stream of values.

For example:

If we have a stream of String containing {"12", "34"} and a method getPermutations() which returns a list of permutations of given string.

When we apply getPermutation() into each string of Stream using map , we get something like [["12","21"],["34","43"]], but if we use flatMap, we get a stream of strings e.g.: ["12","21","34","43"].

Another example:

```
List evens = Arrays.asList(2,4,6);
List odds = Arras.asList(3,5,7);
List primes = Arrays.asList(2,3,5,7,11);

List numbers = Stream.of(evens, odds, primes).flatMap(list -> list.stream()).collect(Collectors.toList());

System.out.println("flattened list : "+numbers);
```

Output: flattened list : [2,4,6,3,5,7,2,3,5,7,11]

3.What is the difference between passing int array and String array to Stream.of()?

Stream.of(int[]) gives Stream<int[]>
Stream.of(String[]) gives Stream<String>

So, when using Stream.of() with int[] , we get Stream<int[]> and then for getting ints from Stream, we use flatMapToInt(i -> Arrays.Stream(i)) to get IntStream and then we can either use map() or forEach().

e.g.:
```
int[] arr = {1,2,3,4};
Stream<int[]> streamArr = Stream.of(arr);
IntStream intStream = streamArr.flatMapToInt(i -> Arrays.Stream(i));
intStream.forEach(System.out :: println);
```

4.What is a Boxed Stream?

If we want to convert stream of objects to collection:

```
List<String> strings = Stream.of("how", "to", "do", "in", "java").collect(Collectors.toList());
```

The same process doesn't work on streams of primitives, however.

```
//Compilation Error:
IntStream.of(1,2,3,4,5).collect(Collectors.toList());
```

To convert a stream of primitives, we must first box the elements in their wrapper class and then collect them. This type of stream is called boxed stream.

Example of IntStream to List of Integers:

```
List<Integer> ints = IntStream.of(1,2,3,4,5).boxed().collect(Collectors.toList());

System.out.println(ints);
```

Output: [1,2,3,4,5]

5:List Spring Core and Stereotype annotations.

Spring Core Annotations:

- @Qualifier
- @Autowired
- @Configuration
- @ComponentScan
- @Required
- @Bean
- @Lazy
- @Value

Spring framework Stereotype annotations:

- @Component
- @Controller
- @Service
- @Repository

6.What is difference between @Controller and @RestController?

@Controller creates a Map of model object and finds a view for that.
@RestController simply returns the object and object data is directly written into HTTP response as JSON or XML.

@RestController = @Controller + @ResponseBody
@RestController was added in Spring 4.

7.Why Spring introduced @RestController when this job could be done by using @Controller and @ResponseBody?

The functioning or output of @Controller and @ResponseBody is the default behavior of RESTFUL web services, that's why Spring introduced @RestController which combined the behavior of @Controller and @ResponseBody.

8.Tell something about Swagger tools which you have used.

Swagger Editor: Can edit OpenAPI specifications.

Swagger UI : A collection of HTML/CSS/Javascript assets that dynamically generate beautiful documentation.

Swagger Codegen: Allows generation of API client libraries.

Swagger Parser: Library for parsing OpenAPI definitions from java.

Swagger Core: Java related libraries for creating, consuming and working with OpenAPI definitions.

Swagger Inspector: API testing tool

SwaggerHub : Built for teams working with OpenAPI.

Java Technical Interview @ IBM
1.How locking mechanism is implemented by JVM?

The implementation of locking mechanism in java is specific to the instruction set of the java platform.

For example with x86, it might use the CMPXCHG - atomic compare and exchange - at the lowest level to implement the fast path of the lock.

The CMPXCHG instruction is a compare-and-swap instruction that guarantees atomic memory access at the hardware level.

If the thread cannot acquire the lock immediately, then it could spinlock or it could perform a syscall to schedule a different thread.
Different strategies are used depending on the platform, JVM switches.

2.Which Event bus is used by Saga pattern?

When we use event based communication, a microservice publishes an event when something notable happens such as when it updates a business entity. Other microservices subscribe to those events.

When a microservice receives an event, it can update it's own business entities which might lead to more events being published.

This publish/subscribe system is usually performed by using an implementation of an event bus.
The event bus is designed as an interface with the API needed to subscribe and unsubscribe to events and to publish events.
The implementation of this event bus can be a messaging queue like RabbitMQ or service bus like Azure service bus.

3.What is the use of Amazon EC2? What are the steps to deploy on EC2?

Amazon EC2: Amazon Elastic Compute Cloud
It offers ability to run applications on the public cloud.

It eliminates investment for hardware. There is no need to maintain the hardware.We can use EC2 to launch as many servers as we need.

Steps to deploy on EC2:

- Launch an EC2 instance and SSH into it. Note: This instance needs to be created first on Amazon console [console.aws.amazon.com]. And we should also have certificate to connect to EC2 instance.
- Install Node on EC2 instance, if our app is in Angular.
- Copy paste code on EC2 instance and install all dependencies.
- Start server to run.

OR:

- Build Spring boot app in the local machine. Make .jar file.
- Upload this .jar on S3.
- Create EC2 instance.
- SSH into it from the local computer. Now, we are in EC2 instance.
- We can install JDK now.
- And using java - .jar file path, we can run our application.

4.Why should we use ThreadPoolExecutor, when we have Executor Framework?

Source code of Executors.newFixedThreadPool() is:

```
public static ExecutorService newFixedThreadPool(int nThreads){

    return new ThreadpoolExecutor(nThreads, nThreads, oL,       TimeUnit.MILLISECONDS, new
LinkedBlockingQueue<Runnable>());

}
```

This method uses ThreadPoolExecutor class which uses default configuration as is seen in above code. Now, there are scenarios where default configuration is not suitable, say instead of LinkedBlockingQueue, a PriorityQueue needs to be used etc.
In such cases, caller can directly work on underlying ThreadPoolExecutor by instantiating it and passing desired configuration to it.

Note: One advantage of using ThreadPoolExecutor is that we can handle RejectedExecutionException using ThreadPoolExecutor.discardPolicy().

5.What is the difference between Spring 2 and Spring 5?

Below are the differences between Spring 2 and Spring 5:

- JDK baseline update
- Core framework revision
- Reactive programming model
- Core Container updates
- Testing improvements

6.What is the difference between @SpringBootApplication and @EnableAutoConfiguration?

Following are the differences between @SpringBootApplication and @EnableAutoConfiguration:

Availability: @SpringBootApplication was introduced in version 1.2, while @EnableAutoConfiguration was introduced in version 1.0.

Purpose: @EnableAutoConfiguration enables auto configuration feature of Spring Boot application which automatically configures things if certain classes are present in classpath e.g. it can configure Thymeleaf TemplateResolver and ViewResolver if Thymeleaf is present in the classpath.

On the other hand, @SpringBootApplication does three things:

- It allows us to run the main class as a jar with an embedded container [Web server Tomcat].
- It enables java configuration.
- It enables component scanning.

7.What happens when we call SpringApplication.run() method in main class of SpringBoot application?

Syntax of the class containing main method looks like code below:

```
@SpringBootApplication
public class StudentApplication{
   public static void main(String[] args){

      return SpringBootApplication.run(StudentApplication.class, args);
   }

}
```

When we run this class as a java application, then our application gets started.

SpringApplication.run() is a static method and it returns an object of ConfigurableApplicationContext.

ConfigurableApplicationContext ctx = SpringApplication.run(StudentApplication.class, args);

Thus, Spring container gets started once run() method gets called.

Spring container once started is responsible for:

- Creating all objects: This is done by @ComponentScan. Remember @SpringBootApplication is a combination of @ComponentScan + @Configuration + @EnaleAutoConfiguration
- Dependency Injection
- Managing the lifecycles of all beans.

Java Interview @ Indie Games
1.How do you kill a thread in java?

There are two ways to kill a thread:

- Stop the thread [Deprecated now]
- Use a volatile variable and running thread will keep checking it's value and return from it's run method in an orderly fashion.

2.Why is Thread.stop() deprecated?

Because it is inherently unsafe. Stopping a thread causes it to unlock all the monitors that it has locked.(The monitors are unlocked as the ThreadDeath exception propagates up the stack.)
If any of the objects previously protected by these monitors were in an inconsistent state, other threads may now view these objects in an inconsistent state. Such objects are said to be damaged.

When threads operate on damaged objects, arbitrary behavior can result. This behavior may be subtle and difficult to detect.

ThreadDeath exception kills thread silently, thus the user has no warning that this program may be corrupted.

3.How to check if a String is numeric in java?

We can use Apache Commons Lang 3.4 and 3.5 versions.
StringUtils.isNumeric() or NumberUtils.isParsable().

4.How can we timeout a thread? I want to run a thread for some fixed amount of time.If it is not completed within that time, I want to either kill it or throw some exception. How to do it?

We can use ExecutorService framework of Java 5.

```
public class Test{

    public static void main(String[] args){

        ExecutorService service = Executors.newSingleThreadExecutor();
        Future<String> future = service.submit(new Task());

        try{

            future.get(4, Timeout.SECONDS);
        }
        catch(Exception e){
            future.cancel(true);
        }

        executor.shutDownNow();

    }

}

class Task implements Callable<String>{

    @Override
    public String call() throws Exception{

        Thread.sleep(5000);
        return "";
    }

}
```
In this code, thread sleeps for 5 seconds, but future will wait for 4 seconds and it will timeout and cancel it.

5.How to wait for all threads to finish using ExecutorService?

We can use shutdown() and awaitTermination() methods of ExecutorService.

```
ExecutorService service = Executors.newFixedThreadPool(4);

while(...){
    service.execute(new Task());

}
service.shutdown();
try{
    service.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
}
catch(InterruptedException e){

}
```

6.Can we create deadlock without thread in java?

It is not possible to run the code without atleast one thread. A single thread can block itself in some cases e.g. attempting to upgrade a read lock to a write lock.
When a thread resource starves, it is called a livelock.

It is also possible to create a deadlock without creating an additional thread e.g. the finalizer thread and the main thread can deadlock each other.

Interview @ 1mg
1.What is System.out, System.in and System.err?

out, in and err all are fields in System class.

out: The standard output stream.This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.
Basically, it gives PrintStream. And all print() methods belong to class PrintStream.

```
public static final PrintStream out
```

in: The standard input Stream. This stream is already open and ready to supply input data. typically this stream corresponds to keyboard input or other input source specified by the host environment or user.

```
public static final InputStream in
```

err: The standard error output stream. This stream is already open and ready to accept output data.
By convention, this output stream is used to display error messages.

```
public static final PrintStream err
```

2.What is the difference between Class.forName() and Class.forName().newInstance() methods?

Lets take an example to understand the difference better:

```
public class Demo{

    public Demo(){


    }

    public static void main(String[] args){
        Class clazz = Class.forName("Demo");
        Demo demo = (Demo)clazz.newInstance();
    }

}
```

Class.forName() returns the Class object associated with the class or interface with the given string name.

Then, calling clazz.newInstance() creates a new instance of the class represented by this Class object. This class is instantiated as if by a new expression with an empty argument list.

3.Why non-static variables are not allowed in static methods?

non-static variables means instance variables and they are only initialized when an instance is created. As static methods can be called without creating an instance , so accessing non-initialized instance variable is wrong as instance doesn't exist.

So, only way to access non-static variable in static method is that , just create instance of class in static method and access that variable.

```
public class StaticTest{

    private int count = 0;

    public static void main(String[] args){

        StaticTest test = new StaticTest();

        test.count++;

    }

}
```

4.What is the difference between HTTP HEAD and GET verbs?

HTTP HEAD is almost identical to GET, but without the response body. Means in HTTP HEAD, we don't get any response body.

In other words, if GET /users returns a list of users, then HEAD /users will make the same request , but will not return the list of users.

HEAD requests are useful for checking what a GET request will return before actually making a GET request - like before downloading a large file or response body.

5.What is the difference between HTTP GET and POST methods?

There are multiple differences between GET and POST methods:

- GET is used to request data from a specified resource. POST is used to send data to server to create a resource.
- GET requests can be cached. POST requests cannot be cached.
- GET requests remain in the browser history. POST requests do not remain in the browser history.
- GET requests can be bookmarked. POST requests cannot be bookmarked.
- GET requests have length restrictions. POST requests have no restrictions on data length.

}

6.Is there any speed increase while indexing a table? And will indexing every column defeat the purpose of indexing?

Indexing any table, either memory or file system based, will speed up queries that select or sort results based on that column.
This is because the index works like a tree structure and the search distance depends upon the depth of the tree, which increases a lot slower than the row count of the column.

Indexing every column doesn't defeat the purpose of the index, but it will slow up inserts and updates because those changes will cause an update of every index of that table.
Also, the indexes take up space on the database server.

7.What is covariant return type?

In covariant return type, parent's instances can be replaced with child's instances.

e.g.:

```
class WildAnimal{

    public String willYouBite(){
        return "Yes";
    }

}

class Lion extends WildAnimal{

    public String whoAreYou(){

        return "Lion";
    }

}


class BengalTiger extends WildAnimal{

    public String whoAreYou(){

       return "Tiger";
    }
}


class Zoo{

    public WildAnimal getWildAnimal(){

        return new WildAnimal();
    }

}
```

```
class AfricaZoo extends Zoo{

    @Override
    public Lion getWildAnimal(){

        return new Lion();
    }

}

class IndiaZoo extends Zoo{

    @Override
    public BengalTiger getWildAnimal(){

        return new BengalTiger();
    }
}


public class Covariant{

    public static void main(String[] args){

        AfricaZoo africaZoo = new AfricaZoo();
        System.out.println(africaZoo.getWildAnimal().whoAreYou());
    }

}
```

So, in class AfricaZoo, parent class WildAnimal is replaced by child class Lion while overriding method.

Java Technical Lead interview @ Accenture
1.What is the difference between SSL and TLS?

There are multiple differences between SSL and TLS:

Alert Messages:
SSL has "No Certificate" alert message. TLS protocol removes the alert message and replaces it with several other alert messages.

Record Protocol:
SSL uses MAC [Message Authentication Code] after encrypting each message while TLS on the other hand uses HMAC - a hash based message authentication code after each message encryption.

Handshake Process:
In SSL, the hash calculation also comprises the master secret and pad while in TLS, the hashes are calculated over handshake message.

2.What changes are introduced in Java 10?

There are multiple changes that have been done in java 10. Explaining all the changes below:

- var keyword

- Unmodifiable Collection enhancements
- G1GC performance improvement
- New JIT compiler [Created in pure java]
- Alternate memory devices for allocating heap
- Application class data sharing

var keyword:
It improves readability. It is not a reserved word, means it can be used as a class name or variable name.
What the var keyword does is , turn local variable assignments:

```
HashMap<String, String> hm = new HashMap<>();
```

into

```
var hm = new HashMap<String,String>();
```

Unmodifiable Collection enhancements:

```
var vegetables = new ArrayList<>(Lists.of("Brocolli", "Celery", "Carrot"));

var unmodifiable = Collections.unmodifiableList(vegetables);
vegetables.set(0, "Radish");
var v = unmodifiable.get(0); //  var v contains Radish.
```

It is so because, unmodifiableList() returns an unmodifiable view collection.
An unmodifiable view collection is a collection that is unmodifiable and that is also a view onto a backing collection.

Note that changes in the backing collection is still possible , and if they occur, they are visible through the unmodifiable view.

So, if we need completely unmodifiable collection then, java 10 has added two new API's.

```
var unmodifiable = List.copyOf(vegetables);
```

The **second API** adds three new methods to the Collector class in the Stream package.

We can now stream directly into an unmodifiable collection using toUnmodifiableList, toUnmodifiableSet and toUnmodifiableMap.

3.What is the improvement in G1GC in java 10?

Java 9 made the Garbage-First garbage collector (G1GC) by default , replacing the concurrent Mark-sweep garbage collector (CMS).

Java 10 introduces performance improvements to G1GC.

In java 10, G1GC is getting a performance boost with the introduction of full parallel processing during a full GC.

4.What is Application class-data sharing?

Java 5 introduced Class-Data sharing (CDS) to improve startup times of small java applications.

The general idea was that when the JVM first launched , anything loaded by the bootstrap classloader was serialized and stored in a file on disk that could be reloaded on future launches of the JVM.
This means that multiple instance of the JVM shared the class metadata so it wouldn't have to load them all every time.

5.How to write thread safe code in java?

Example of Non thread safe code in java:

```
public class Counter{

    private int count;

// This method is not thread safe because ++ is not an atomic operation.
    public int getCount(){

        return count++;
    }

}
```

How to make code thread safe in java:

There are multiple ways to make this code thread safe:

1). Use synchronized keyword in java and lock the getCount() method so that only one thread can execute it at a time.

2). Use atomic integer , which makes this ++ operation atomic and since atomic operations are thread-safe and saves cost of external synchronization.

Below is the thread-safe version of Counter class in java:

```
public class Counter{

    private int count;

    AtomicInteger atomicCount = new AtomicInteger(0);

    //This method is thread safe now because of locking and synchronization

    public synchronized int getCount(){

        return count++;
    }

    //This method is thread safe because count is incremented atomically.

    public int getCountAtomically(){

        return atomicCount.incrementAndGet();
    }
}
```

6.How does AtomicInteger work?

Java 5 introduced java.util.concurrent.atomic package with a motive to provide a small kit of classes that support lock-free thread-safe programming on single variables.

AtomicInteger uses combination of volatile and CAS [Compare and Swap] to achieve thread safety for Integer Counter.

It is non blocking in nature.

Compare-and-Swap:

CAS is an atomic instruction used in multi-threading to achieve synchronization.
It compares the contents of a memory location with a given value and only if they are same, modifies the contents of that memory location to a given new value.
This is done as a single atomic operation.

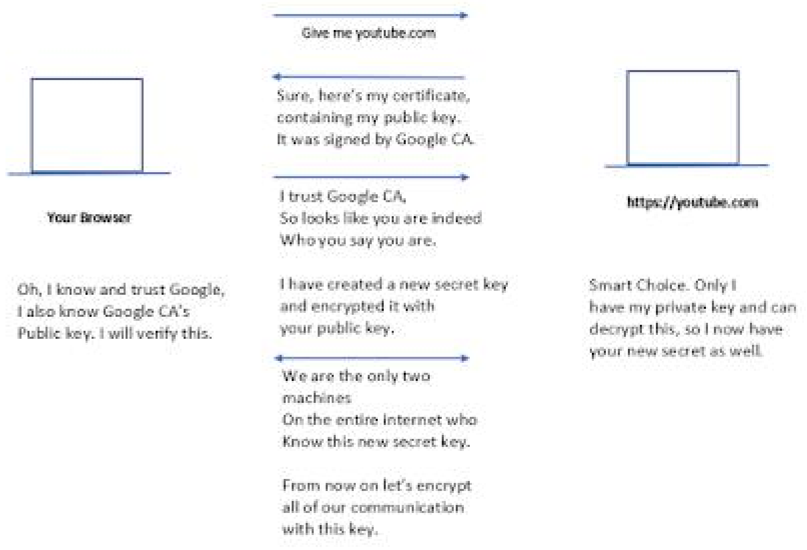
The atomicity guarantees that the new value is calculated based on up-to-date information. If the value has been updated by another thread in the meantime, the write would fail.

Java Interview @ Altran
1.How does HTTPS work?

In HTTPS, every message is encrypted or decrypted by use of public/private keys.

So, we need to trust that public key cryptography and signature works.

- Any message encrypted with Google's public key can only be decrypted with Google's private key.
- Anyone with access to Google's public key can verify that a message (signature) could only have been created by someone with access to Google's private key.


Give me youtube.com
Sure, here's my certificate, containing my public key. It was signed by Google CA.
Your Browser
https://youtube.com
I trust Google CA, So looks like you are indeed Who you say you are.
Oh, I know and trust Google, I also know Google CA's Public key. I will verify this.
I have created a new secret key and encrypted it with your public key.
Smart Choice. Only I have my private key and can decrypt this, so I now have your new secret as well.
We are the only two machines On the entire internet who Know this new secret key.
From now on let's encrypt all of our communication with this key.

2.What is a Certificate Authority? How is a certificate signed?

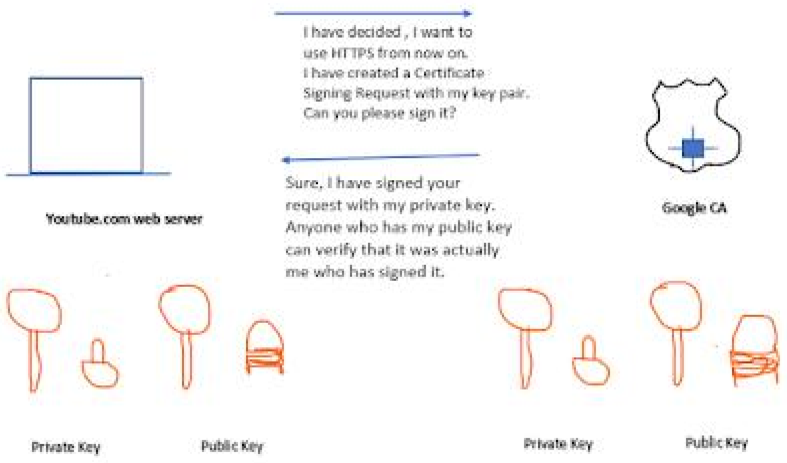
Suppose youtube.com web server uses HTTP currently. And now youtube wants to secure communication using HTTPS.

Also there is Google CA which is considered as a trusted certificate authority.

As any party envolved in public key cryptography , the google certificate authority has a private key and a public key.

As youtube.com wants to communicate using HTTPS, they also need to create a new pair: public key and private key.

Now youtube.com web server creates a Certificate signing request with this key pair.


I have decided , I want to use HTTPS from now on. I have created a Certificate Signing Request with my key pair. Can you please sign it?
Sure, I have signed your request with my private key. Anyone who has my public key can verify that it was actually me who has signed it.
Youtube.com web server
Google CA
Private Key
Public Key
Private Key
Public Key

So most of the browsers [when they are delivered] has a list of these certificates issued by known Certificate Authorities [Google, Symantec, Thwate].

So, when we try to open youtube.com, then as youtube.com uses HTTPS, it sends certificate to the browser. As browser knows the Google's public key [as it is having that certificate with Google's public key], so it can use that key to know that this certificate is actually signed by trusted certificate authority [Google].

Means. now that public key can be used to create and encrypt a secret key which will then be used in symmetric encryption.

3.Explain the difference between HTTP, HTTPS, SSL and TLS?

HTTP: Hyper Text Transfer Protocol

Using this protocol, data is transferred in clear text.

HTTPS: Secure Hypertext Transfer Protocol

Means, HTTP with a security feature.

Encrypts the data that is being retrieved by HTTP. So, it uses encryption algorithms to encrypt the data.

HTTPS uses SSL [Secure Socket Layer] protocol to protect the data.

SSL: This protocol is used to ensure security on the internet. Uses public key encryption to secure data.

TLS: Transport Layer Security

It is the latest industry standard cryptographic protocol. It is a successor to SSL.
And it is based on same specifications.
And like SSL, it also authenticates the server, client and encrypts the data.

4.What are the collection types in hibernate?

Below is the list of collection types in hibernate:
- Bag
- List
- Map
- Array
- Set

5.What is hibernate proxy?

Mapping of classes can be made into a proxy instead of a table.
A proxy is returned when actually a load() is called on a session. The proxy contains actual method to load the data.

The proxy is created by default by hibernate for mapping a class to a file.

6.What is the difference between load() and get() methods in hibernate?

Hibernate's session interface provides several overloaded load() [It will not hit database] methods for loading entities. Each load() method requires the object's primary key as an identifier and it is mandatory to provide it.

In addition to the ID, hibernate also needs to know which class or entity name to use to find the object with that ID.

In case of get() method, we will get return value as NULL if identifier is absent.
But in case of load() method, we will get a runtime exception [ObjectNotFoundException].

When we call load() on hibernate's session object, then hibernate doesn't make a call to database. It creates and returns a proxy object. Now, when some state is fetched from the proxy object, then hibernate issues the appropriate SQL statement to database and builds the real persistent object.

So, when actual data is requested using proxy object and if no data exist for the identifier [id], then ObjectNotFoundException is thrown.

On the other hand, when we call get() method on hibernate's session object, then hibernate immediately issues a SQL statement to the database to fetch associated data [usually a row in database] to rebuild the requested persistent object.
So, if no data is found for the requested identifier [id], the method will return null.

Multithreading Interview in Ericsson
1.Can a thread acquire multiple locks at the same time and how?

Yes, a thread can acquire locks on multiple objects at the same time.
Also, a thread can acquire lock on the same object multiple times.

Locks obtained using synchronized are implicitly reentrant.

Whenever we use nested synchronized blocks with 2 different objects, then the same thread can acquire lock on both of these objects. But care needs to be taken in this case, as if we take two pairs of synchronized blocks with different order of similar objects, then it may cause deadlock.

2.What are the steps to avoid deadlock in java programs?

There are multiple ways in which we can avoid deadlocks:
- Avoid using nested synchronized blocks
- Always request and release locks in the same order.
- Use synchronization only whenever required.
- If two threads are waiting for each other to release shared resources, then this issue can be resolved by letting each waiting thread retry the operation at random interval until they successfully acquire the resource.
- Do not perform operations that can block while holding a lock. e.g. Do not perform file I/O over the network while holding a lock.

3.When we call Thread object's run() method, then in which thread that run() method gets executed?

Calling run() method on Thread's object will cause execution of that method in current thread.

If a thread object was constructed by instantiating the subclass of Thread and fails to override run() method, then any calls to the run() method will invoke Thread.run(), which does nothing.

So, we should never call run() method on thread object.

4. When we can and should use ThreadGroup?

Each thread in java is assigned to one ThreadGroup upon it's creation. These groups are implemented by java.lang.ThreadGroup class.

When thread group name is not specified, the main default thread group is assigned by JVM.

ThreadGroups are useful for keeping threads organized.

There are few useful methods which can be used for specific use cases.

ThreadGroup.activeCount():
- Returns an estimated number of active threads in the current thread's thread group and it's subgroups.

ThreadGroup.enumerate():

- It copies into the specified array every active thread in this thread group and it's subgroups.

5.How to decide which method to call : notify() or notifyAll() ?

Whenever we call notify() or notifyAll() methods, it is guaranteed that only one thread will get the lock and resume execution.

Calling notify() is permitted under following conditions:

- All waiting threads have identical condition predicates.
- All threads perform the same set of operations after waking up. That is, any one thread can be selected to wake up and resume for a single invocation of notify.
- Only one thread is required to wake upon the notification.

6.Why to use Thread pool for handling multiple tasks?

Thread Pools allows a system to limit the number of simultaneous requests that it process to a number that it can comfortable serve.

If we don't use Thread Pool then, for every request, creating a new thread consumes a lot of time and resources and for task processing.

Describing here some of the benefits of using Thread Pool:

- Thread pool minimizes the overhead of thread lifecycle management because the threads in a thread pool can be reused and can be efficiently added or removed.
- It also reduces the time and resources required for thread creation.
- It also ensure graceful degradation of service when traffic bursts.

7.What is bounded thread pool and what happens when we execute interdependent tasks in bounded thread pool?

A bounded thread pool is a pool in which we specify an upper limit on the number of threads that can currently execute in a thread pool.

We should not use bounded thread pool to execute tasks that depend on the completion of other tasks in the pool.

A form of deadlock called thread-starvation deadlock arises when all the threads executing in the pool are blocked on tasks that are waiting on an internal queue for an available thread in which to execute.

Code Review Practices

While doing the code review, multiple factors are taken into account : Clean code, Security, performance, General points etc.

I'm explaining each one of them below:

Java Code Review Checklist:

Clean Code:

- **Use intention-revealing names**
- Names should be such that, they reveal the purpose.
- **Use solution-problem domain names**
- Names should be such that they tell about the actual solution or problem.
- **Classes should be small**
- Keep the code in a class as less as possible and create other classes or subclasses for specific purpose.
- **Functions should be small**
- Always break the functions in small.
- **Functions should do one thing**
- Keep one separate function for each action.
- **Don't repeat yourself (Avoid duplication)**
- Don't write duplicate codes. Check the entire code before writing the same code twice in the project.
- **Explain yourself in code :**
- Write proper Class level and method level Comments.
- **Use exceptions rather than return codes**
- **Don't return null**
- Never return null values from a function.

Security:

- Make class final if not being used for inheritance
- Avoid duplication of code
- Minimize the accessibility of classes and members
- Document security related information
- Input into a system should be checked for valid data size and range
- Release resources[Streams , Connections] in all cases.
- Purge sensitive information from exceptions
- Don't log highly sensitive information
- Avoid dynamic SQL, use prepared statement
- Limit the accessibility of packages, classes interfaces, methods and fields.
- Avoid exposing constructors of sensitive classes.
- Avoid serialization of sensitive classes
- Only use JNI when necessary

Performance:

- **Avoid excessive synchronization**
- Don't use synchronize constructs unneccessarily
- **Keep synchronized sections small**
- **Beware the performance of String concatenations**
- Avoid joining strings as much as possible.
- **Avoid creating unnecessary objects.**
- Try to create only local objects and also create them based on actual need.

General:

- Don't ignore exceptions
- Return empty Arrays or Collections , not nulls
- In public classes, use accessor methods not public methods
- Avoid finalizers
- Refer to objects by their interfaces
- Always override toString()
- Document thread safety
- Use marker interfaces to define types

Static Code Analysis:

- Check static code analyzer report for the classes added/modified

Java Technical Architect interview
1.What is the difference in using Eureka and spring cloud consul?

Eureka and Spring cloud Consul are both Service Discovery tools.
Difference between them lie in multiple factors.

The architecture of Eureka is primarily client/server with a set of eureka servers per datacentre , usually one per availability zone.
Typically clients of eureka use an embedded SDK to register and discover services.

Eureka provides a weak consistent view of services using best effort replication. When a client registers with a server , that server will make an attempt to replicate to the other servers but provides no guarantee. Service registrations have a short Time-To-Live, requiring clients to heartbeat with the servers. Unhealthy services or nodes will stop heartbeating, causing them to timeout and be removed from the registry.

Consul provides a super set of features, including richer health checking, key/value store and multi datacentre awareness. Consul requires a set of servers in each datacentre , along with an agent on each client, similar to using a sidecar like Ribbon. The Consul agent allows most applications to be consul unaware, performing the service registration via configuration files and discovery via DNS or load balancer sidecars.

Consul provides a strong consistency guarantee since servers replicate state using the Raft protocol.
Consul supports a rich set of health checks.
Client nodes in Consul participate in a gossip based health check , which distributes the work of health checking, unlike centralized heartbeating which becomes a scalability challenge.

In Consul, discovery requests are directed to the elected consul leader which allows them to be strongly consistent by default.

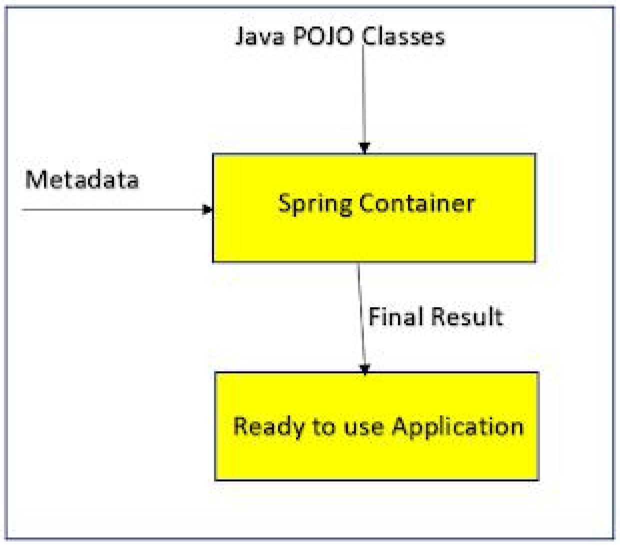
The strongly consistent nature of Consul means it can be used as a locking service for cluster coordination. Eureka does not provide similar guarantee and typically requires running ZooKeeper for services that need to perform coordination or have stronger consistency needs.

2.How many types of containers are there in spring framework?

There are two types of Spring IOC containers:

- BeanFactory Container

- ApplicationContext Container



Spring BeanFactory container is the simplest container which provides basic support for DI.
It is defined by org.springframeworkbeans.factory.BeanFactory interface. XMLBeanFactory is the main implementation of this interface. It reads configuration metadata from xml files for creating a fully configured application.

BeanFactory container is preferred where resources are limited to mobile devices or applet based applications.

Spring ApplicationContext Container:

It is defined by org.springframework.context.ApplicationContext interface.
The ApplicationContext container has all the functionalities of BeanFactory container. It is generally recommended over BeanFactory container.

The most common implementations are :

- FileSystemXmlApplicationContext
- ClassPathXmlApplicationContext
- WebXmlApplicationContext

3.What is the difference between @Controller and @RestController?

@RestController = @Controller + @ResponseBody

@RestController was added in Spring 4.

@Controller : The job of @Controller is to create a map of model object and find a view.

@ResponseBody: It will just append the result object as JSON or XML in Http response.

@RestController : It simply return the object and object data is directly written into HTTP response as JSON or XML.

4.What is the difference between @RequestParam and @RequestAttribute?

@RequestParam:

This annotation is used to access the query parameters from the HTTP request URL. e.g.:

http://example.com?param1=1&param2=2

In this URL, param1 and param2 are query parameters and they are accessed in methods like:

```
@RequestMapping("/")
public void method(@RequestParam("param1") String par){

}
```

@RequestAttribute:

It is used to access objects which have been populated on the server-side but during the same HTTP request.
e.g.:

We want to maintain a counter to know the number of visits to that page or number of requests made to a specific URL. So, in this case , we take an interceptor with AtomicInteger to increment the counter on every request.

e.g.:

The Interceptor:

```
public class Counter  extends HandlerInterceptorAdapter{

   private AtomicInteger counter = new AtomicInteger(0);

   @Override
   public boolean preHandle(HttpServletRequest request, HttpServletResponse response,Object handler)
throws Exception{

       request.setAttribute("visitorCounter", counter.incrementAndGet());
   }

}
```

The Controller class:

```
public class CounterController{

   @RequestMapping("/")
   @ResponseBody
   public String handle(@RequestAttribute("visitorCounter") Integer counter){

       // code here.......
   }
}
```

Interview questions on SpringBoot
1.What is the use case of Spring Boot?
Spring Boot is used while creating microservices. As with increasing features in our applications, we need to create a separate microservice for each new feature and doing entire setup for this including adding all dependencies will take time. So, in this scenario, Spring Boot is required.

In a monolithic application, we just do setup and dependency addition only once, so in monolithic application, we don't need spring boot.

2.What are the steps to use Spring-boot-starter-caching ?

Steps for using spring-boot-starter-cache are following:

1). Add dependency:

```
  <dependency>
     <groupId> org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-cache </artifactId>
  </dependency>
```

2). Add @EnableCaching annotation on main Spring Boot Application class.

The @EnableCaching triggers a post-processor that inspects every spring bean for the presence of caching annotations on public methods. If such an annotation is found, a proxy is automatically created to intercept the method call and handle the caching behavior accordingly.

The post-processor handles the @Cacheable, @CachePut and @CacheEvict annotations.

Spring boot automatically configures a suitable CacheManager to serve as a provider for the relevant cache.

3). Put @Cacheable and other annotations on methods.

e.g.:

```
public interface BookRepository{

   Book getByIsbn(String isbn);
}
```

Use our own implementation if we are not using Spring Data.

```
public class SimpleBookRepository implements BookRepository{

   @Override
   @Cacheable("books")
   public Book getByIsbn(String isbn){

      return new Book(isbn);
   }

}
```

3.What happens when we call SpringApplication.run() method in main class of SpringBoot

application?

Syntax of the class containing main method looks like code below:

```
@SpringBootApplication
public class StudentApplication{

   public static void main(String[] args){

       SpringApplication.run(StudentAppplication.class, args);
   }
}
```

When we run this class as a java application, then our application gets started.

SpringApplication.run() is a static method and it returns an object of ConfigurableApplicationContext.

ConfiguravleApplicationContext ctx = SpringApplication.run(StudentApplication.class, args);

Thus, Spring container gets started once run() method gets called.

Spring container once started is responsible for:

- Creating all objects: This is done by @ComponentScan. Remember @SpringBootApplication is a combination of @ComponentScan + @Configuration + @EnableAutoConfiguration
- Dependency injection
- Managing the lifecycle of all beans

Steps executed under run() method are as follows:

- Application Context is started.
- Using application context, auto discovery occurs: @ComponentScan
- All default configurations are setup.
- An embedded servlet container is started e.g. Tomcat. No need to setup a separate web server. **Note**: Embedded servlet container is launched only if the web is mentioned in a dependency.

4.What are Spring Boot Actuator endpoints?

Spring Boot Actuator is used to monitor and manage application usages in production environment without coding and configuration for any of them.

This monitoring and managing information is exposed via REST like endpoint URL's.

Actuator Endpoints:

/env : Returns list of properties in current environment

/health : Returns application health information.

/beans : Returns a complete list of all the Spring beans in the application.

/trace : Returns trace logs [By default the last 100 HTTP requests]

/dump : It performs a thread dump

/metrics : It shows several useful metrics information like JVM memory used, CPU usage , open files and much more.

Note: These endpoints can be explicitly enabled/disabled.

If we need to list all endpoints in the browser:
Just run the application and from browser , use localhost:8080/actuator. It will list 2 endpoints by default : info and health

For other endpoints, we need to expose them manually.

Now, if we want to explore health endpoint, then just open localhost:8080/actuator/health URL and it will just display "status" : "UP"

We can enable or disable the endpoint from application.properties file:

management.endpoint.shutdown.enabled = true

By default all the endpoints are enabled except shutdown endpoint.

5.By default on which port Spring Boot Actuator runs?

On port 8080

We can override that setting by adding application.properties file.

6.Difference between Spring 2 and Spring 5?

Below are the differences between Spring 2 and Spring 5 versions:

- JDK Baseline update
- Core framework revision
- Reactive programming model
- Core Container updates
- Testing improvements

7.What is the difference between @SpringBootApplication and @EnableAutoConfiguration?

Following are the differences between @SpringBootApplication and @EnableAutoConfiguration:

Availability : SpringBootApplication was introduced in version 1.2 , While EnableAutoConfiguration was introduced in version 1.0

Purpose : @EnableAutoConfiguration enables auto configuration feature of Spring Boot application which automatically configures things if certain classes are present in classpath e.g. : it can configure Thymeleaf TemplateResolver and ViewResolver if Thymeleaf is present in the classpath.

On the other hand , **@SpringBootApplication does three things**.

- It allows us to run the main class as a jar with an embedded container [Web server Tomcat].
- It enables java configuration.
- It enables component scanning.

Java Technical Architect interview @ Tech Mahindra
1.What are Spring boot actuator endpoints?

Spring boot actuator is used to monitor and manage application usage in production environment without coding and configuration for any of them.

This monitoring and managing information is exposed via REST like endpoint URL's.

Actuator endpoints:

/env : Returns list of properties of current environment.

/health : Returns application health information.

/beans : Returns a complete list of all the spring beans in the application.

/trace : Returns trace logs [By default last 100 HTTP requests]

/dump : It performs a thread dump.

/metrics : It shows several useful metrics information like JVM memory usage, CPU usage, open files and much more.

Note: These endpoints can be explicitly enabled/disabled.

If we need to list all endpoints in the browser:
Just run the application and from the browser, use localhost:8080/actuator. It will list 2 endpoints by default : info and health.

For other endpoints, we need to expose them manually.

Now, if we want to explore health endpoint, then just open localhost:8080/actuator/health URL and it will just display "status" : "UP"

We can enable or disable the endpoint from application.properties file:

management.endpoint.shutdown.enabled = true

By default , all the endpoints are enabled except shutdown endpoint.

2.What is the difference between map and flatmap in java 8?

map() method is used to map an object or entry from stream to some other value.
flatMap() method on the other hand applies map() on the entries and then flatten the result.

e.g.:

Suppose we have a string array with entries:
String[] strArray = {"12", "46"};

And we have to find all permutations of these strings.
So output with map() method will be: [12, 21] , [46, 64]

While the output with flatMap() method will be : [12, 21, 46, 64]

Lets take another example:

```
List<Integer> evens = Arrays.asList(2,4,6,8);
List<Integer> odds = Arrays.asList(1,3,5,7);
List<Integer> primes = Arrays.asList(5,7,11,13);

List numbers = Stream.of(evens, odds, primes).flatMap(list-> list.stream()).collect(Collectors.toList());

System.out.println("List = "+numbers);
```

Note: this output is possible only with flatMap() method.

3.Explain Spring Data JPA versus Hibernate?

JPA : Java Persistence API provides specification for creation, deletion, persistence and data management from java objects to relations [tables] in database.

Hibernate : There are various providers which implement JPA. Hibernate is one of them. So , we have other providers as well e.g.: Eclipse Link.

Spring Data JPA: This is another layer on top of JPA which spring provides to make coding easier.

4.When to use @Component and when to use @Bean?

@Component is a class level annotation while @Bean is a method level annotation.

@Component is preferable for component scanning and automatic wiring.
Sometimes automatic configuratin is not an option. When? Let's imagine that we want to wire components from 3rd party libraries[where we don't have the source code ,so we can't annotate the class with @Component], so automatic configuration is not possible.

So, in this case , we should use @Bean.

The @Bean annotation returns an object that spring should register as bean in application context/Spring IOC container. The body of the method bears the logic responsible for creating the instance.

Another difference is that:

@Component does not decouple the declaration of bean from the class definition whereas @Bean decouples the declaration of the bean from the class definition.

There are two ways to create beans:

One is to create a class with an annotation @Component. The other is to create a method and annotate it with @Bean

@Bean requires the @Configuration annotation on the class in which @Bean is used. Whereas @Component doesn't require the class to be annotated with @Configuration.

Once we run the spring project, the class with @ComponentScan annotation would scan every class with @Component on it and store the instance of this class to the IOC container.

Another thing , the @ComponentScan would do is , running the methods with @Bean on it and store the returned object to the IOC container as a bean.

5.Explain the use case with example where @Bean is more beneficial to use than @Component?

Lets suppose, we want some specific implementation depending on some dynamic state. @Bean is perfect for that case.

```
@Bean
@Scope("prototype")
public Service anyService(){

   switch(state){
      case 1:
         return new Impl1();

      case 2:
         return new Impl2();

      case 3:
         return new Impl3();

      default:
         return new Impl();

   }
}
```

In the above case, only @Bean can be used , not the @Component.

Note: We can use both @Component and @Configuration annotations in our same application/module.

6.Explain the difference between @Service and @Component in Spring.

@Component is a generic for other stereotypes.
So, we can replace @Component with @Service, @Repository or @Controller and nothing will change.
But for better readability , we should use @Repository, @Service and @Controller.

They all inform spring that the class is involved in the DI context.

In order to configure spring so that it can provide us with the instances of the class we need , we are supposed to tell spring what objects are involved and how they are built. To do this, we can use XML configuration file or java annotations.

They also have semantic meaning:

- @Controller : @Component belonging to presentation layer
- @Service : @Component belonging to Service/Use case layer
- @Repository : @Component belonging to Persistence layer

Interview @ SirionLabs

1.What is the difference between Saga and 2 Phase Commit patterns?

Saga and 2 Phase Commit patterns are used for distributed transactions.
A distributed transaction would do what a transaction would do but on multiple databases.

Differences:

- 2 Phase Commit is for immediate transactions. Saga pattern is for long running transactions.
- In 2 Phase Commit, there is a single controlling node and if it goes down, then entire system fails. In Saga Pattern, There are multiple coordinators: Saga Execution Coordinator.
- 2 Phase Commit works in 2 steps: Prepare and Commit. And the result of all the transactions are seen to the outer world at once together. But in Saga, there is only one step for execution and commit, but if any transaction fails, then compensatory transaction gets executed for rollback operation.

2.Which database should be used when storing billions of payment requests come to database and why?
In this case, data is relational in nature like payment amount, user id, payment date, payment location, account number etc.

So, I can easily use SQL Server 2016 with proper indexes on proper fields.

3.Which Event Bus is used by Saga pattern?

When we use event-based communication, a microservice publish an event when something notable happens such as when it updates a business entity. Other microservices subscribe to these events. When a microservice receives an event, it can update it's own business entities which might lead to more events being published.

This publish/subscribe system is usually performed by using an implementation of an event bus.
The event bus is designed as an interface with the API needed to subscribe and unsubscribe to events and to publish events.
The implementation of this event bus can be a **messaging queue like RabbitMQ or service bus like Azure service bus.**

4.Explain ACID properties of database.
A transaction is a single unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
In order to maintain consistency in a database, before and after transaction, certain properties are followed. These are called ACID properties.

Atomicity:
By this, we mean that either the entire transaction takes place at once or doesn't happen at all.

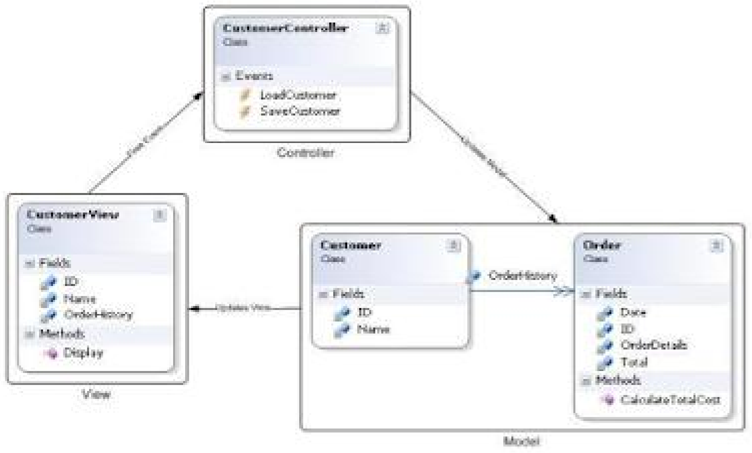
Consistency:
This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

Isolation:
This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference.
Changes occuring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

Durability:
This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if system failure occurs. The effects of the transaction , thus, are never lost.

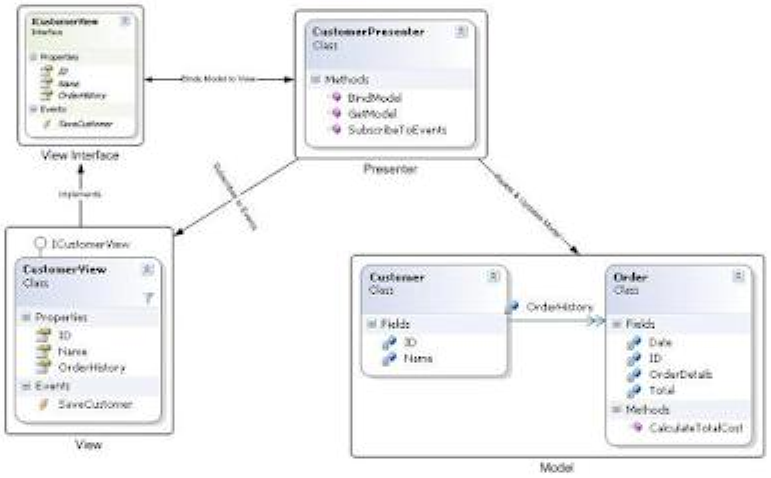
5.What is difference between MVC and MVP architectures?
MVC Architecture:

Model-View-Controller



MVP Architecture:

Model-View-Presenter



MVP is just like MVC , only difference is that Controller is replaced with Presenter.
View and Presenter are decoupled from each other using an interface. This interface is defined in Presenter .

View implements this interface and communicates with Presenter through this interface. Due to this loose coupling it is easy to mock View during unit testing.

6.What is Swagger and what are annotations used by swagger?

It is a tool for developing APIs specification with the OpenAPI specification.

Swagger vs OpenAPI:

OpenAPI : Specification
Swagger : Tool for implementing the specification

Annotations used by Swagger:

- @Api
- @ApiModel
- @ApiModelProperty
- @ApiParam
- @ApiResponse
- @ApiResponses

7.What are the steps to use Swagger?

Steps to use Swagger are defined below:

- Put dependencies :
- <dependency>
- <groupId> io.springfox</groupId>
- <artifactId>springfox-swagger2</artifactId>
- </dependency>
- Also add dependency <artifactId>springfox-swagger-ui</artifactId>
- Create a swagger config file [.java file] annotated with @EnableSwagger2 and should have accompanying @Configuration annotation. In this class, create a Docket bean by calling api() method.
- Use @Api on controller class.
- Use @ApiOperation and @ApiResponses on methods.
- Use @ApiModel on entity class.

8.What is a cloud? What are the benefits of using it?

A cloud is actually a collection of web servers [instead of a single server] owned by a 3rd party.

Cloud provides inexpensive ,efficient and flexible alternatives to computers.

Benefits of cloud:

- No need of extra space required to keep all the hardware [Servers, digital storage]
- Companies don't need to buy software or software license for all it's employees. They just pay a small fees to the cloud computing company to let their employees access a suite of software online.
- It also reduces IT problems and costs.

9.What is sharding ? When we need sharding? How to implement MongoDB sharding?

Sharding involves breaking up of one data set into multiple smaller chunks, called logical shards. The logical shards can then be distributed across separate database nodes,referred to as physical shards , which can hold multiple logical shards.

Sharding is a method of splitting and storing a single logical dataset in multiple databases.

Sharding adds more servers to database and automatically balances data and load across various servers. These databases are called shards.

When and why we need sharding?

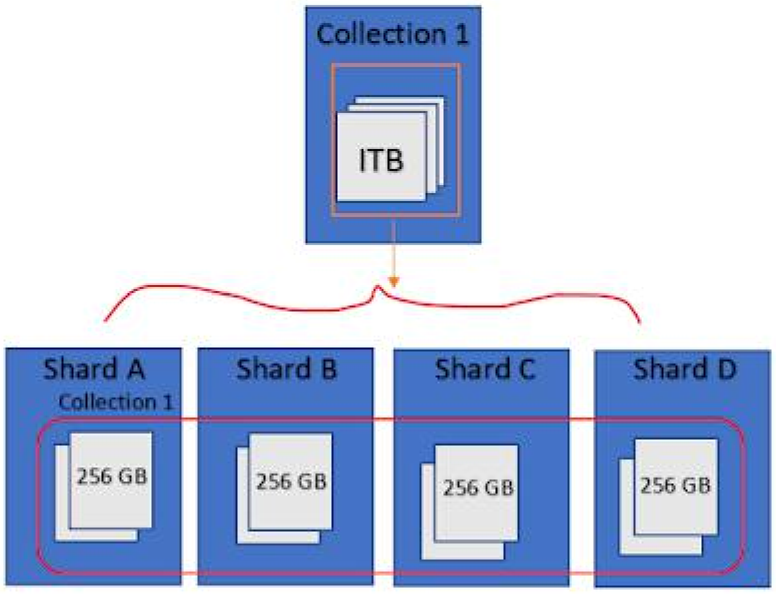
- When the dataset outgrows the storage capacity of a single database instance.
- When a single instance of DB is unable to manage write operations.

How to implement MongoDB sharding?

When deploying sharding, we need to choose a key from a collection and split the data using the key's value.

Task that the key performs:

- Determines document distribution among the different shards in a cluster.



Choosing the correct shard key:

To enhance and optimize the performance , functioning and capability of the DB, we need to choose the correct shard key.

Choosing correct shard key depends upon two factors:

- The schema of the data.

- The way database applications query and perform write operations.

10.How an application is deployed to Amazon EC2 ?

Amazon EC2 offers ability to run applications on cloud.

Steps to deploy application on EC2:

- Launch an EC2 instance and SSH into it. This instance needs to be created first on AWS Console [console.aws.amazon.com]. And we should have certificate to connect to EC2 instance.
- Install Node on EC2, if our app is in angular.
- Copy paste code on EC2 and install dependencies.
- Start server to run.

Another way:
Suppose we have Spring Boot application.

- Build out Spring Boot app in our local computer. Make .jar file.
- Upload this .jar file on S3.
- Create EC2 instance.
- SSH into EC2 from our local computer.
- Now, we are in EC2 instance.
- Now install JDK.
- And using java - .jar file path, we can run our application.

11.What is serverless Architecture?

ServerLess Architecture means: Focus on the application, not the infrastructure.

Serverless is a cloud computing execution model where the cloud provider dynamically manages the allocation and provisioning of servers.

A serverless application runs in a stateless container that are event-triggered and fully managed by the cloud provider.

Pricing is based on the number of executions rather than pre-purchased compute capacity.

Serverless computing is an execution model where the cloud provider is responsible for executing a piece of code by dynamically allocating the resources.
And only charging for the amount of the resources used to run the code. The code typically runs inside stateless compute containers that can be triggered by a variety of events including HTTP requests, database events, file uploads, scheduled events etc.

The code that is sent to the cloud provider is in the form of a function. Hence serverless is sometimes also referred to as "Functions as a Service".

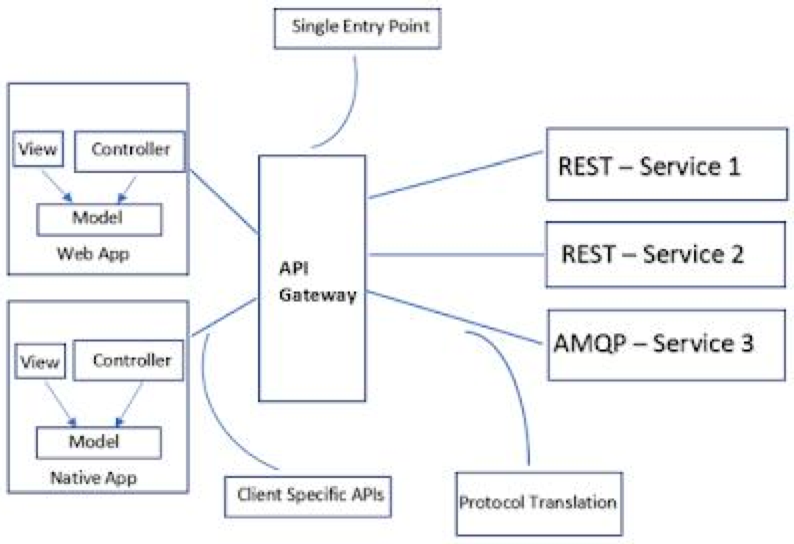
Following are the FaaS offerings of the major cloud providers:

- AWS Lambdas
- Microsoft Azure : Azure Functions
- Google Cloud : Cloud Functions
- IBM Openwhishk

- Auth0 Webtask

Interview questions on Microservices
1.What is the API Gateway pattern?
API Gateway is the entry point for all clients. The API Gateway handles requests in one of two ways:

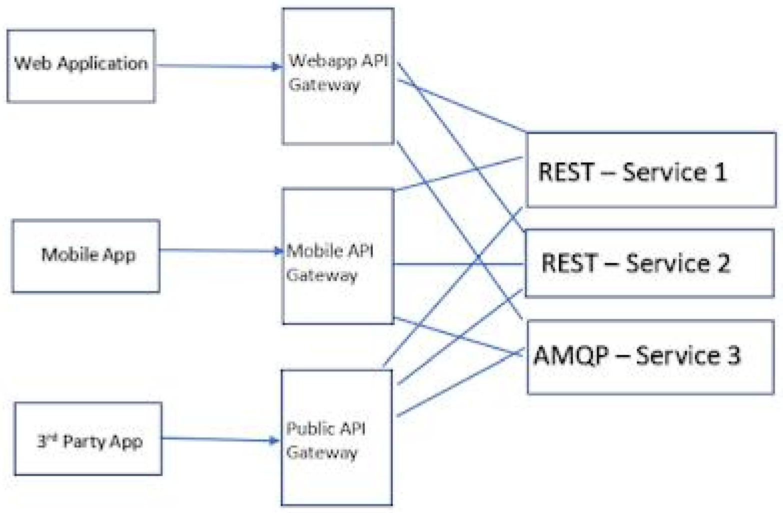
- Some requests are simply proxied/routed to the appropriate service.
- It handles other requests by fanning out to multiple services.



Rather than provide a one-size-fits-all style API, the API gateway can expose a different API for each client.

Variation : Backends for Frontends:

A variation of API Gateway pattern is the Backends for Frontends.
It defines a separate API Gateway for each kind of client.



In this example, there are 3 different types of APIs for 3 different types of clients.

Benefits of using API Gateway Pattern:

- Separates the clients from how the application is divided into microservices.
- Frees the clients from determining the location of microservices.
- Providing the optimal API for each client.
- It reduces the number of requests/round trips. e.g. : The API gateway enables the clients to retrieve data from multiple services with a single round-trip. Fewer round-trips means less requests from client and less overhead.
- Simplifies the client by moving the logic for calling multiple services from client to API Gateway.

Limitations of using API Gateway Pattern:

- Increased complexity: API Gateway needs to be developed, deployed and managed.
- Increased response time due to the additional network hop through the API Gateway.

Security using API Gateway's Access Token Pattern:

Problem: How to implement the identity of the requestor to the services that handle the request?

Solution:

The API Gateway authenticates the request and passes an access token [e.g. JSON Web Token] that securely identifies the requestor in each request to the services. A service can include the access token in requests it makes to other services.

This pattern has following benefits:

- The identity of the requestor is securely passed around the system.
- Services can verify that the requestor is authorized to perform the operation.

2. Client Side Discovery/Server Side Discovery

As, we know, services need to call one another. In a monolithic application, services invoke each other through language-level method or procedure calls.

In a traditional distributed system deployment , services run at fixed, well known locations, so can easily call one another using REST API or some RPC mechanism.

However a microservice-based application runs in a virtualized or containerized environments where the number of instances of a service and its locations change dynamically.

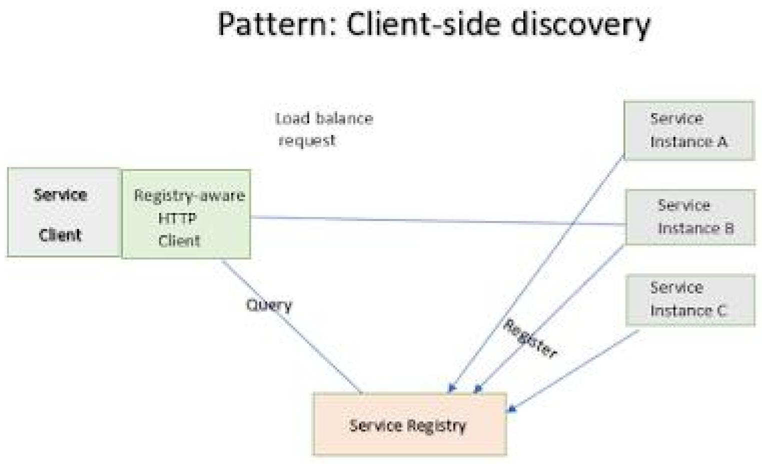
So, the **problem** is:
How does the client service - API Gateway or another service - discover the location of a service instance?

Forces:

- Each instance of a service exposes a remote API such as HTTP/REST at a particular location.
- The number of service instances and their location change dynamically.
- Virtual machines and containers are usually assigned dynamic IP addresses.

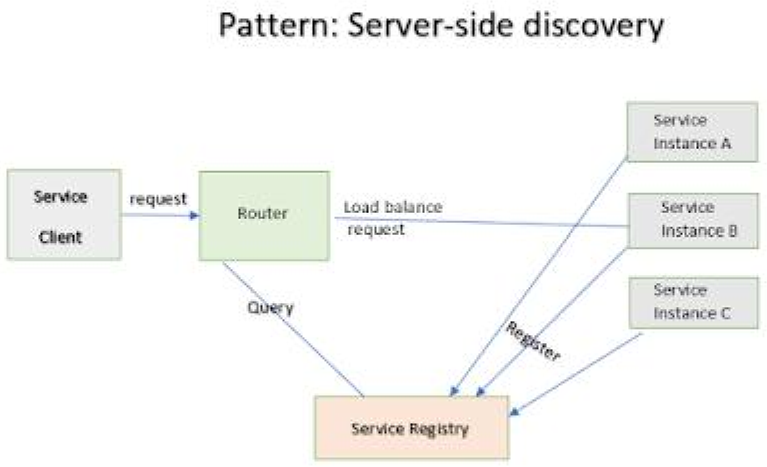
Solution:

Client Side Service Discovery:
When making a request to a service, the client obtains the location of the service instance by querying a service registry, which knows the locations of all service instances.



So, in Client Side Service discovery, Client service - API Gateway send request to Service registry to find a service and then load balance the request to any instance of that service.

Server Side Service Discovery:



In server side service discovery, client service - API Gateway sends request to a Router a.k.a. Load Balancer which in turn send a query to service registry ,finds the service and propagates request to appropriate instance.

Benefits of server side service discovery:
- Compared to client side discovery, the client code is simpler since it does not have to deal with discovery. Instead, a client just makes a request to the router.
- Some cloud environments provide this functionality , e.g.: AWS Elastic Load Balancer.

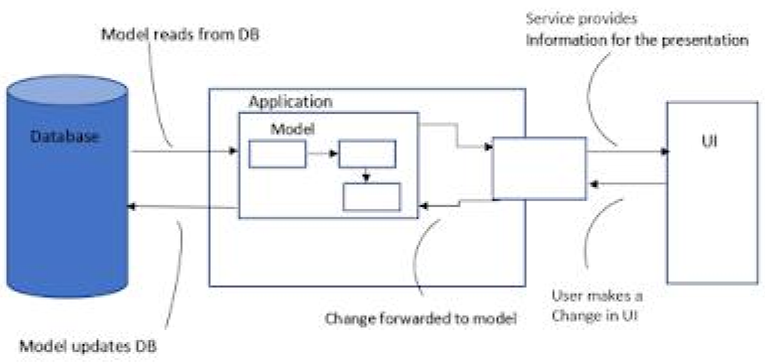
3.CQRS

CQRS : Command Query Responsibility Segregation

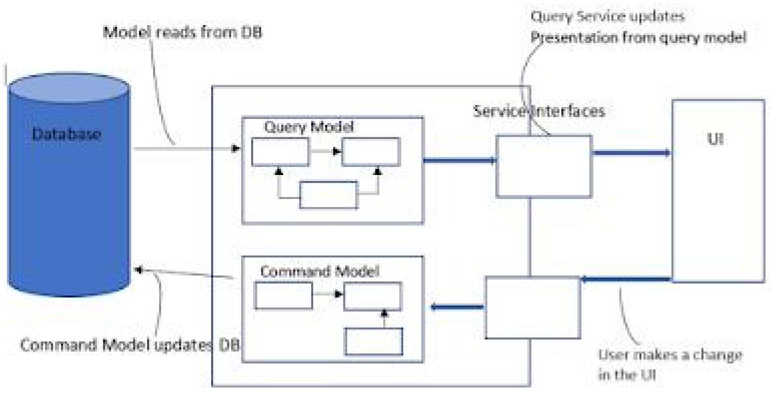
This pattern says, we can use a different model to update information than the model we use to read information.

For some situations, this separation can be valuable, but for more systems, CQRS adds risky complexity.

Non-CQRS Model:



CQRS Model :



In this we have used different models for read (Query) and update (Command) operations.

By different models we mean different object models, probably running in different logical processes, perhaps on separate hardware.

A web example would see a user looking at a web page that's rendered using the Query model. If the user initiate a change, that change is routed to the separate command model for processing, the resulting change is communicated to the query model to render the updated state.

In this case, there is room for considerable variation here:

The in-memory models may share the same database, in which case, the database acts as the communication between the two models. However, they may also use separate databases, effectively making the query-side database into a real time Reporting database.

In this case, there needs to be some communication mechanism between the two models of their databases.

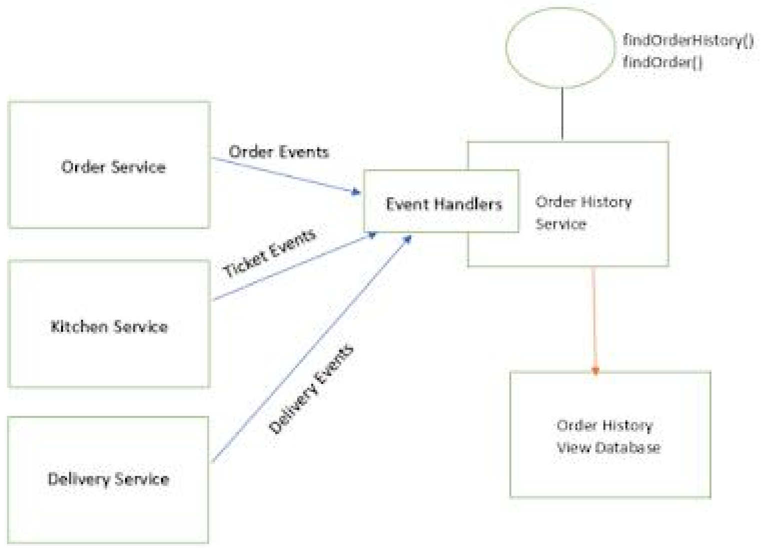
The two models might not be separate object models, it could be that the same objects have different interfaces for their command side and their query side.

CQRS naturally fits with some other architectural patterns:

- As we move away from a single representation that we interact with via CRUD, we can easily move to a task-based UI.
- CQRS fits well with event-based programming models. It's common to see CQRS system split into separate services communicating with event collaboration. This allows these services to take advantage of Event Sourcing.
- Having separate models raises questions about how hard to keep those models consistent, which raises the likelihood of using eventual consistency.

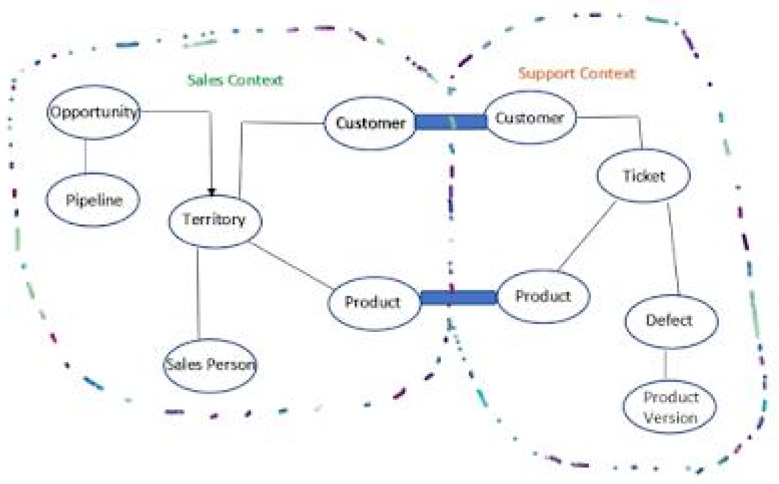
It also solves a problem:
How to implement a query that retrieves data from multiple services in a microservice architecture?

Solution:
Define a view database that is a read-only replica which is designed to support that query.
The application keeps the replica up-to-date by subscribing to Domain events published by the service that own the data.



When to use CQRS:
In particular, CQRS should only be used on specific portions of a system (a Bounded Context in DDD lingo) and not the system as a whole.

BoundedContext:
Bounded Context is a central pattern in Domain-Driven Design.
Domain Driven Design deals with large models by dividing them into different bounded contexts and being explicit about their relationships.
e.g. of Bounded Context:



4.Circuit Breaker

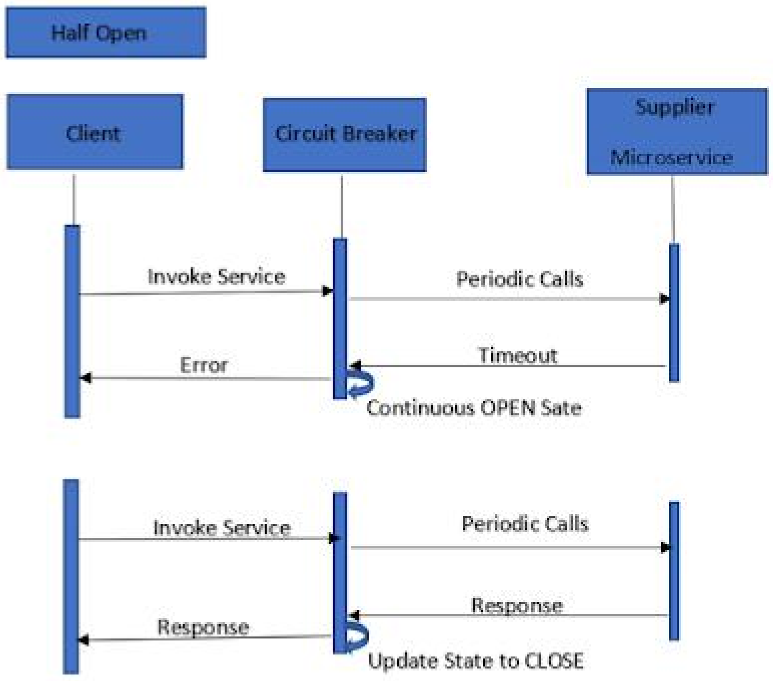
It is very important to prepare system in case of partial failure, especially for a microservice-based architectures, where there are many applications running in separate processes.

A single request from the client point of view might be forwarded through many different services and it is possible that one of thee services is down because of a failure, maintenance or just might be overloaded , which causes an extremely slow response to client requests coming into the system.

There are several best practices for dealing with failures and errors:

- The first practice recommends that we should always set network connect and read timeouts to avoid waiting too long for the response.
- The second approach is about limiting the number of accepted requests if a service fails or responses take too long. In this case, there is no sense in sending additional requests by the client.

The last two patterns are closely connected to each other. Here I'm thinking about the circuit breaker pattern and fallback.



Few points about the Circuit breaker diagram:

- Initially the circuit is closed. Means the service is working fine , accepting requests and returning response.
- When few requests get failed, circuit gets tripped and enter into Open state and doesn't allow any further request to go to that non-responding service.
- In between, after some timeout , circuit breaker allows few requests to the non-responding service to check the status of that service. This state is called Half-open state as it allows only few requests to go further. If service responds well, then circuit gets into Closed state , else it will get into Open state again.

Circuit Breaker Pattern:

The major assumption of this approach relies on monitoring successful and failed requests. If too many requests fail or services take too long to respond, the configured circuit breaker is tripped and all further requests are rejected.

The most popular implementation of Circuit Breaker pattern is available in Netflix Hystrix which is used by many java frameworks like Spring Cloud or Apache Camel.

Implementation of a circuit breaker with Spring Cloud Netflix is quite simple.
In the main class, it can be **enabled with one annotation:**

```
@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
public class Application{

  public static void main(String[] args){
    SpringApplication.run(Application.class, args);
  }
}
```

And also need to add dependency :

In gradle: Use :

```
compile group: 'org.springframework.cloud', name: 'spring-cloud-starter-hystrix', version: '1.4.7.RELEASE'
```

For fallback mechanism, we can use FeignClient like:

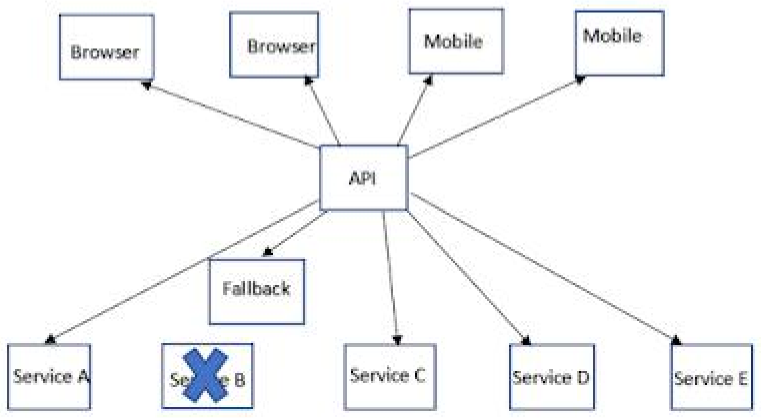
Our service A is going to interact with service B , but circuit breaker is tripped and requests to service B are rejected.

So , we need to call any fallback method for providing the response.

A.java:

```
@FeignClient(name = "A",  url = " URL or endpoint of service B" , fallback = A.Fallback.class)
public interface A{

   @GetMapping(value = "/{id}?app_id={app_id}")
   ResponseEntity<String> getInfo(@PathVariable("id") String id);


   class Fallback implements A{

      @Override
      public ResponseEntity<String> getInfo(String id){

         // Send back any info.
      }

   }


}
```

A pictorial diagram is like:

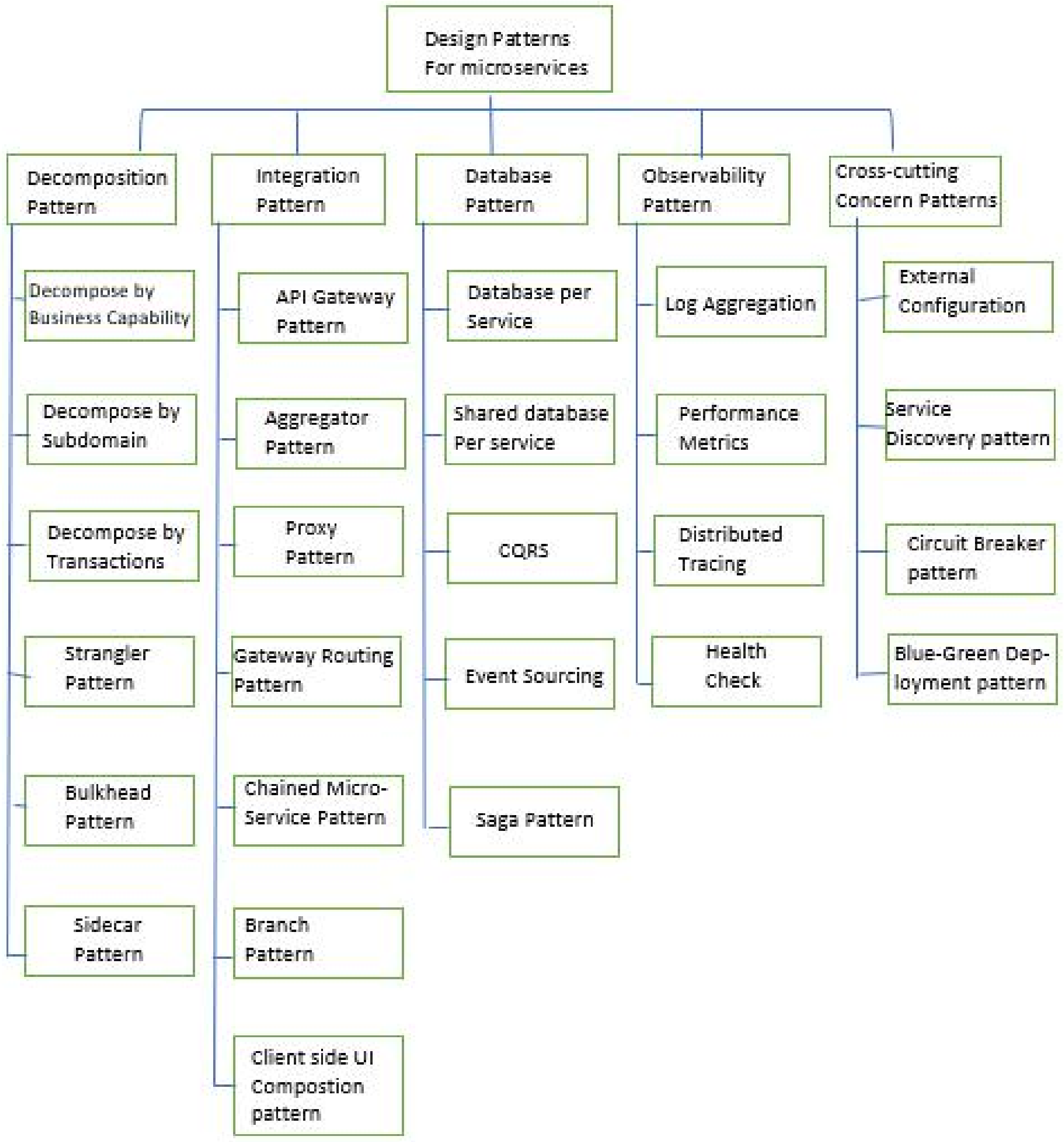


5.What principles microservice architectures has been built upon?

- Scalability
- Availability
- Resiliency
- Flexibility
- Independent, autonomous
- Decentralized governance
- Failure Isolation
- Auto-Provisioning
- Continuous delivery through DevOps

Adhering to the above principles brings several challenges and issues while bring our system to live. These challenges can be overcome by using correct and matching design pattern.

6.What types of design patterns are there for microservices?



7.Explain in detail Decomposition patterns.

- **Decompose by Business capability**: This pattern is all about making services loosely coupled and applying the single responsibility principle. It decomposes by business capabilities. Means define services corresponding to business capability. It is something that a business does in order to generate values. A business capability often corresponds to a business object, e.g.:
- Order management is responsible for orders
- Customer management is responsible for customers
- **Decompose by Subdomain**: This pattern is all about defining services corresponding to Domain-Driven-Design [DDD] subdomains. DDD refers to the application's problem space - the business - as the domain. A domain consists of multiple subdomains. Each subdomain corresponds to a different part of the business. Subdomains can be classified as follows:

- Core
- Supporting
- Generic

The subdomains of Order management consists of:

- product catalog service
- Inventory management service
- Order management services
- Delivery management services

- **Decompose by transactions/Two-Phase commit pattern** : This pattern can decompose services over transactions. There will be multiple transactions in a system. One of the important participants in a distributed transaction is the transaction coordinator. The distributed transaction consists of two steps:
- Prepare phase
- Commit phase
- **Sidecar Pattern**: This pattern deploys components of an application into a separate processor container to provide isolation and encapsulation. This pattern can also enable applications to be composed of heterogeneous components and technologies. It is called sidecar as it resembles to a sidecar of a bike. In this pattern, a sidecar is attached to an application and provides supporting features for the application.

8.Explain in detail Integration patterns.

- **API Gateway Pattern**: When an application is broken down into multiple microservices, there are a few concerns that need to be considered:
- There are multiple calls for different microservices from different channels
- There is a need for handling different type of protocols
- Different consumer might need a different format of the responses.

An API gateway helps to address many of the concerns raised by the microservice implementation, not limited to the ones above.

- An API gateway is the single point of entry for any microservice calls.
- It can work as a proxy service to route a request to the concerned microservice.
- It can aggregate results to send back to the user.
- This solution can create a fine-grained API for each specific type of client.
- It can also offload the authentication/authorization responsibility of the microservice.
- **Aggregator Pattern**: This pattern helps aggregate the data from different services and then send the final response to the client. This can be done in two ways:
- A composite microservice will make calls to all the required microservices, consolidate the data and transform the data before sending back.
- An API gateway can also partition the request to multiple microservices and aggregate the data before sending it to the consumer. An API gateway can have different modules:
- Mobile API
- Browser API
- Public API

9.Explain in detail the database patterns, you have worked upon.

To define the database architecture for microservices, we need to consider the below points:

- Services must be loosely coupled.They can be developed, deployed and scaled independently.
- Business transactions may enforce invariants that span multiple services.
- Some business transactions need to query data that is owned by multiple services.
- Databases must be sometimes replicated and shared in order to scale.
- Different services have different data storage requirements.
- **Database per service**: To solve the above concerns, one database per service must be designed. It must be private to that service only. It should be accessed by the microservice API only. It cannot be accessed by other services directly. e.g.: for relational database, we can use private-tables-per-service, schema-per-service or database-server-per-service.
- **Shared database per service**: This pattern is useful when we have an application which is monolith and we try to break it into microservices.
- **Command Query Responsibility Segregation [CQRS]**: Once we implement database per service, there is a requirement to query, which requires joint data from multiple services, it's not possible. CQRS suggests splitting the application into two parts: the command side and the query side.
- The command side handles the create, update and delete requests.
- The query side handles the query part by using the materialized views.

The event sourcing pattern is generally used along with it to create events for any data change. Materialized views are kept updated by subscribing to the stream of events

- **Event Sourcing Pattern**: This pattern defines an approach to handling operations on data that is driven by a sequence of events , each of which is recorded in an append-only store. Application code sends a series of events that imperatively describe each action that has occurred on the data to the event store, where they're persisted. Each event represents a set of changes to the data.
- **Saga Pattern**: When each service has it's own database and a business transaction spans multiple services, how do we ensure data consistency across services? Each request has a compensating request that is executed when the request fails. It can be implemented in two ways:
- **Choreography**: In microservice choreography, each microservice performs their actions independently. It does not require any instructions. It is like the decentralized way of broadcasting data known as events. The service which are interested in those events , will use it and perform actions. It is like an asynchronous approach.
- **Orchestration**: In the microservice orchestration, the orchestration handles all the microservice interactions. It transmits events and responds to it. The microservice orchestration is more like a centralized service. It calls one service and waits for the response before calling the next service. This follows a request/response type paradigm.

Technical Architect interview in Incedo
1.What is database sharding? Why we need database sharding?

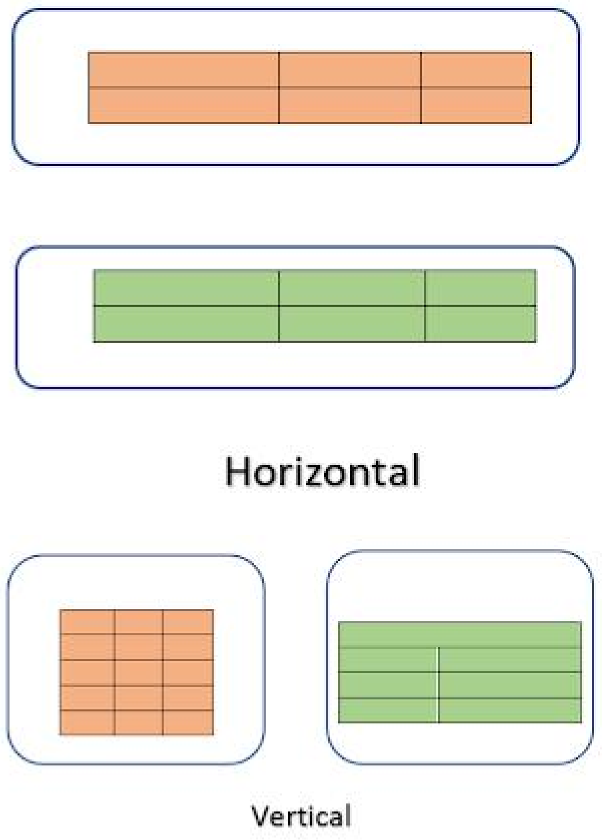
Sharding involves breaking up one's data into two or more smaller chunks, called logical shards. The logical shards are then distributed across separate **database** nodes, referred to as physical shards, which **can** hold multiple logical shards.

Sharding is a method of splitting and storing a single logical dataset in multiple databases. Sharding adds more servers to a database and automatically balances data and load across various servers. These databases are called shards.

Sharding is also referred as **horizontal partitioning**. The distinction of **horizontal** vs **vertical** comes from the traditional tabular view of a database.

2.What are the types of database sharding?

A database can be split vertically — storing different tables & columns in a separate database, or horizontally — storing rows of a same table in multiple database nodes.



Example of Vertical partitioning:

fetch_user_data(user_id) -> db["USER"].fetch(user_id)
fetch_photo(photo_id) -> db["PHOTO"].fetch(photo_id)

Example of Horizontal partitioning:

fetch_user_data(user_id) -> user_db[user_id % 2].fetch(user_id)
Vertical sharding is implemented at application level – A piece of code routing reads and writes to a designated database.
Natively sharded DB's are : Cassandra, MongoDB

Non-sharded DB's are : Sqlite, Memcached etc.

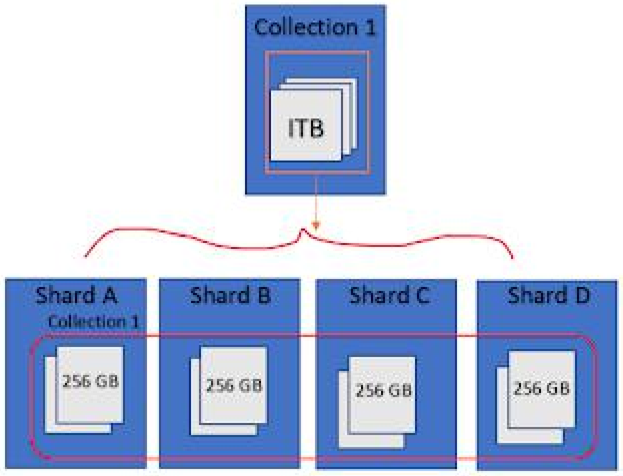
When we need to do sharding?

- When the data set outgrows the storage capacity of a single MongoDB instance.
- When a single MongoDB instance is unable to manage write operations.

3.How to implement sharding in MongoDB?
When deploying sharding, we need to choose a key from a collection and split the data using the key's value.

Task that the key performs:

- Determines document distribution among the different shards in a cluster.



Choosing the correct shard key:

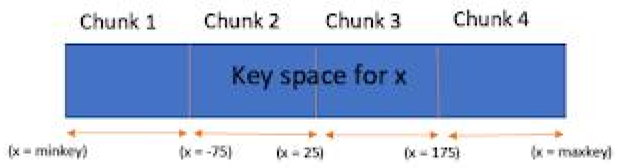
To enhance and optimize the performance, functioning and capability of the DB, we need to choose the correct shard key.

Choosing correct shard key depends on two factors:

- The schema of the data
- The way database applications query and perform write operations.

Using range-based Shard key:

In range-based sharding, MongoDB divides data sets into different ranges based on the values of shard keys. In range-based sharding, documents having "close" shard key values reside in the same chunk and shard.



Data distribution in range-based partitioning can be uneven, which may negate some benefits of sharding.

For example, if a shard key field size increases linearly, such as time, then all requests for a given time range will map to the same chunk and shard. In such cases, a small set of shards may receive most of the requests and system would fail to scale.

Hash-based sharding:

In this , MongoDB first calculates the hash of a field's value and then creates chunks using those values. In this sharding, collections in a cluster are randomly distributed.

No real schema is enforced:

- We can have different fields in every document if we want to.
- No single key as in other databases:

o But we can create indices on any fields we want, or even combinations of fields.
o If we want to shard, then we must do so on some index.

4.What is the use of Amazon EC2? What are the steps to deploy on EC2?

Amazon EC2 : **Amazon Elastic Compute Cloud**

It offers ability to run applications on the public cloud.

It eliminates investment for hardware. There is no need to maintain the hardware. We can use EC2 to launch as many virtual servers as we need.

Steps to deploy on EC2:

· Launch an EC2 instance and SSH into it. **Note**: This instance needs to be created first on AWS console[console.aws.amazon.com]. And we should also have certificate to connect to EC2 instance.

· Install Node on EC2 instance, if our app is in Angular.

· Copy paste code on EC2 instance and install dependencies.

· Start server to run.

OR:

- Build out Spring boot app in our own computer. Make .jar file.
- Upload this .jar on S3
- Create EC2 instance.
- SSH into it from our computer.
- Now, we are in EC2 instance.

- We can install JDK now.
- And using java - .jar file path, we can run our application.

5.How will we use Amazon S3?
For using S3, we need to first choose S3 from AWS console and then we need to create a bucket in that for storing our files.

After creating the bucket, we need to click on the upload option and select jar file or any file from our computer and upload it to S3.

While uploading file to S3, we need to provide it some access level so that we can download it from S3 to EC2 instance.

So before that we need to make EC2 instance up and running.
And then we need to connect from local to EC2 instance using private key that we have. For connecting to EC2, we need to do SSH login [from terminal] using some certificate and using some SSH command.

Now we need to install java on EC2 [As , it is not there by default]. After that we can download our jar file from S3 to EC2 instance.

Now just run this jar using java -jar <filename> to run the springboot application.

6.What are the best Code Review Practices?

Clean Code

- Use intention-revealing names
- Use Solution-Problem domain names
- Classes should be small
- Functions should be small
- Functions should do one thing
- Don't repeat yourself(Avoid duplication)
- Explain yourself in code : Comments
- Use Exceptions rather than return codes
- Don't return Null

Security

- Make class final if not being used for inheritance
- Avoid duplication of code
- Minimize the accessibility of classes and members
- Document security related information
- Input into a system should be checked for valid data size and range
- Release resources (Streams, Connections) in all cases
- Purge sensitive information from exceptions
- Don't log highly sensitive information
- Avoid dynamic SQL, use prepared statement
- Limit the accessibility of packages, classes, interfaces, methods and fields
- Avoid exposing constructors of sensitive classes
- Avoid serialization of sensitive classes
- Only use JNI when necessary

Performance

- Avoid excessive synchronization
- Keep synchronized sections small
- Beware the performance of String concatenations
- Avoid creating unnecessary objects

General

- Don’t ignore exceptions
- Return empty arrays or collections , not nulls
- In public classes, use accessor methods not public fields
- Avoid finalizers
- Refer to objects by their interfaces
- Always override toString()
- Document thread safety
- Use marker interfaces to define types

Static Code Analysis

- Check static code analyzer report for the classes added/modified

7.What are the types of Http Error codes?

There are 5 types of Error codes:

1XX Informational:

100: Continue

2XX Success:

200 : OK
201 : Created
202 : Accepted
204 : No Content

3XX Redirection:

4XX Client Error:

400 : Bad Request
401 : Unauthorized
402 : Payment Required
403 : Forbidden
404 : Not Found

5XX Server Error:

500 : Internal Server Error
501 : Not Implemented
502 : Bad Gateway
503 : Service Unavailable [Website's server is simply not available]

8.What are the Asymptotic Notations?
To calculate running time complexity of an algorithm, we use following asymptotic notations:

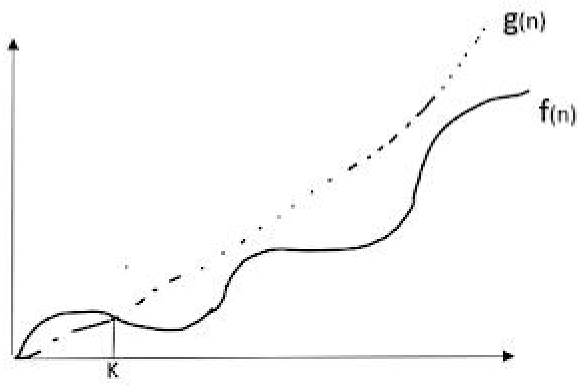
O Notation

Ω Notation

Θ Notation

Big Oh Notation O:

The notation O(n) is the formal way to express the upper bound of an algorithm's running time.
It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

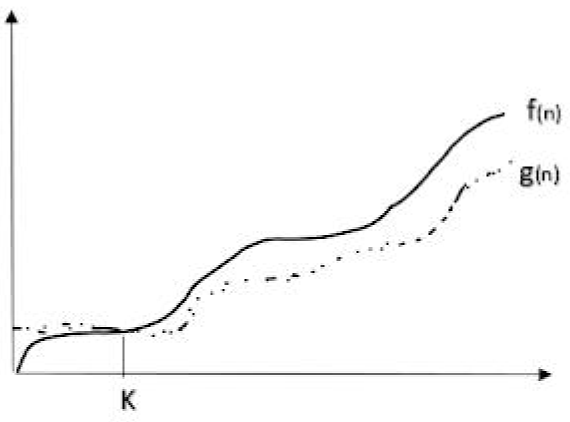


For example, for a function f(n)

O(f(n)) = { g(n) : there exists c > 0 and n0 such that f(n) <= c.g(n) for all n > n0. }

Omega Notation:

The Omega notation is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

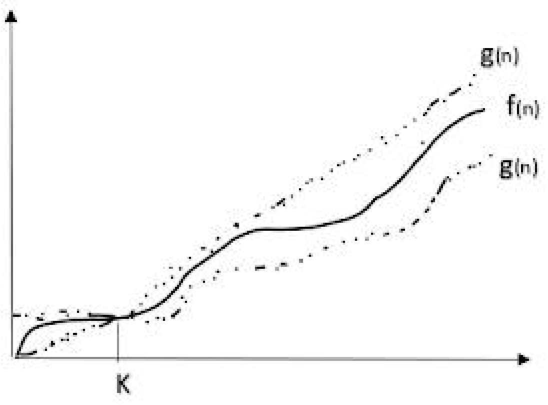


For example, for a function f(n)

```
Ω(f(n)) ≥ { g(n) : there exists c > 0 and n₀ such that g(n) ≤ c.f(n) for all n > n₀. }
```

Theta Notation:

The notation theta (n) is the formal way to express both the lower bound and upper bound of an algorithm's running time.It is represented as follows:

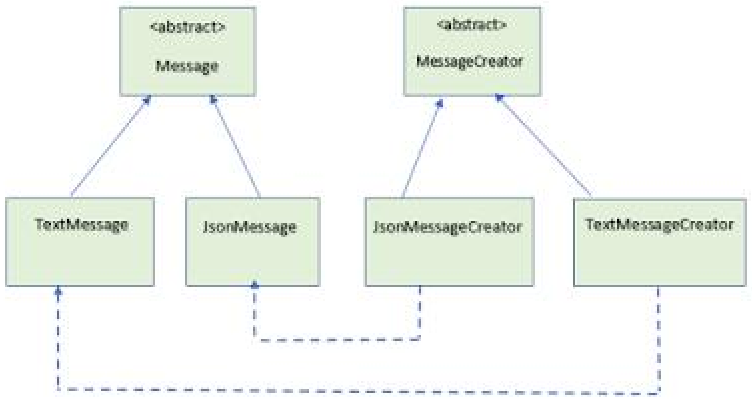


9.Explain when and why to use Factory Design Pattern.

Factory Design Pattern is used in following scenarios:

- When we need to separate the logic of object creation from our other code or client code.
- Whenever we need the ability to add more subclasses to the system and modified them without changing the client code.
- We let subclasses to decide which object to instantiate by overriding the factory method.

UML diagram for Factory Design Pattern is :



10.Explain when and why to use Abstract Factory Design Pattern.

Whenever we have to work with two or more objects which work together forming a set/kit and there can be multiple sets or kits that can be created by client code, then we will use Abstract Factory Design Pattern.

Lets take an example of a War Game:

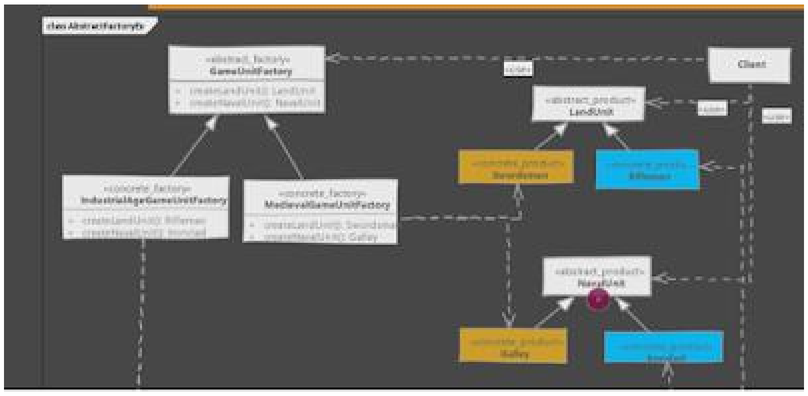
Medieval:
- Land Unit
- Naval Unit

Industrial:
- Land Unit
- Naval Unit

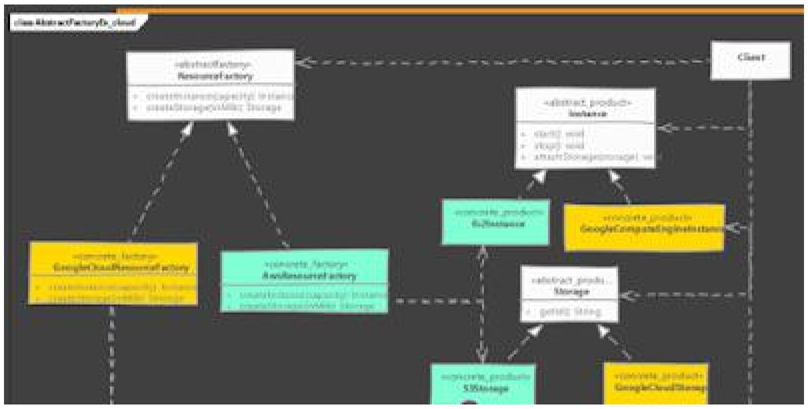
Here , we have two types of objects Land Unit and Naval Unit under Medieval age.
And also two types of objects under Industrial age.

So, we can take two factories : Medieval Age Factory and Industrial Age Factory.

UML Diagram for above scenario is :



Another example we can take is :



11.When and why to use Generic class? Explain with example. Can we use generics with array?

A **generic type** is a type with formal type parameters.

e.g.:

```
interface Collection<E>{

   public void add(E e);
   public Iterator<E> iterator();

}
```

A **parameterized type** is an instantiation of a generic type with actual type arguments.

Example of a parameterized type:

```
Collection<String> coll = new LinkedList<String>();
```

Generic class is used when we need to create similar types of object with similar features.
e.g.:

Suppose , we are developing a database library called Data Access Object (DAO) for a program that manages resources in a university.

We would write a DAO class for managing Students like:

```
public class StudentDAO{

   public void save(Student st){
        // some code here ....
   }

   public Student get(long id){

      // Some code here ...
   }

}
```

This looks fine. But if we need to persist Professor objects to database. Then we would write another

DAO class as:

```
public class ProfessorDAO{

   public void save(Professor st){
        // some code here ....
   }

   public Professor get(long id){

       // Some code here ...
   }

}
```

Note: These two classes are same. And if we need to add more entity classes to the system like course, facility etc, then?

So, to avoid such situation ,we write a Generic class:

```
public class GeneralDAO<T>{

   public void save(T entity){

   }

   public T  get(long id){

   }

}
```

Here T is called type parameter of GeneralDAO class. T stands for any type. The following code illustrates how to use this generic class:

```
GeneralDAO<Student> studentDAO = new GeneralDAO<Student>();

Student student = new Student();

studentDAO.save(student);
```

Note: Type parameters are never added in names of methods and constructors. They are only added in names of classes and interfaces.

12.What is Type Erasure?

Generics provide compile time type safety and ensures we only insert correct type in Collection and avoids ClassCastException.

Generics in java are implemented using Type Erasures.

It is a process of enforcing type constraints only at compile time and discarding the element type information at runtime.

There are two types of Type Erasures:

- Class Type Erasure
- Method Type Erasure

e.g.:

```
public static <E> boolean  containsElement(E[] elements , E element){

   for(E e : elements){

      if(e.equals(element)){
         return true;
      }
   }
   return false;
}
```

When compiled, the unbound type E gets replaced with an actual type of Object as:

```
public static  boolean  containsElement(Object[] elements , Object element){

   for(Object e : elements){

      if(e.equals(element)){
         return true;
      }
   }
   return false;
}
```

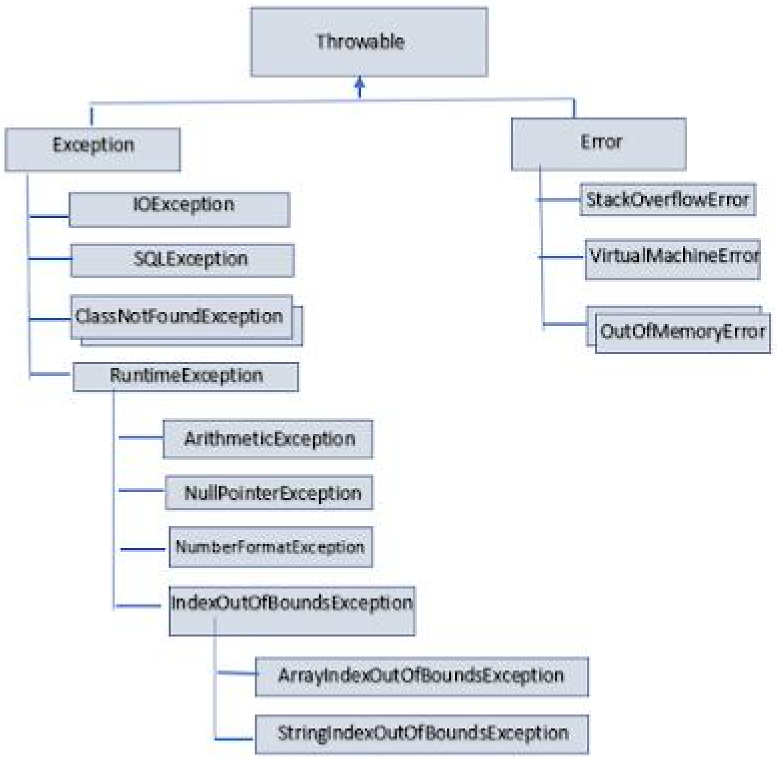
The compiler ensures type safety of our code and prevents runtime errors.

13.Explain Exception handling in java.

Types of Exceptions:
- Checked Exceptions
- Unchecked Exceptions
- Error

Checked Exception: Exceptions that are directly inherit from Throwable class except RuntimeException and Error are known as checked exceptions.

Unchecked Exceptions: The classes which inherit RuntimeException are known as unchecked exception.



14.What is static import in java? How to use it?

Static import allows developers to access static members of a class directly.There is no need to qualify it by the class name.

Example:

```
import static java.lang.System.*;

class StaticImport{

    public static void main(String[] args){

        out.println("Ok"); // No need to use System.out
        out.println("Java");
    }

}
```

Output

```
Ok
Java
```

Interview questions on SQL
1.What is the difference between MySQL and SQL Server?

Difference between MySQL and SQL server are as follows:

- MySQL server is owned by Oracle while SQL server is owned and managed by Microsoft
- MySQL supports multiple languages like Perl, Scheme, Tcl, Eiffel etc than SQL server.
- MySQL supports multiple storage engines [InnoDB, MYISAM] which makes MySQL server more flexible than SQL server.
- MySQL blocks the database while backing up the data. And the data restoration is time consuming due to execution of multiple SQL statements. Unlike MySQL, SQL server doesn't block database while backing up the data.
- SQL Server is more secure than MySQL. MySQL allows database files to be accessed and manipulated by other processes at runtime. But SQL server doesn't allow any process to access or manipulate it's database files or binaries.
- MySQL doesn't allow to cancel a query mid-execution. On the other hand, SQL server allows us to cancel a query execution mid-way in the process.

2.What is a Primary key ? How is it different from Unique key?

Primary key is a column or set of columns that uniquely identifies each row in the table. There will be only 1 primary key per table.

Important points about Primary key:

- Null values are not allowed.
- It must contain unique values. Duplicates are not allowed.
- If the primary key contains multiple columns , the combination of values of these columns must be unique.
- When we define a primary key for a table, MySQL automatically creates an index named primary.

Difference between Primary key and Unique key:

- Primary can be only 1 per table. Unique can be many per table.
- Primary key doesn't allow null value. While Unique key allows null value.
- Primary key can be made foreign key in another table. While Unique can not be made foreign key in MySQL but it can be made foreign key in SQL server.
- By default primary key is clustered index and data in the database table is physically organized in the sequence of clustered index. By default, unique key is a unique non-clustered index.

3.What is a join and why to use it?
A join clause is used to access/retrieve data from multiple tables based on the relationship between the fields of the tables.
Keys play a major role when Joins are used.

4.What are different normalization forms? Explain.

There are multiple normalization forms available in SQL:

- 1NF
- 2NF
- 3NF
- BCNF: Boyce Codd Normal Form
- 4NF

Explanation of each normalized form:

1NF Rules:

- Each table cell should contain a single value.
- Each record needs to be unique

2NF Rules:

- Be in 1NF
- Single column primary key

3NF Rules:

- Be in 2NF
- Has no transitive functional dependencies

5.What is an index and types of indexes?

An index is a performance tuning method of faster accessing the records from a table. An index creates an entry for each value and hence it will be faster to retrieve data.

There are following types of indexes:

- Normal Index
- Unique Index
- Clustered Index
- Non-Clustered Index
- BitMap Index
- Composite Index
- B-Tree Index
- Function based index

Unique Index:
This index doesn't allow the field to have duplicate value if the column is unique indexed. If a primary key is defined , Unique index can be applied automatically.

Clustered Index:
This index reorders the physical order of the table and searches based on the basis of key values. Each table can only have one clustered index.

Non-Clustered Index:
This index doesn't alter the physical order of the table and maintains a logical order of the data. Each table can have many non-clustered indexes.

6.What is a subquery and what are types of subqueries?
A query within a query is called a subquery. The outer is called an main query and inner query os called a subquery. Subquery is always executed first and the result of subquery is passed to the main query.

There are two types of subqueries:

Correlated Query: It can't be considered as independent query but it can refer the column in a table listed in the FROM of the main query.

NonCorrelated Query: It can be considered as an independent query and the output of subquery are substituted in the main query.

7.What is the difference between char and varchar2?
Both char and varchar2 are used to display character datatype but varchar2 is used for character strings of variable length whereas char is used for strings of fixed length.

e.g.: char(10) can store only 10 characters and will not be able to store a string of any other length whereas varchar(10) can store any length till 10 characters.

8.What is the use of UNION clause?

UNION clause is used to remove duplicate records from the result of a query.

9.Which aggregate functions are there in SQL?

There are multiple aggregate functions in SQL:

- MIN()
- MAX()
- SUM()
- COUNT()
- FIRST()
- LAST()
- AVG()

10.What is a View in SQL and how to create and execute a View ?

A View can be defined as a virtual table that consists of rows and columns from one or more tables.

Data in the virtual table is not stored permanently.

Note: Views are stored in system tables : sys.sysobjvalues

Use case or benefits of View:

- Views can hide complexity: If we have a query that requires joining several tables or have complex logic or calculations , we can code all that logic into a view, then select from the View just like you would a table.
- Views can be used as a security mechanism: View can select certain columns and/or rows from a table (or tables) and permissions set on the view instead of the underlying tables. This allows surfacing only the data that a user needs to see.

Creating View:

```
Create View view_name AS
SELECT column1 , column2, ...
FROM table_name
```

```
WHERE condition;

Create View [Items Above Average Price] AS
SELECT ItemName, Price
From Orders
where Price > (Select AVG(Price) from Orders);
```

Note: "Items Above Average Price" is the view name.

Querying the above created View:

```
Select * from [Items Above Average Price]
```

11: What are the JDBC statement interfaces?

JDBC statement interfaces are used for accessing the database.

These are of 3 types:

1. Statement
2. PreparedStatement
3. CallableStatement

Statement: Use this for general purpose access to the database. It is useful when we are using static SQL statements at runtime. The Statement interface cannot accept parameters.

PreparedStatement: It is used when we plan to use SQL statements many times. This interface accepts input parameters at runtime.

CallableStatement: It is used when we want to access database stored procedures. This interface also accepts runtime input parameters.

Creating Statement Object:

We need to create Statement object before we can use it. We can call createStatement() method on Connection object as follows:

```
try{
   Statement st = Connection.createStatement();
}
catch(Exception e){

}
```

Now, we can call any of the methods given below to execute SQL statement:

- execute()
- executeUpdate()
- executeQuery()

Creating PreparedStatement Object:

```
try{
   String SQL = "update Student set age = ? where id = ?";
   Statement st = Connection.prepareStatement(SQL);
}
```

```
catch(Exception e){

}
```

All parameters in JDBC are represented by the ? symbol, which is known as the parameter marker. We must supply values for each parameter before executing the SQL statement.

12.What is the difference between BLOB and CLOB?

Blob and Clob are both known as LOB [Large Object Types].

BLOB: It is a variable length Binary large object string that can be upto 2GB long. Primarily intended to hold non-traditional data such as voice or mixed media.

CLOB: It is a variable length character large object string that can be upto 2GB long. A CLOB can be used to store single byte character strings or multibyte character-based data.

Following are the differences between BLOB and CLOB:

1. The full form of Blob is Binary Large Object. The full form of Clob is Character Large Object.
2. Blob is used to store large binary data. Clob is used to store large textual data.
3. Blob stores values in the form of binary streams. Clob stores values in the form of character streams.
4. Using Clob, we can store files like text files, PDF documents, word documents etc. Using Blob, we can store files like videos, images, gifs and audio files.
5. MySQL supports Blob with the following datatypes:
    1. TinyBlob
    2. Blob
    3. MediumBlob
    4. LongBlob

   MySQL supports Clob with the following datatypes:
    1. TinyText
    2. Text
    3. MediumText
    4. LongText
6. In JDBC API, it is represented by java.sql.BLOB interface. While Clob is represented by java.sql.Clob interface.

13. What is the benefit of using PreparedStatement?

PreparedStatements help us prevent SQL injections attack.

In this the query and the data are sent to the database server separately.

While using prepared statement, we first send prepared statement to the DB server and then we send the data by using execute() method.

e.g.:

```
$db-> prepare("Select * from Users where id = ?");
```

This exact query is sent to the server. Then we send the data in second request like:

$db->execute($data);

If we don't use prepared statement then, SQL injection attack can occur like:

$spoiledData = "1; DROP table Users";
$query = "Select * from Users where id=$spoiledData";

will produce a malicious sequence:

Select * from Users where id = 1; DROP table Users;

14.How to handle indexes in JDBC?

Indexes in a table are pointers to the data which speeds up the retrieval of data from table.
If we use indexes, INSERT and UPDATE statements execute slower whereas SELECT and WHERE get executed faster.

Creating an Index:

create index index_name on table_name(column_name);

Displaying the Index:

Show indexes from table_name;

Dropping the index:

drop index index_name;

Interview questions on Executor Framework
1.Why to use executor framework?

Executor framework is used to decouple command submission from command execution.

2.What is executor framework?

Executor framework gives us the ability to create and manage threads. It provides below functionalities:

- Creating Thread
- Managing Thread
- Task submission and execution

3.What interfaces are there in executor framework?

There are multiple interfaces in executor framework. Let me describe a few of them here:

Executor: A simple interface that contains a method called execute() to launch a task specified by a Runnable object.

ExecutorService: A sub-interface of Executor that adds functionality to manage the lifecycle of the tasks. It also provides a submit() method whose overloaded versions can accept a Runnable as well as a Callable object. Callable objects are similar to Runnable except that the task specified by a Callable object can also return a value.

ScheduledExecutorService: A sub-interface of ExecutorService. It adds functionality to schedule the execution of tasks.

4.What are the classes in Executor framework?

Executors: It contains factory methods for creating different kinds of executor services.

5.How to use Executor Service?

```
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

public class ExecutorExample{

   public static void main(String[] args){

       ExecutorService exeService = Executors.newSingleThreadExecutor();

       Runnable runnable = () -> {

           System.out.println("Executing thread");
       }

       exeService.submit(runnable);

   }


}
```

Note: In the above code, Executor service uses only one thread for executing tasks. If that thread is busy, then the task will wait in a queue until the thread is free to execute it.
This program never exits, because the executor service keeps listening for new tasks until we shut it down explicitly.

6.How to shutdown the executor service?

We can shutdown the executor service using 2 methods:

shutdown(): When shutdown() method is called on an executor service, it stops accepting new tasks, waits for previously submitted tasks to execute and then terminates the executor.

shutdownNow(): This method interrupts the running task and shuts down the executor immediately.

7.How to schedule a task using Executor framework?

```
ScheduledExecutorService exeService = Executors.newScheduledThreadPool(1);

Runnable task= () ->{


   System.out.println("Executing thread");
}

exeService.schedule(task, 5, TimeUnit.SECONDS);
```

exeService.shutdown();

Note: If we want to execute the task periodically, we can do so by:

exeService.scheduleAtFixedRate(task, 0, 2, TimeUnit.SECONDS);

8.Why should we use ThreadPoolExecutor, when we have Executor Framework?

Source code of Executors.newFixedThreadPool() is:

```
public static ExecutorService newFixedThreadPool(int nThreads){

    return new ThreadPoolExecutor(nThreads, nThreads, 0L, TimeUnit.MILLISECONDS,
new                                             LinkedBlockingQueue<Runnable>());

}
```

This method uses ThreadPoolExecutor class with default configuration as is seen in above code. Now there are scenarios where default configuration is not suitable say instead of LinkedBlockingQueue, a priority queue needs to be used etc.
In such cases, caller can directly work on underlying ThreadPoolExecutor by instantiating it and passing desired configuration to it.

Note: One advantage of using ThreadPoolExecutor is that we can handle RejectedExecutionException using ThreadPoolExecutor.DiscardPolicy().

9.How the number of threads contained in thread pool is determined when using ThreadPoolExecutor?

The number of threads in the thread pool is determined by these variables:

- corePoolSize
- maximumPoolSize

If less than corePoolSize threads are created in the thread pool when a task is delegated to the thread pool, then a new thread is created, even if idle threads exist in the pool.

If the internal queue of tasks is full and corePoolSize threads or more are running, but less than the maximumPoolSize threads are running, then a new thread is created to execute the task.

10.What is the difference between Fixed thread pool and cached thread pool?

Fixed thread pool allows to create a pool with fixed number of threads.
Cached thread pool creates any number of threads depending upon the tasks provided. If number of tasks passed to ExecutorService is large, then cached thread pool creates too many threads and may increase overhead of the system.

11.What is the difference between submit() and execute() methods in executor framework?

There are multiple differences between submit() and execute() methods in executor framework which are described below:

- execute() method is declared in Executor interface. While submit() method is declared in ExecutorService interface which extends Executor interface.
- execute() method can take only Runnable object as parameter. While submit() has 3 overloaded forms which accept Runnable , Callable , Runnable and result type.
- execute() method returns void. While all forms of submit() method returns Future instance.

Note: A thread can be created only with Runnable instance.

12.When result will be returned by Future.get() , when we call submit(Runnable instance)?

As we know, run() method in Runnable doesn't return anything. So, Future instance in this case will not hold any result and calling get() on Future object will return null upon successful completion.

13.How to execute a collection of tasks at once in Executor Framework?

We can call invokeAll() method from ExecutorService passing a collection of all tasks in it.

Syntax:

```
List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
```

14.Describe the purpose and use case of Fork/Join framework.

Answer:

The fork/join framework allows parallelizing recursive algorithms. The main problem with parallelizing recursive algorithms with ThreadPoolExecutor is that we may quickly run out of threads because each recursive step requires its own thread.

The Fork/Join framework entry point is ForkJoinPool class which is an implementation of ExecutorService. It implements the work-stealing algorithm , where idle threads try to steal work from busy threads. This allows to spread the calculations among various threads and make progress while using fewer threads than it would require with a usual thread pool.

15.How many ways are there to create a thread and run it?

There are 3 ways to create and run threads in java:

First way:

```
Thread t = new Thread(() -> System.out.println("Running thread using lambda expression"));
t.start();
```

Second way:

```
Thread t = new Thread(new Runnable(){

   public void run(){

      System.out.println("Running thread using Runnable instance");
   }

});
```

```
t.start();
```

Third way:

```
Thread t = new Thread(){

    public void run(){

        System.out.println("Running thread using anonymous class");
    }
}

t.start();
```

16.What are the available implementations of ExecutorService interface?

There are mainly 3 implementations of ExecutorService interface.

- ThreadPoolExecutor
- ScheduledThreadPoolExecutor
- ForkJoinPool

17.What special guarantee does the Java Memory Model hold for final fields of a class?

Java Memory Model guarantees that final fields of the class will be initialized before any thread gets hold of the object.
Without this guarantee, an object of the class may be published i.e. become visible to other threads before all the fields of this object are initialized due to reorderings or other optimizations. This could cause racy access to these fields.

Interview Questions on Java 8
1.How to use Default method in java 8?
Java 8 introduced default or "Defender methods" as a new feature which allows developers add new methods definitions in interface without breaking existing functionalities/implementation of these interfaces.

Lets take an example for understanding "How to use it":

```
public interface OldInterface{

    public void existingMethod();

    default public void newDefaultMethod(){

         System.out.println("Code goes here");
    }

}

public class OldImplementation implements OldInterface{

    public void existingMethod(){

    }
}
```

Now create instance of OldImplementation instance and call default method as:

```
OldImplemenattaion olI = new OldImplementation();
olI.newDefaultMethod();
```

Example of Default method added in Java JDK is forEach() which is added in Iterable and Stream interfaces as:

```
public interface Iterable<T>{

    default public void forEach(Consumer<? super T> action){
        for(T t : this)
            action.accept(t);
    }
}
```

2.Define Multiple inheritance ambiguity problem in Default methods.

Since java class can implement multiple interfaces and each interface can define a default method with same method signature , therefore the inherited methods can conflict with each other.

Let's understand it through an example:

```
public interface InterfaceA{

    public void default method(){

    }

}

public interface InterfaceB{

    public void default method(){

    }
}

public class Implementation implements InterfaceA,  InterfaceB {

}
```

The above code will fail to compile.

In order to fix this, we need to provide default method implementation as:

```
public class Implementation implements InterfaceA,  InterfaceB {

    public void default method(){

    }
}
```

If we want to invoke default implementation provided by any of the super Interface, then we can do that as:

```
public class Implementation implements InterfaceA,  InterfaceB {

    public void default method(){
        Interface.super.method();
    }

}
```

Note: We can choose any default implementation or both as part of our new method.

3.What other method has been introduced in interface in Java 8?

In java 8 , static method has also been introduced. These are called by interface name.
They are introduced to increase the degree of cohesion of a design by putting together all related methods in one single place without having to create an object.

For example, see below code:

```
public interface Vehicle{

    static int getPower(){

        return 0;
    }

}
```

Call it as:

Vehicle.getPower();

4.What is the difference between default method and regular method?

Different between default method and regular method are as follows:

- Default method comes with default keyword by default. Regular method doesn't use any such keyword.
- Default method is defined in interface and an interface doesn't have any state, so default method cannot change the state of the object. While regular method can change the state of the object by changing value of it's parameters.

5:How Lambda is useful in java?

Lambda is a type of functional interface. A functional interface is an interface that has exact one abstract method. e.g. : Runnable, Callable, ActionListener, Comparator etc.

Lets' take an example to understand how Lambdas are useful in java:

```
@FunctionalInterface
public interface ITrade{
    public boolean check(Trade t);
}
```

So, above interface has only one abstract method that takes a Trade object.

Lets create a Lambda expression for this interface:

```
ITrade newTradeChecker = (t) -> t.getStatus().equals("new");
```

The real power of Lambdas come when we start creating a multitude of them representing real world behavioral functions as:

```
ITrade bigLambda = (t) -> t.getQuantity() > 13000;

ITrade smallLambda = (t) -> t.getStatus() < 10000;
```

Now, we have a class filterTrades that will filter trades based on which Lambda we have passed for filetering.

```
private List<Trade> filterTrades(ITrade trade, List<Trade> trades){

    List<Trade> newTrades = new ArrayList<>();

    for(Trade t : trades){

        if(trade.check(t)){

            newTrade.add(t);
        }
    }
}
```

So, in this way, it will behave differently based on Lambda passed to filterTrades() method.

6.What are the differences between Lambda and anonymous class?

Below is the list of differences between Lambda and anonymous class:

- **Usage of 'this' keyword**: In anonymous class, this keyword refers to anonymous class itself. While in Lambda, this keyword refers to enclosing class of Lambda.
- **Compilation**: Anonymous class gets compiled to .class file in java. While Lambdas are compiled to private static methods in enclosing class. It uses invokedynamic bytecode instruction to bind this method dynamically.
- Lambdas implement a functional interface, so implements only 1 method. Anonymous class can extend a class or implement any number of interfaces.

7.What are the types of Lambdas?

There are two types of Lambdas in Java 8:

Non-capturing Lambdas: These lambdas only use fields inside their bodies, as:

```
public class NonCapturingLambda{

    public static void main(String[] args){

      Runnable  nonCapLambda = () -> System.put.println("NonCapturingLambda");
      nonCapLambda.run();
    }
}
```

Capturing Lambdas: These lambdas access fields outside their bodies, as:

```
public class CapturingLambda{

    public static void main(String[] args){

      String str = "CapturingLambda";
      Runnable  capLambda = () -> System.put.println("CapturingLambda = "+str);
      capLambda.run();
    }
}
```

8.What are the benefits of using Streams?

Streams were added in java 8 as part of java.util.stream package.

There are multiple benefits of using streams in java :

- Using streams, we can process data in declarative way as we do in SQL.
- Using parallel streams, we can use all available cores in the processor and process data parallely, which improves performance.
- Using streams, we can compose functions and data flows through the functions.

9.Can we use streams with primitives?

Yes, there are two ways to use stream on primitives:

- Use wrapper classes.
- Use primitive types of streams: IntStream, LongStream and DoubleStream.

10.What is the difference between passing int array and string array to Stream.of()?

Stream.of(int[]) gives Stream<int[]>.
Stream.of(String[]) gives Stream<String>

So, when using Stream.of() with int[] array , we get Stream<int[]> and then for getting ints from this stream, we use flatMapToInt(i-> Arrays.stream(i)) to get IntStream and then we can use either map() or forEach().

11.What is a boxed stream in java 8?

If we want to convert stream of objects to collection, we can use:

```
List<String> list = Stream.of("a","b","c","d","e").collect(Collectors.toList());
```

The same process doesn't work on streams of primitives, however.

IntStream.of(1,2,3,4,5).collect(Collectors.toList()); // Compilation Error!!

To convert a stream of primitives, we must first box the elements in their wrapper class and then collect them. This type of stream is called boxed stream.

List<Integer> list = IntStream.of(1,2,3,4,5).boxed().collect(Collectors.toList());

System.out.println(list);

O/P: [1,2,3,4,5]

12.What is the difference between sequential stream and parallel stream?

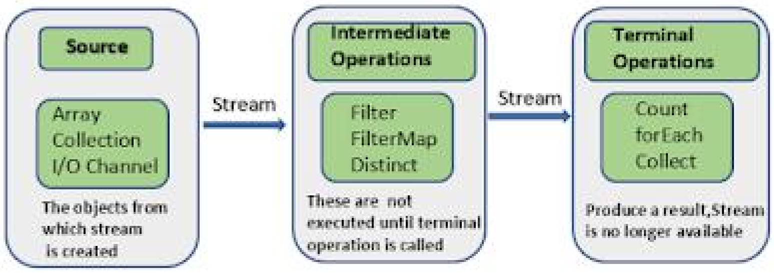
Parallel stream divides the task into many and run them in different threads, using multiple cores of the computer. It uses main thread along with ForkJoinPool.commonPool worker threads.
Sequential stream uses a single core. So, it uses only 1 thread that is main thread.

13. What are the characteristics of a stream?

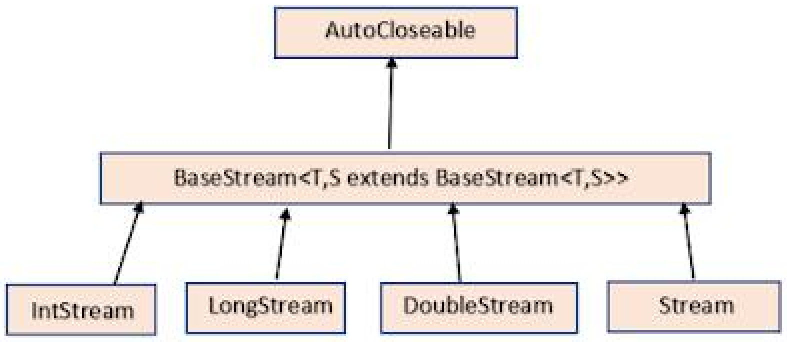
Below is the list of characteristics of streams in java 8:

- **Sequence of elements**: A stream provides a set of elements of specific types in a sequential manner. A stream computes elements on demand. It never stores the elements.
- **Source**: Stream takes Collections, Arrays or I/O resources as input source.
- **Aggregate Operations**: Stream supports aggregate operations like filter, map, limit, reduce, find, match and so on.
- **Pipelining**: Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations.
- **Automatic Iterations**: Stream operations do the iterations internally over the source elements provided, in contrast to collections where explicit iteration is required.

14.Draw Stream lifecycle diagram.


Source
Array
Collection
I/O Channel
The objects from which stream is created
Stream
Intermediate Operations
Filter
FilterMap
Distinct
These are not executed until terminal operation is called
Stream
Terminal Operations
Count
forEach
Collect
Produce a result,Stream is no longer available

15.What is architecture of Steam?

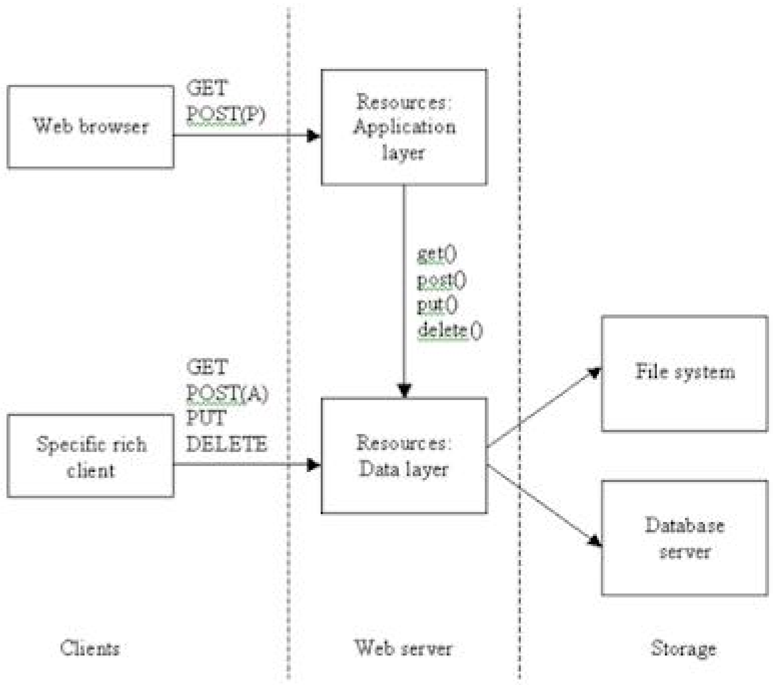


16.How streams are lazy?

Streams are lazy because intermediate operations are not evaluated unless terminal operation is invoked.

Each intermediate operation creates a new stream, stores the provided operation/function and return the new stream. The pipeline accumulates these newly created streams.

Java Interview @ SAP
1.Draw REST Architecture



2.Name core components of HTTP request.

Each HTTP request includes five key elements:

- The verb which indicates HTTP methods such as GET, POST, PUT, DELETE.
- URI: Stands for Uniform Resource Identifier. It is the identifier for the resource on server.
- HTTP version which indicates HTTP version , for example -- HTTP v1.1
- Request Header carries metadata (as key-value pairs) from the HTTP request message. Metadata could be a client type, the format that client supports, message body format and cache settings.
- Request Body: It indicates the message content or resource representation.

3.Explain the term statelessness with respect to RESTFUL Web service.

In REST, ST itself defines State Transfer and statelessness means complete isolation. This means, the state of the client's application is never stored on the server.

In this process, the client sends all the information that is required for the server to fulfill the HTTP request that has been sent. Thus every client request and the response is independent of the other with complete assurance of providing required information.

4.State the core components of an HTTP response.

Every HTTP response includes 4 key elements:

- Status/Response code: Indicates server status for the resource present in HTTP request e.g. 404 means, resource not found and 200 means , response is OK.
- HTTP Version
- Response Header: Contains metadata for the HTTP response in the form of key-value pairs. e.g. content length, content type, response date and server type
- Response Body: Indicates response message content or resource representation.

5.What is cloud and what are the benefits of Cloud Computing?
A cloud is actually a collection of web servers [instead of single server] owned by 3rd party. Cloud provides inexpensive, efficient and flexible alternative to computers.

Benefits of Cloud are:

- No need of extra space required to keep all the hardware [Servers, Digital storage]
- Companies don't need to buy software or software license for all it's employees. They just pay a small fees to the cloud computing company to let their employees access a suite of software online.
- It also reduces IT problems and costs.

6.

```
class A{


}


class B extends A{


}

class C extends B{
}

public class MainClass{

  static void overloadedMethod(A a){
     System.out.println("One");
  }

  static void overloadedMethod(B b){

     System.out.println("Two");
  }


  static void overloadedMethod(Object obj){

     System.out.println("Three");
  }

  public static void main(String[] args){

     C c =  new C();
     overloadedMethod(c);

  }

}
```

Given above code, what will be the output?

Output: Two

Explanation:

In method overloading, more specific method is chosen over generic.

7.As a best practice , what should be the maximum nesting level for associations in URI (e.g. /vacations/{id}/reviews)?

3

8.In a class, one method has two overloaded forms.One form is defined as static and another form is defined as non-static. Is that method properly overloaded?

Yes. That method is properly overloaded. Compiler checks only method signature to verify whether a particular method is properly overloaded or not. It doesn't check static or non-static feature of the method.

9. Method signature consists of which one out of below:

1. **Method name, return type and number of arguments**
2. **Access modifier, method name and types of arguments**

3. **Method name, number of arguments, types of arguments and order of arguments**
4. **Return type, access modifier and order of arguments**

Java Interview @ GalaxE India
1.What are Binary Literal? How to use them in java?

Binary Literals were introduced in java 7. Now using them, we don't need to convert binary to decimal or hexadecimal.

Binary Literals must be started with 0b or 0B.

Binary Literals are used in Protocols, Processors and bitmapped hardware devices.

Example showing their usage:

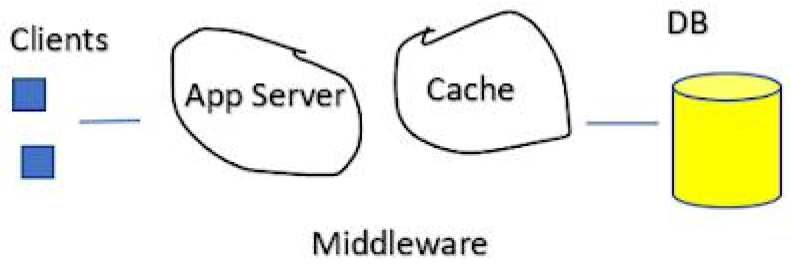
```
public class BLiterals{

    public int a = 0b0110;
    public byte b  = (byte) 0b0110;
    public long l = (long) 0b0110L;


   System.out.println("a = "+a);
   System.out.println("b = "+b);
   System.out.println("l = "+l);

}
```

Output:

```
a = 6
b = 6
l = 6
```

2.Which Application server have you used? Where does it occur in multi-tier architecture? What benefits we get while using an Application server?

I have used Weblogic server.



Weblogic server provide support for Network protocols [HTTPS, SOAP etc.] It also provides data access and persistence from database server. It also supports SQL transactions for data integrity.
Weblogic also provides security.So this means, when we use Weblogic server , we don't have to care

about protocol, security, database integrity, transactions etc. All these are handled by Weblogic itself. We just have to focus on business logic.

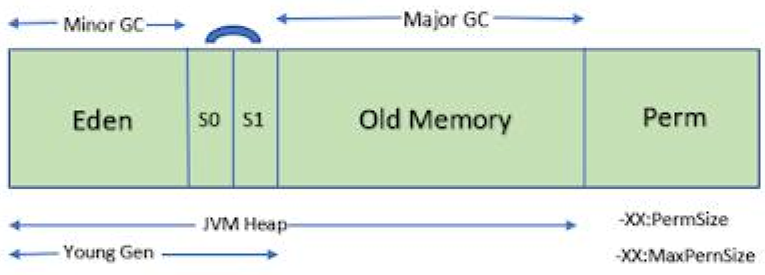
3.Write algorithm for Level-Order traversal of a Binary tree.

Level-Order traversal means moving from root to leaf step-by step horizontally.

Algorithm for Level-Order Traversal:

1. Check if root node is empty. If yes, then return.
2. If root not null, create a queue and put root node in the queue.
3. Take a while loop on if Queue is not empty.
4. store the size of queue in a variable named size.
5. Create another while loop inside outer loop. IN this loop, check the value of size variable. It should be > 0. Use size-- in while loop.
6. Now print element[node] from queue. And put all child nodes of that node on queue if these are not null.
7. Continue from step 5 until it is false and then continue from step 3.

4.Explain Java Memory Model.



5.Explain JVM memory structure.
As per the spec, JVM is divided into 5 virtual memory segments:

- Heap
- Method [Non-heap]
- JVM Stack
- PC Registers
- Native Stack

JVM Stack:

- Has a lot to do with methods in java classes
- Stores local variables and regulates method invocation, partial result and return values.
- Each thread in java has it's own copy of stack and is not accessible to other threads.
- Tuned using -Xss JVM option.

Native Stack:

- Used for native methods [Non-java code]

6.What are the common Java Heap related issues?

Below is the list of all java heap related issues which occur in java applications at runtime.

- **'OutOfMemory' error due to insufficient Heap**
- To identify it, we can use JvisualVM

- To fix it, we can increase heap size or we can revisit the code to see why the demand is high in first place.
- **Poor application response time due to long garbage collection pauses**
- TO identify it, we can use JvisualVM
- To fix this, we can tune GC [Garbage Collector].
- **OutOfMemory error due to memory leak**
- To identify it, we can use JvisualVM
- To fix it, we need to analyse the code and correct it.
- **Heap Fragmentation**
- It is due to when , small and large objects are allocated in a mixed fashion. To some extent, we cannot avoid heap fragmentation -- over time, heap will get fragmented.
- To identify, we see poor application response times, longer GC pauses and in some cases 'OutOfMemory' errors.
- To fix it, tuning can help.

7.What are the ways to capture Java Heap dump?

There are great tools like Eclipse MAT and Heap Hero to analyze Heap dumps. However we need to provide these tools with heap dumps captured in the correct format.

Options to capture heap dump are:
- Jmap
- Jcmd
- JvisualVM
- JMX

8. Why reflection is slow?

Reflection needs to inspect metadata in bytecode instead of just using precompiled addresses and constants.

Everything requires to be searched. That's why reflection is slow.

Java Interview @ Gemini Solutions
1.Write algorithm for Infix expression evaluation.
expression is : 3*(5-1)/3 + 1

Algorithm:

Approach: Use two Stacks

- operand stack
- operator stack

Process:
1. Pop out two values from operand stack, let's say it is X and Y.
2. Pop out operator from operator stack. Let's say it is '+'
3. Do X + Y and push the result on the operand stack.

Iterate through given expression, one character at a time.

1. If a character is an operand, push it on the operand stack.
2. If the character is an operator,

- If the operator stack is empty, push it onto the operator stack.
- Else if the operator stack is not empty,
- If the character's precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.
- If the character's precedence is less than the precedence of the stack top of the operator stack , then do Process [As described above] until character's precedence is less or stack is not empty.

3. If the character is "(", then push it onto the operator stack.
4. If the character is ")", then do Process [As described above] until the corresponding "(" is encountered in operator stack. Now just pop out the "(".

2.What is the difference between Lambda and anonymous class?

There are multiple differences between lambda and anonymous class:

- Usage of 'this' keyword: IN anonymous class, this keyword resolves to anonymous class itself. Whereas for Lambda, this keyword resolves to enclosing class where lambda expression is written.
- Compilation: Java compiler compiles lambda expression and convert them into static private method of the class. It used invokedynamic bytecode instruction that was added in java 7to bind this method dynamically. While anonymous class compiles to a class. The compiler generates a class file for each anonymous inner class.
- Lambdas implement a functional interface, so implements only 1 method. Anonymous class can extend a class or implement multiple interfaces with any number of methods.

3.Design Email [Inbox, Outbox, attachment]. Write all technologies that will be used. Create Class diagram.

4. What is the use of volatile keyword? How does it help in synchronization? Does it provide full synchronization?

Volatile is used to avoid the local copy of a variable inside a thread. All threads will access the volatile variable from main memory , updates it and immediately put it back in main memory.
All, reads and writes would be made directly to main memory instead of to registers or local processor cache.

volatile variable is not thread safe. If multiple threads try to write on a volatile variable, then Race condition occurs.

Each thread has a separate memory space known as working memory. This holds the value of different variables for performing operations. After performing an operation, thread copies the updated value of the variable to main memory and from there other threads can read the latest value.

But if the variable is non-volatile, then new value may not be flushed immediately to main memory. And other threads may not read the updated value.

5.Why to use transient variable? What will be the value of transient variable after deserialization? Where should we use transient variable?

transient variable is used when we don't want value of that variable to be serialized.
So, if a variable is declared as transient, then it's value will not be serialized.

On deserialization, we get the default value of transient variable. e.g. :

transient int i ;

When the instance of this class is serialized, then upon deserialization, we get 0 in i , as 0 is the default value for an int in java.

Even if we declare i as transient int i = 10;
In this case, as well we get value of i as 0 after deserialization.

Use case [Where should we use it]:

When we have a variable whose value can be calculated from other variables, the we should make it transient, as it should be calculated from other variables every time.

6.How will you write an immutable class?

There are two ways to write an immutable class:

- Make the class final and all it's instance variables as final, so that they can be initialized in constructor only.
- Make all instance variable of the class as private and don't modify them inside in method. Means don't provide any setters methods. And also make all getters methods as final so that they can't be overridden and subclasses cannot override them and return some other values from overridden forms.

7.Write Lambda expression for printing even numbers from a list.

Suppose , we have a list of integers.

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

List<Integer> list = Arrays.asList(array);

list.stream().filter(i-> i%2 == 0).collect().forEach(System.out :: println);

8.Difference between default method and regular method in java 8.

Differences between default method and regular method are as follows:

- default method has default keyword in it's method signature. While regular method has no such keyword.
- default cannot change the state of the object as it works only inside interface and ca only access it's local variables. While a regular method ca change the state of the object by modifying method arguments as well as fields of the class.

Java Interview @ OLX
1.If you have three, you have three. If you have two, you have two, but if you have one, you have none. What is it?
Choices

2. 3 bulbs 3 switches problem:

There is a room with a door [Closed] and three light bulbs. Outside the room, there are 3 switches, connected to the bulbs. You may manipulate the switches as you wish, but once you open the door, you can't change them. Identify each switch with it's bulb.

Turn on 1 switch and keep it on for 5 minutes. Now turn it off and turn on 2nd button and enter the room. Now the bulb which is ON maps to ON switch.The hot bulb maps to previous switch [which was turned on first] and 3rd bulb maps to 3rd switch.

3.REST API must use HTTP. Is that true or false?

FALSE

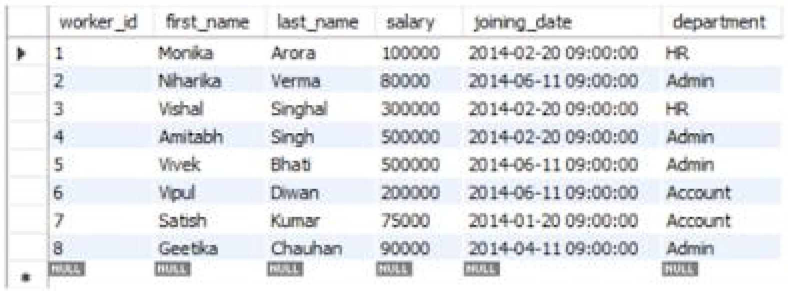
4.A resource in the context of REST is (or may be) which one of these:

- Thing
- Object
- Real World Entity
- An account
- An Item
- A Book
- All of the above

All of the above

5.Suppose you have a "Worker" table as shown below. You have to write a SQL query to find employees with different salary.

Show only 2 Columns in output: first_name and salary.



worker_id	first_name	last_name	salary	joining_date	department
1	Monika	Arora	100000	2014-02-20 09:00:00	HR
2	Niharika	Verma	80000	2014-06-11 09:00:00	Admin
3	Vishal	Singhal	300000	2014-02-20 09:00:00	HR
4	Amitabh	Singh	500000	2014-02-20 09:00:00	Admin
5	Vivek	Bhati	500000	2014-06-11 09:00:00	Admin
6	Vipul	Diwan	200000	2014-06-11 09:00:00	Account
7	Satish	Kumar	75000	2014-01-20 09:00:00	Account
8	Geetika	Chauhan	90000	2014-04-11 09:00:00	Admin
NULL	NULL	NULL	NULL	NULL	NULL

```
select distinct w1.salary , w1.first_name
from Worker w1, Worker w2
where w1.salary = w2.salary
AND w1.worker_id = w2.worker_id;
```

6.Write SQL query to display first 5 records from the table shown above?

```
select * from worker LIMIT 5;
```

7.What is the difference between UNION and UNION ALL?

UNION removes duplicate records. UNION ALL does not.

There is a performance hit when using UNION instead of UNION ALL, since the database server must do additional work to remove the duplicate rows. But usually , we don't want the duplicates especially when developing reports.

8.What are foreign key and super key?

Foreign Key:

Foreign key maintains referential integrity by enforcing a link between the data in two tables.
Foreign key in child table references the primary key in parent table.
The foreign key constraint prevents actions that would destroy the link between the child and parent table.

Super key : It is a column or a combination of columns which uniquely identifies a record in a table.

e.g.:

Super key stands for superset of a key. e.g. We have a table Book with columns:
Book (BookID, BookName, Author)

So, in this table we can have:

- (BookID)
- (BookID, BookName)
- (BookID, BookName, Author)
- (BookID, Author)
- (BookName, Author)

as our super keys.

Each super key is able to uniquely identify each record.

Java Interview @ OrangeMantra
1.List all types of HTTP Error code categories

There are mainly 5 categories in HTTP Response codes:
- 1XX : Informational
    - 100 : Continue
- 2XX : Success
    - 200 : OK
    - 201 : Created
    - 202: Accepted
    - 204: No Content
- 3XX : Redirection
- 4XX : Client Side Error
    - 400 : Bad Request
    - 401 : UnAuthorized

- 402 : Payment Required
- 403 : Forbidden
- 404 : Not Found
- 5XX : Server Side Error
- 500 : Internal Server error
- 501: Not Implemented
- 502: Bad Gateway
- 503: Web service not available

2.List all the features in Java 7

There are multiple features which have been added in java 7:

- String in switch
- try with resources
- Binary Literals
- multiple Exception types in catch block

3. Explain about one very typical designing/Coding situation which you have solved.

4. Explain all differences between Stack and Heap?

- Heap is very large in size while Stack is small in size as compared to Heap.
- Every thread has it's own Stack while Heap is shared among all threads.
- When Heap gets full, java.lang.OutOfMemoryException is thrown. While in case of Stack, java.lang.StackOverflowException is thrown.
- Heap is mainly used to store the objects, while Stack is used for storing local data, method body , method return values etc.
- Heap is tuned by using -Xmx and -Xms options. While Stack size is tuned by -Xss.
- In Heap, data is stored in random order. While in Stack , data is stored in LIFO [Last-In-First-Out] order.

5.What is the contract between natural ordering and equals() method in java?

Contract between natural ordering or ordering provided by comparator interface and equals() method is that the output returned by natural ordering / comparator interface should be consistent with value returned by equals() method.

If these values are not consistent, then it will lead unexpected results in TreeMap and TreeSet Collections.

6.What is the use case of Marker interface ? How to write custom marker interface?

Marker interface is used to indicate something special to JVM/Compiler so that it can perform necessary actions on the class implementing marker interface.

Examples of marker interfaces are : Serializable, Cloneable, Remote

Custom marker interface:

```
public interface Ingestible{
  //nothing
}

public interface Edible extends Ingestible{
}

public interface Drinkable extends Ingestible{

}

public class Food implements Edible{

}

public class Drink implements Drinkable{

}

public class Person{

    public void ingest(Ingestible something){

         if(something instanceof Edible){

              System.out.println("something can be eaten");

         }
    }
}
```

In above example, we are telling JVM that the class implementing Ingestible interface can be ingested.

Use case:
A marker interface is used for Identification purpose.

7.Why to use BigDecimal over Float or Double?

Float and Double cannot be used for precise values: e.g.:

```
double d1 = 131.44;
double d2 = 130.34;

System.out.println("d1 - d2 = "+ (d1-d2));
```

Output: 1.10000000000013

That's why in financial applications, where while doing calculations , scale and rounding mode for the numbers is an important aspect . So it's a better choice to go with BigDecimal.

Using BigDecimal:

```
BigDecimal bd1 = new BigDecimal("131.44");
BigDecimal bd2 = new BigDecimal("130.44");
```

System.out.println(bd1-bd2); **gives 1.10**

8.

```
Integer i = 127;
Integer j = 127;

Integer ii = 128;
Integer kk = 128;

if( i == j){
   System.out.println("true");
}
else{
   System.out.println("false");
}

if(ii == kk){
   System.put.println("true");
}
else{
   System.out.println("false");
}
```

What will be the output?
true
false

9.What is Serverless architecture?

Serverless is a cloud computing execution model where the cloud provider dynamically manages the allocation and provisioning of servers.

A serverless application runs in stateless compute containers that are event-triggered , ephemeral(may last for one invocation) and fully managed by the cloud provider.

Pricing is based on the number of executions rather than pre-purchased compute capacity.

Our applications are installed in terms of cloud functions. There are multiple cloud services that act as Functions as a service [FaaS].

- IBM OpenWhisk
- AWS Lambda
- Microsoft Azure Functions
- Google Cloud Functions
- Auth0 Webtask

So, serverless computing [or serverless for short] is an execution model where the cloud provider is responsible for executing a piece of code by dynamically allocating the resources and only charging for the amount of resources used to run the code.

The code typically runs inside stateless containers that can be triggered by a variety of events including Http requests, database events, file uploads, monitoring alerts etc.

10. What happens when we call SpringApplication.run() method in main class of SpringBootApplication?

Syntax of the class containing main method is:

```
@SpringBootApplication
public class TestApplication{

    public static void main(String[] args){

        SpringApplication.run(TestApplication.class); // returns
ConfigurableApplicationContext            instance
    }

}
```

When we run this class as a java application then our application gets started.

When this method is called, then Spring Container gets started.
Spring container once started is responsible for:

- Creating all objects: This is done by component scan. As @SpringBootApplication is a combination of @Configuration , @ComponentScan and @EnableAutoConfiguration.
- Dependency Injection
- Managing the lifecycle of all beans.

Steps executed under this method:

- Application Context is started
- Using application context , auto discovery occurs: @ComponentScan
- All default configurations are setup i.e. based on dependencies mentioned, spring boot automatically sets up defaults.
- An embedded servlet container is started.[No need to setup a separate web server]. Note: Embedded servlet container is launched only if web is mentioned in a dependency.

11.What are Spring Boot Starter projects. Name some of them?

Starters are a set of dependency descriptors that we can include in our application. We get one-stop-shop for all required technologies in our application. without having to hunt through sample code and copy paste loads of dependency descriptors.

e.g.: If we want to get started using Spring and JPA for database access, just include the sprint-boot-starter-data-jpa dependency in the project and we are good to go.

Below is a list of few starter projects provided by Spring Boot:

- sprint-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-web-services
- sprint-boot-starter-test
- sprint-boot-starter-data-rest
- spring-boot-starter-jdbc

- spring-boot-starter-security

12.How does spring enables creating production ready applications in quick time?

Spring Boot enables creating production ready applications by providing a few non-functional features out of the box like caching, monitoring, logging and embedded server.

Spring-boot-starter-actuator: For monitoring and tracing of application

Spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat: To pick the choice of Embedded Servlet Container.

Spring-boot-starter-logging: For logging using Logback

Spring-boot-starter-cache: Enabling spring framework's caching support

13.What is the minimum baseline version for spring boot 1.5 and 2.1.0 versions?

Java 8

14.How many ways are there to create ApplicationContext?

If we are using xml configuration using application-context.xml then

```
ApplicationContext context = new ClassPathXmlApplicationContext("Path to xml file");
```

If we are using Java configuration, then:

```
ApplicationContext context = new AnnotationConfigApplicationContext(ConfigClass.class);
```

Java Interview in Concirrus
1.How Singleton is handled in Deserialization? What, if we use clone() in singleton?

When we serialize Singleton instance and try to deserialize it, then if we call deserialization multiple times, then it can result in more than one instance of singleton class.

To avoid this problem, we can implement readResolve() method. This readResolve() method is called immediately after an object of this class is deserialized, Means when ObjectInputStream has read an object from input stream and is preparing to return it to the caller, then it checks whether the readResolve() method is implemented or not.

Note: Both objects [Read by ObjectInputStream and returned by readResolve() method] should be compatible [Identical] else ClassCastException is thrown.

Implementation of readResolve() method:

```
protected void readResolve(){
   // Instead of the object we are on, return the class variable singleton.
   return singletonInstance;
}
```

What if we use clone() method in Singleton?

If we override clone() method in singleton class, then we must throw CloneNotSupportedException from this method.

2.How HashMap works?

HashMap in java contains key-value pairs. It doesn't maintain the insertion order of key-value pairs.

Whenever a key-value pair is required to be stored in HashMap, The overridden hashcode() method [if there is one] is used to calculate the hashcode and then Object's class hash() method is called on that value which provides the index in underlying bucket of HashMap.

HashMap uses array based structure to store each key-value entry. That array is called bucket and each location in bucket maintains a simple optimized LinkedList.

So, after the index in the bucket is found, then if there is no entry matching the key in that bucket indexed LinkedList, then that key-value pair is stored. Else , if there is some key-value pairs already there in the LinkedList, then key is compared to every key in LinkedList using equals() method and if a match found, then the value for that key is replaced with new value.

If there is no match found, then that new key-value pair is stored as a new entry in HashMap.

3.Create Immutable class?

There are 2 ways to create immutable class in java:

- Make the class final and all it's instance variable as final. So they can be initialized in constructor only.
- Make all the instance variables private and don't change them except constructor. Don't provide any setter methods for them. Make the getter methods final so that subclasses don't override these getter methods and return other values.

4.Detect loop in LinkedList

```
class LinkedList{

    static Node head;

    static class Node{
        int data;
        Node next;

        Node(int d){
            data = d;
            next = null;
        }
    }

    int detectLoop(Node node){

        Node slow = node, fast = node;

        while(slow != null && fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;

            // If fast node reaches slow node, means there is a loop.
            if(slow == fast){

                return 1;
            }
        }
        return 0;
}


}
```

5.Dynamically confiuring a new microservice. Does it require deploying all microservices, if we use static way?

No. We can just deploy our microservice to some host and it will register itself with Service Registry. From there it can be discovered by other microservices and also it can discover other required microservices.

And this way, communication can be established among various microservices without any external configuration.

6.Is there any loophole in using Git-> Jenkins CI/CD way?

There is no loophole in Git-> Jenkins CI/CD way. The problems/loopholes occur in the way these tools are configured by Software Delivery teams [DevOps].

There are multiple things that can cause problems in using Jenkins as CI/CD tool:

- Jenkins has too many plugins
- Jenkins was not designed for the Docker age
- Jenkins doesn't support microservices well

Jenkins has too many plugins:

Plugins are good when they are used properly and efficiently. Plugins give users the choice to add various features to the tools they use.
But in Jenkins, for achieving every single basic task, you need a separate plugin.

e.g.: for creating a build for Docker environment, you need a plugin.
To pull code from Github, you need a plugin.

Jenkins was not designed for the Docker age:

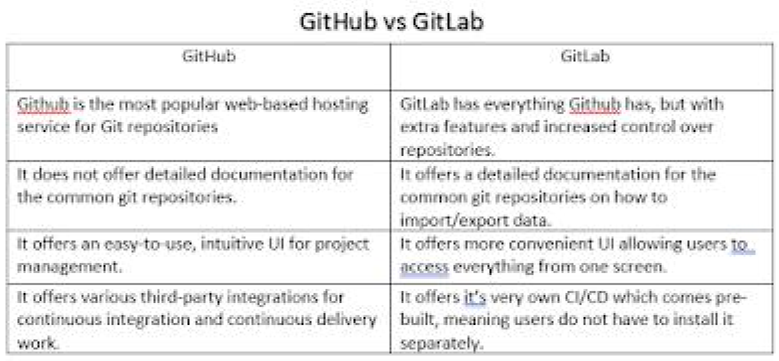
CI servers don't match with Docker container infrastructure easily. They require multiple plugins to integrate with Docker. There are more than 14 plugins with Docker in their names. And almost 6 of them are for core Docker platform.

7.Design TinyURL algorithm.

8.Which Docker command is used to know the list of running containers?

docker ps

9.Difference between GitHub and GitLab?



GitHub vs GitLab

GitHub	GitLab
Github is the most popular web-based hosting service for Git repositories	GitLab has everything Github has, but with extra features and increased control over repositories.
It does not offer detailed documentation for the common git repositories.	It offers a detailed documentation for the common git repositories on how to import/export data.
It offers an easy-to-use, intuitive UI for project management.	It offers more convenient UI allowing users to access everything from one screen.
It offers various third-party integrations for continuous integration and continuous delivery work.	It offers it's very own CI/CD which comes pre-built, meaning users do not have to install it separately.

Interview @ NexGen Solutions
1. What is the benefit of Executor framework?

Executor Framework provides separation of Command submission from Command execution.

It has several interfaces. Most important ones are:

- Executor
- ExecutorService
- ScheduledExecutorService

Executors is Factory methods class used in this framework.

2. Asked me to write custom ArrayList.

//You can write your own implementation

3. In your Custom ArrayList, there are multiple methods. How will you provide synchronization?

We have two ways of providing synchronization in this source code:

- Make each method synchronized by placing synchronized keyword before return type of method
- Use synchronized block inside method body.

4.How did you use Amazon EC2?

I installed EC2 instance using Amazon Management console after login into it.
I used General instance for my purpose, as in my application I needed to send multiple emails to client very fast.

5.What is the benefit of Generics in java?

Generics provides Compile-time safety. And it also avoids ClassCastException. It was introduced in Java

5.

6.How Generic code written in java 5 runs in Java 4 compiler?

Java compiler uses type check information at compile time and after all validation it generates type erased byte code which is similar to the byte code generated by Java 4. So, it provides backward compatibility.

7. Asked about my last project.

I explained it.

8.Why we can't pass List<String> to List<Object> in java?

String is a subclass of Object, but List<String> is not a subclass of List<Object>.

If it would be allowed, then look at below code:

```
List<Object> listO = new ArrayList<String>();
listO.add("String");
listO.add(new Object());
```

Now, in the same List , we have both a String and Object object. Which is wrong.

Java Interview @ Dew Solutions
1.Java is pass by value or reference.

Everything in java is pass-by-value.
When we pass object reference to some method, we actually pass address of that reference which in turn contains memory address of actual object.
And as we know, memory address is a value, so means, we always use pass-by-value in java.

e.g.:

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

   Book book = new Book("Java");
   Book newBook = book;

   change(book);
   book.getName();// Prints Java

}

public void change(Book book){
   book.getName(); //Prints Java

   book = new Book("Angular");
   book.getName();// Prints Angular
}
```
So here, what we see Book name doesn't change after change() method, because passed book reference in change() method now points to new Book Object with value "Angular".

Now, lets look at another example:

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

   Book book = new Book("Java");
   Book newBook = book;

   change(book);
   book.getName();//Prints .Net
}

public void change(Book book){
   book.getName();//Prints Java

   book.setName(".Net");
}
```

Here, what we see, value of book object after change() method call has been changed to .Net. It is because book reference in change() method still contains memory address of Book object with java as value.

2.We have multiple Employee objects and we need to store these objects in TreeMap. What problems we can face while storing Employee objects?

We will get error at runtime : "No suitable method found for sort(List<Employee>)".
It is because Employee class doesn't implement the Comparable interface so the sort() method cannot compare the objects.

As TreeMap stores object in Ascending order by default using natural ordering. So, each object which needs to be stored in TreeMap should implement Comparable interface or Comparator interface.

3.What is Callable interface and how it is similar/different to/from Runnable?

Callable interface is just like Runnable interface.

Difference between Callable and Runnable is given below:

- Runnable is defined in Java 1.0 . While Callable was introduced in Java 5.
- Callable includes call() method which returns object and also throws Checked exception. On the other hand , Runnable's run() method neither return any value nor throws any Checked exception.
- Runnable instance can be passed to a thread. While Callable instance can't be passed to a thread.

Similarities between Runnable and Callable:

- Both are used to encapsulate code which needs to be executed parallelly in separate thread.
- Both interfaces can be used with Executor framework introduced in java 5.
- Both includes single abstract method. Means , both can be used in Lambda expressions in java 8.

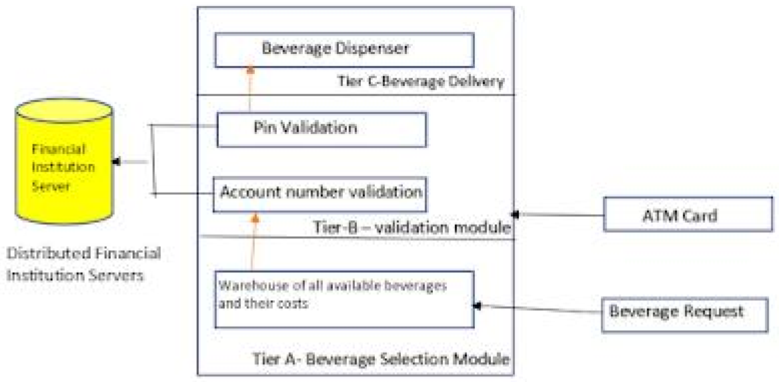
4.Can HashSet store duplicate objects?

No, HashSet cannot store duplicate objects. As HashSet is implementation of Set interface and Set interface is meant to store unique elements. That's why HashSet doesn't store duplicate objects.

5.SQL query to find employee with highest salary and corresponding department.

```
select department , max(salary) from employee
                         group by department;
```

6.Design the architecture of Vending machine?



Beverage Dispenser
Tier C-Beverage Delivery
Pin Validation
Financial Institution Server
Account number validation
Tier-B – validation module
ATM Card
Distributed Financial Institution Servers
Warehouse of all available beverages and their costs
Beverage Request
Tier A- Beverage Selection Module

7.Describe one problem that you have solved in production build?

In my previous company, client complained about slowness in the system performance.
We tracked the logs and found there was problem with some memory leaks happening.

We just immediately asked client to increase the JVM heap size for the moment so that it worked and later on we corrected the problem by removing all the memory leaks in the code.

8. Why to use Lock in multithreading? What are the locking mechanisms available in java?

Lock is required in multithreading to properly distribute access of Monitor associated with an object among multiple threads.
If we don't use Locks, then multiple threads will try to acquire shared resources and we will get corrupted outputs from threads.

Locking mechanisms available in java are:

- synchronized block
- synchronized methods
- Read/Write lock
- Atomic classes
- final classes

Java Interview @ CapGemini
1.How to handle transaction in Spring?
In Spring, there are two ways of managing transactions:

- Programmatically manage by writing custom code
- Using Spring to manage transaction
    - Programmatic Transaction management
    - Declarative Transaction management

Programmatically manage by writing custom code:

```
EntityManagerFactory factory = Persistence.createEntityManagerFactory("persistance_unit_name");
EntityManager entityManager = factory.createEntityManager();
Transaction transaction = entityManager.getTransaction();

try{
   transaction.begin();
   //..........Business logic
   transaction.commit();
}
catch(Exception e){
   transaction.rollback();
   throw e;
}
```

Pros:
- The scope of the transaction is very clear in the code

Cons:
- It's repetitive and error prone
- Any error can have a very high impact
- A lot of boilerplate code needs to be written and if you want to call any another method from this method, then again you need to manage it in the code.

Using Spring to manage transaction:

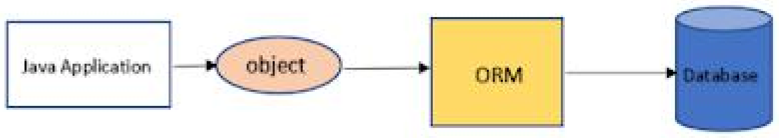
1. Programmatic Transaction Management:
- Using the transaction template
- Using a PlatformTransactionManager implementation directly

2. Declarative Transaction

2.What is ORM and how to use it?

ORM means Object Relational Mapping. It is a tool that helps in interacting with the database.

for example JPA is an ORM which specifies some specifications which needs to be implemented.
In Spring, Hibernate is the implementation of JPA specifications.



So using ORM, Java object is mapped to relational table in database.

3.What is Heap dump?

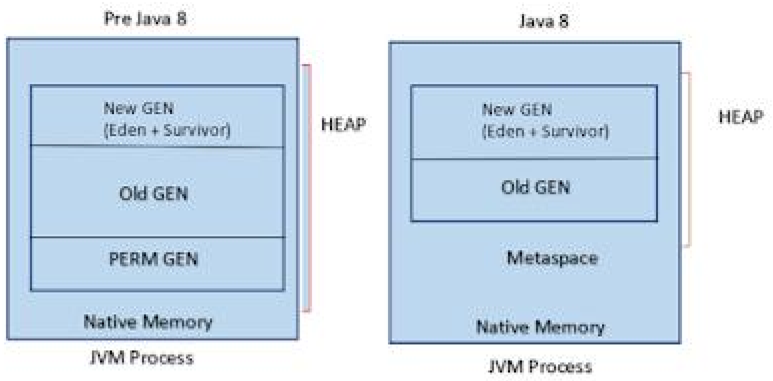
Heap Dump: It is a snapshot of the memory of a java process.
This snapshot contains information about java objects and classes in the heap at the moment the snapshot is triggered.

Typically , a full GC is triggered before the heap dump is written, so the heap dump contains information about the remaining objects in the heap.

Typical information in a heap dump are:

- All objects: Class, fields, primitive values and references
- All Classes: Class Loader, name, super class and static fields
- Garbage collection roots: It is an object that is accessible from outside the heap.

4.Memory management in Java 8?



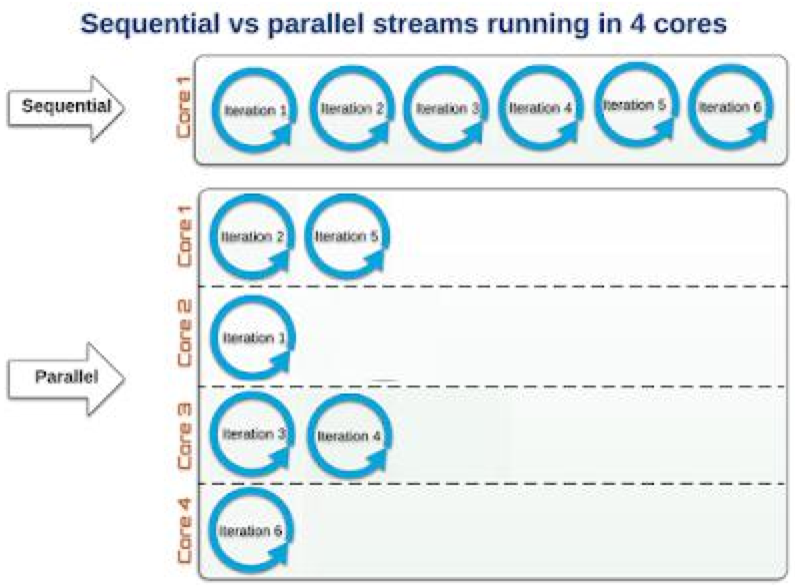
Perm Gen in java 7 and earlier version has a default maximum size of 64 MB for 32-bit machine and 84 MB for 64-bit machine.
While default maximum size for metaspace is unlimited.

In java 8 ,we can set the initial and maximum size of metaspace using:

-XX:MetaspaceSize = N : Sets the initial and minimum size of metaspace
-XX:MaxMetaspaceSize=N : Sets the maximum size of the metaspace

5.Difference between Serial and Parallel streams.
Parallel streams divide the task into many and run them in different threads, using multiple cores of the computer. It uses main thread along with ForkJoinPool.commonPool worker threads.

Sequential stream uses a single core. So it uses only 1 thread that is main thread.



6.How to find OutOfMemoryError in production app? And how to avoid this error immediately for long time?

If we face OutOfMemoryError in production and we cannot really reason about it from stacktraces or logs, we need to analyze what was in there.

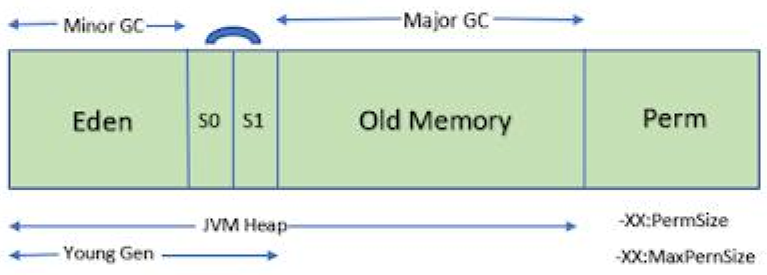
Get the VM Heap dump on OOM:

- -XX:+HeapDumpOnOutOfMemoryError
- -XX:HeapDumpPath="/tmp"

And use that for analysis. We can use memory analyzer tool for this.

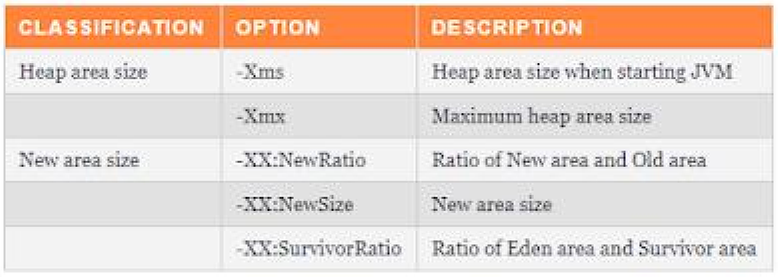
How to avoid this error immediately for long time?

We can increase Heap space using -XX argument for a while and it will work fine.
7.What is internal structure of heap memory?



8.How to configure GC in Java application?

GC tuning is only required when we have not set memory size in the application and too many Timeout logs are printed.



CLASSIFICATION	OPTION	DESCRIPTION
Heap area size	-Xms	Heap area size when starting JVM
	-Xmx	Maximum heap area size
New area size	-XX:NewRatio	Ratio of New area and Old area
	-XX:NewSize	New area size
	-XX:SurvivorRatio	Ratio of Eden area and Survivor area

9.What are Serial and Parallel GC's?

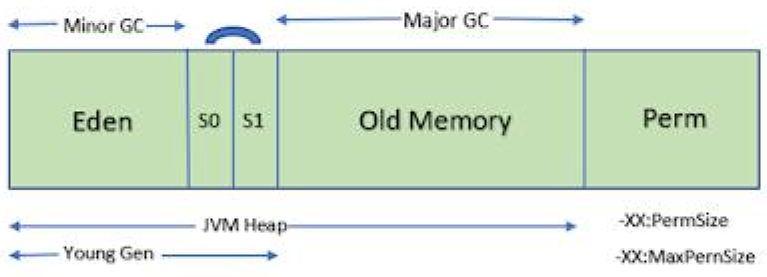
Serial Collector:

- Used for single threaded environments [32-bit Windows] and apps that use small heap
- It stops all threads when it is running. Means it is not suitable for server environment.
- We can use it by turning on the -XX:+UseSerialGC JVM argument.

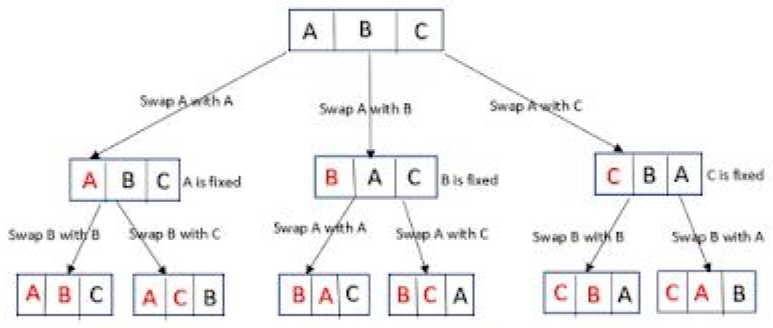
Parallel Collector:

- This is JVM's default garbage collector
- It uses multiple threads to scan through and compact the heap and speeds up garbage collection.
- The downside is that it will stop application threads when performing either a minor GC or full GC.
- It is best suited for apps that can tolerate application pauses.
- Although it is default collector, but it can be configured using -XX:UseParallelGC JVM argument

10. What is Java Memory model?



Java Interview @ BirdEye
1.Print all unique permutations on a String?



2.Write Algorithm for custom BlockingQueue.

Algorithm steps:

- Define an array to store elements for queue. Specify the initial size for that array.
- Use Lock and conditions objects to create custom blocking queue.
- Define two methods , put() and take().
- While putting the elements in queue, check the size of array. If it is full, then the producer will wait for the queue to have some space.
- While consuming element from queue, if the queue is empty, then consumer will wait for the queue to have some objects in it.

Packages to be included:

java.util.concurrent.locks.Condition;
java.util.concurrent.locks.Lock;
java.util.concurrent.locks.ReentrantLock;

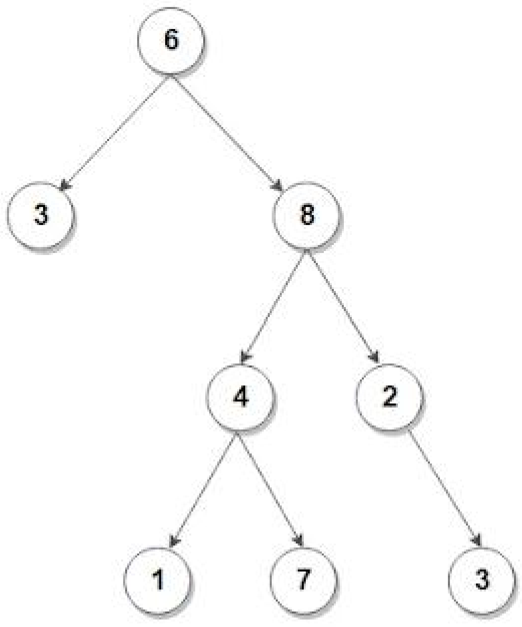
3.Singleton and Synchronization question:

If Thread1 calls synchronized method in Singleton class, then can another thread call getInstance() method [If synchronize(Singleton.class) is 1st statement in getInstance() method] of singleton class?

Yes, another thread can call getInstance() method of singleton class. It is because, this time thread will acquire lock on Class object [Singleton.class].

So first thread acquired lock on Singleton instance and this another thread will acquire lock on Singleton Class's object.

4.Print all edge/corner nodes of a binary tree.



Follow Level order traversal of binary tree. So , while doing level order traversal, if the current node happens to be the first node or last node in current level, print it.

```
void print(Node *root){

   //return if tree is empty.
   if(root == null)
      return;

   // Create an empty queue to store tree nodes.
```

```
    Queue<Node*> q;

    //enqueue root node
    q.push(root);

    // run till queue is not empty

    while(!q.empty()){

        //get size of current level
        int size = q.size();
        int n = size;

        //Process all noes present in current level
        while(n--){

             Node* node = q.front();
             q.pop();

              // If corner node found, print it.
              if(n == size-1 || n == 0)
                  println(node);

              //enqueue left and right child of current node
              if(node-> left != null)
                  q.push(node->left);

              if(node -> right != null)
                  q.push(node->right);
        }
        //terminate level by printing newline
        println();

    }
}
```

5.How locking mechanism is implemented by JVM?

The implementation of locking mechanism in java is specific to the instruction set of the java platform. For example with x86, it might use the CMPXCHG instruction - atomic compare and exchange - at the lowest level to implement the fast path of the lock.

The CMPXCHG instruction is a compare-and-swap instruction that guarantees atomic memory access at the hardware level.

If the thread cannot acquire the lock immediately , then it could "spinlock" or it could perform a syscall to schedule a different thread. Different strategies are used depending on the platform , JVM Switches.

6.Is Java pass-by-value or pass-by-reference?

Java is always pass-by-value. Whenever we pass an object to some method, then we actually send a reference variable that points to the actual object. Means that reference variable will be containing the memory address as value. That's why Java is called as pass-by-value.

Let's take an example:

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

   Book book = new Book("Java");
   Book bookNew = book;

   change(book);
   book.getName(); //Prints Java
}

public void change(Book book){
   book.getName() // Prints Java

   book = new Book("Angular");
   book.getName(); // Prints Anguar
}
```

So, here what we see, book name doesn't change after change() call , because passed book reference value points to new Book object [Angular].

Now, lets look at another example:

```
public static void main(String[] args){
   Book book = new Book("Java");
   Book bookNew = book;

   change(book);
   book.getName(); //Prints .Net

}

public void change(Book book){

   book.getName(); //Prints Java
   book.setName(".Net");
}
```

Here, what we see, value of book object gets changed after change() method call. Because book reference value in change() method still contains the address of actual Book object and it will act upon it only.

Java Interview @ RBS
1.What is the difference between PUT and POST? Which request you will use to recharge a mobile phone?
PUT vs POST:

- PUT is used for update operation. While POST is mainly used for Create operation.
- PUT is idempotent. So if we retry a request multiple times, that should be equivalent to single request modification. While POST is not idempotent. So, if we retry a request N times , you will end up having N resources with N different URI's created on server.

Generally, in practice, always use PUT for Update operations and always use POST for create operations.

We will use POST request to recharge a mobile , because we need to create an entry for that mobile number on the server side. And we know, for creation purpose, we use POST.

2.Difference between Collections API and Stream API.
Collection API was introduced in Java 5. While Stream API was introduced in Java 8.
Collection API is used to persist the elements. While Stream API doesn't persist the elements. It just performs operations on elements.

In Collection API, Iterator is used to iterate the collection elements. While in Streams , Spliterator is used to iterate over elements.

3.Is Stream API faster than collection?

Yes, if we are using parallel streams, then performance of streams is better than collections.
As parallel streams use multi cores available in today's hardware, and it uses Fork Join pool and gives better performance.

4.What are the changes in Collections API in Java 8?

Java 8 has made various changes in Collection API:

- Sorting map directly with Comparators
- Iterate over Map easily with forEach.
- Get rid of if-else condition, use getOrDefault method
- Replace and Remove utilities
- Operate directly on values
- Merge maps with merge method
- Performance improvement in HashMap, LinkedHashMap, ConcurrentHashMap

5.Have you used JProfiler? How does it works and why you will use it?

Yes, I have used JProfiler. It is a code profiling tool.
We have to install it and then integrate with any IDE, Eclipse or IntelliJ.

It provides all the information we need with proper views and filters.
It displays the information while the program is running.

It is used to analyze performance bottlenecks like memory leaks, CPU load etc. and is also used to resolve threading issues.

In Eclipse, we use the Profile command to profile the code using JProfiler.
JProfiler is very easy to use and finds problems in the code.

6.What are Microservices?

The main principle behind microservices is to break a single large monolithic system into multiple independent components/processes.

Microservices architecture allows decoupled components to be built and deployed independently to integrate into a single larger system.

These components interact with each other through a standard XML/JSON interface, irrespective of the technologies used to create the component.

7.What is REST API?

REST stands for Representational State Transfer
REST is an architectural style for designing networked applications and developing web services.
REST API uses HTTP protocol methods/verbs for all it's operations.

It resolves around resources where every component is a resource which can be accessed by a common interface using HTTP methods.

8.Have you worked with Spring Boot? Which version?

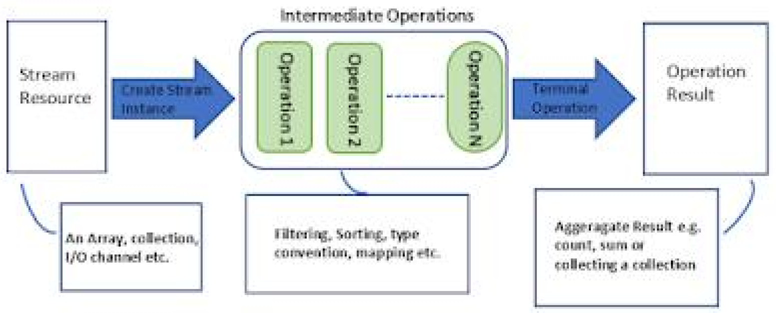
Yes, I've worked with Spring Boot. And version used is Spring Boot 2.

9.What are the features added in Java 8?
Below are all the features added in java 8:

- Functional interfaces, Lambda Expressions
- New Date and Time API
- Streams
- default methods in Interfaces
- static methods in interfaces
- Collection API improvements
- forEach() method in Iterable interface.

10.What are Stream Lifecycle methods?
Stream lifecycle is divided into three types of operations:

- Stream Source : Array, Collection, IO Channel
- Intermediate operations: filter, sort, map, flatMap
- Operation result: List, count, sum


Intermediate Operations
Stream Resource
Create Stream Instance
Operation 1
Operation 2
Operation N
Terminal Operation
Operation Result
An Array, collection, I/O channel etc.
Filtering, Sorting, type convention, mapping etc.
Aggregate Result e.g. count, sum or collecting a collection

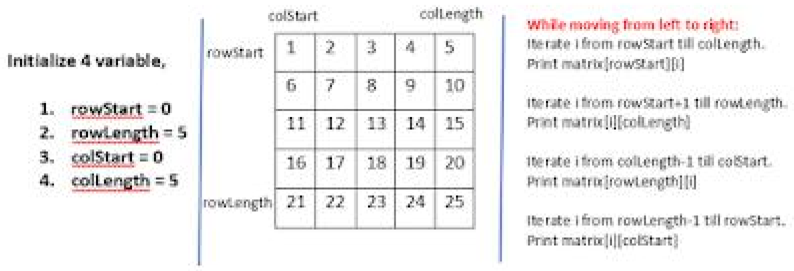
Java Interview @ Virtusa Polaris
1.Find frequency of a character and if more than one characters have same frequency then find character with more ASCII value.

Algorithm to solve this problem:

- Convert String into char array and take a HashMap, count variable and a char variable.

- Iterate this array.
- for every character, store it inside HashMap as key and with it's frequency as value. Also increase it's frequency value by 1. Along with that compare this frequency with count variable and if it is > count , then update count with frequency and also update char variable with that specific character for which we have updated count variable.
- If frequency = count, then check if new character's ASCII code is > ASCII of char. If it is, the update char with new character.
- At last, we will have the required character in char variable with more frequency or with more ASCII value, in case some other char has same frequency.

2.Implement spiral movement in 2D array.



Initialize 4 variable,
1. rowStart = 0
2. rowLength = 5
3. colStart = 0
4. colLength = 5
colStart
colLength
rowStart
rowLength
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
While moving from left to right:
Iterate i from rowStart till colLength.
Print matrix[rowStart][i]
Iterate i from rowStart+1 till rowLength.
Print matrix[i][colLength]
Iterate i from colLength-1 till colStart.
Print matrix[rowLength][i]
Iterate i from rowLength-1 till rowStart.
Print matrix[i][colStart]

Left to Right:

Move variable i from rowStart till colLength. Print data from first row till last column.

Top to Bottom:

Move variable i from (rowStart+1) till rowLength. Print data in last column.
We need to start from rowStart+1, because we already printed corner element in Left to Right printing and no need to include it again. Same treatment for corner elements in other directions.

Right to Left:

Move variable i from colLength - 1 till colStart. Print data in last row.

Bottom to Up:

Move variable i from rowLength - 1 till rowStart. Print data in first column.

After printing all 4 directions , in next iteration, we need to start from second row and second column , so increment rowStart++ and colStart++.
We need to print till second last column and till second last row , so decrement (colLength--) and (rowLength--).

3.What are the approaches for REST API versioning ?
There are multiple approaches for doing versioning in REST API.

Important ones are described below:

URI Versioning:

Using the URI is the most straight forward approach[and also most commonly used]. Though it does violate the principle that URL should refer to a unique resource.

http://api.example.com/v1
http://apiv1.example.com

Versioning using custom Request Header:

e.g.:

Accept-version : v1
Accept-version : v2

Versioning using Accept Header:

e.g.:

Accept : application/vnd.example.v1+json
Accept : application/vnd.example+json;version=1.0

4. Explain Builder Design pattern and in which scenario it is used?

What problem does it solve?:

- Class constructor requires a lot of information.

So, whenever we have an immutable class, then we need to pass all the information/parameters inside the constructor of the class.

When to use Builder Pattern:

- When we have a complex process to construct an object involving multiple steps, then builder design pattern can help us.
- In builder, we remove the logic related to object construction from "client" code and abstract it in separate classes.

5.Difference between split() and StringTokenizer class?

split() vs StringTokenizer:

Using StringTokenizer class, we have the option to use multiple delimiters with the same StringTokenizer object. But that's not possible with split() method.

split() method in String class is more flexible and easy to use. But StringTokenizer class is faster than split() method.

e.g.:

```
StringTokenizer st = new StringTokenizer("a:b:c" , ":");

while(st.hasMoreTokens()){

   System.out.println(st.nextToken());

}
```

StringTokenizer with multiple identifiers:

```
StringTokenizer st = new StringTokenizer("http://100.90.80.3/", "://.");

while(st.hasMoreTokens()){

    System.out.println(st.nextToken());
}
```

Output:

```
100
90
80
3
```

Using split() method:

```
for(String token : "a:b:c".split(":")){

    System.out.println(token);
}
```

6.Why String is immutable in java?

String is immutable due to multiple reasons:

- Due to String Pool facility. The String Pool is managed by String class internally.
- Due to Network Security, as URL is sent in String format.
- Due to thread security. Strings can be shared among multiple threads.
- Class loading mechanism
- Immutability allows string to store it's hashcode and we don't need to calculate hashcode every time we call hashcode() method, which makes it very fast as hashmap keys to be used in HashMap in java.

7.Why the variables defined in try block cannot be used in catch or finally?

When we define any variable in try block and also use that variable in catch or finally, then suppose some exception occurs in try block before the line where that variable is defined. In that case, control goes to catch or finally and we will be using undefined variable in these blocks, because exception occured before the declaration.

That's why variables defined in try block cannot be used in catch or finally.

Java interview @ handygo
1.What are the features of Java 8 version?

Java 8 Features:

- Functional Interfaces
- Lambda Expressions

- Streams
- New Date and Time API
- Changes in Collections API
- Changes in Map classs, HashMap, LinkedHashMap, ConcurrentHashMap
- Added StampedLock

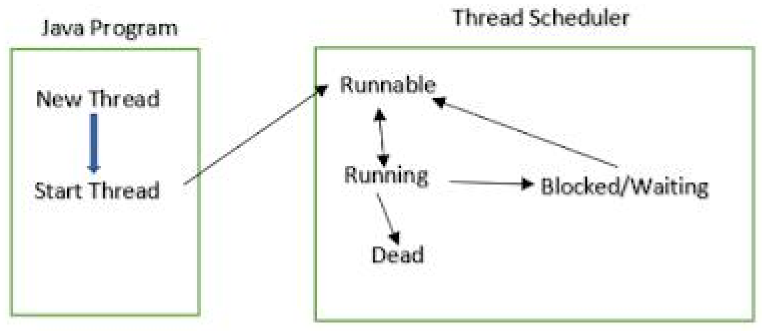
2.How Java Streams are lazy? Explain.
Streams in java contain ternary operations. Like count(), collect(), list() etc. So, intermediary operations on streams are not executed until ternary operation is not called. That's why streams are lazy in nature.

3.Tricky One: I have a functional interface with 3 abstract methods in it. How I'll write lambda expression for it?
This is a tricky question. Actually interviewer wants to see the presence of mind.
Functional interfaces in Java 8 can contain only 1 abstract method. So, lambda expression will not operate on that interface, as it is not functional interface.

4. I want to send a collection to some method as parameter and want to make sure that this collection cannot be updated. How I'll do that?

In this case, we can send unmodifiable collection as the method parameter. In Collections class, we have methods like unModifiableCollection(), unModifiableMap() , unModifiableList() etc.
Using these, we can create and send unModifiable collection as method parameter.

5.What is lifecycle of a thread?



Java Interview @ Rocketalk
1.Can we use String in switch case?

We can use String in switch case from java 7.

Note: switch uses string.equals() method to compare the passed value with case values, so make sure to add a NULL check to avoid NullPointerException on the string passed.

According to java 7 documentation , java compiler generates more efficient bytecode for String in switch statement than chained if-else-if statements.

2. Why char array is preferred over String for storing password?

String is immutable and final and it's value cannot be changed. So it's a security risk, as it takes time to garbage collect that string value and it resides in memory.

So, always use char array so that it's value can be changed to null.

Also , if string object is printed by mistake in some logs statement, then it prints the actual value, while char array prints memory location.

3.Why String is popular HashMap key?

String is immutable and it's hashcode is calculated once and doesn't need to calculate again. It makes it a good candidate as HashMap key.

4.Find maximum repetitions of a number in integer array.

First create a HashMap, a int variable max and an int variable temp.

HashMap: Stores key-value pairs
max : maximum count of occurences of a number
temp: number with maximum occurences till time

Steps to solve this problem are:

- Check whether a key exists in HashMap or not. If it exists , then get its count value, increment it by 1 and put it back and if this new incremented value is > max, then update max with this new value. And also put this new key in temp variable.
- This way, keep iterating and follow step 1 and atlast value stored in temp variable will be the key with max occurences and integer max will give total maximum count of that key.

5.What are the technologies required in Web application development?

In Web application development, technologies are required on client side and server side.

Below is the list of technologies required:

Client side development: HTML, CSS, JavaScript, Angular, Ajax

Server side development: PHP, Java , .NET, Perl, Python

6.What is the difference between Web server and Application server?

Web server is basically used to provide static content. While Application server contain the main business logic.

The clients for Web server are Web Browsers, Mobile app etc.
The clients for Application server are Web Server, Application servers, Mobile devices etc.

Most of the application servers also contain Web server as integral part of the system.

Examples of Web servers are : Tomcat, JBoss
Examples of Application servers are : Weblogic, WebSphere etc.

Java Interview @ Aricent
1.When does deadlock occur?

- Due to nested synchronized block
- Due to calling of synchronized method from another synchronized method

- Trying to get lock on two different objects

Code that causes Deadlock:

```
public class DeadlockDemo{

   public void method1(){

      synchronized(String.class){

         synchronized(Integer.class){

         }
      }
   }

   public void method2(){

      synchronized(Integer.class){

         synchronized(String.class){

         }
      }

   }

}
```

2.What is LiveLock?
When all the threads are blocked or unable to proceed due to unavailability of required resources, then it is known as LiveLock.

1. It occurs when all threads call Object.wait(0) on an object with 0 as parameter. The program is live-locked and cannot proceed until one or more threads call Object.notify() or Object.notifyAll() on the relevant objects.
2. When all the threads are stuck in infinite loops.

3.Is there any way to find a deadlock has occured in java?

Yes. There is a way to find it.
From JDK 1.5, we have java.lang.management package to diagnose and detect deadlocks.
java.lang.management.ThreadBean interface is management interface for the thread system of JVM.

It has methods like findMonitorDeadlockedThreads() and findDeadlockedThreads().

4. Sort list of Employee objects using Designation, then age and then salary [nth level sorting]

```
public class ComparatorChain implements Comparator<Employee>{

   private List<Comparator<Employee>> listComparators;

   public ComparatorChain(Comparator<Employee>... comparators){
      this.listComparators = Arrays.asList(comparators);
   }

   @Override
```

```
    public int compare(Employee emp1, Employee emp2){

        for(Comparator<Employee> comparator : listComparators){
            int finalResult = comparator.compare(emp1, emp2);
            if(finalResult !=0){

                return finalResult;
            }

        }
        return 0;
    }

}
```

DesignationComparator.java:

```
public class DesginationComparator implements Comparator<Employee>{

    @Override
     public int compare(Employee emp1, Employee emp2){

         return emp1.getDesignation().compareTo(emp2.getDesignation());
    }

}
```

AgeComparator.java

```
public class AgeComparator implements Comparator<Employee>{

    @Override
     public int compare(Employee emp1, Employee emp2){

        return emp1.getAge() - emp2.getAge();
    }

}
```

SalaryComparator.java:

```
public class SalaryComparator implements Comparator<Employee>{

    @Override
    public int compare(Employee emp1, Employee emp2){

        retrurn emp1.getSalary() - emp2.getSalary();
    }
}


public class ListObjectsComparisonAndSortingExample{

    public static void main(String[] args){
        List<Employee> employees = new ArrayList<Employee>();

       employees.add(new Employee("Mittal", "Developer", 35, 100000));
       // Add more Employee objects

      Collections.sort(employees, new ComparatorChain(
         new DesignationComparator(),
         new AgeComparator(),
         new SalaryComparator())
       );

    }

}
```

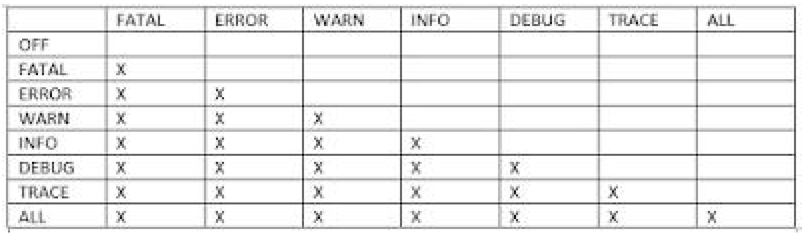
5. How to change logs from Info to Debug using log4j?

In Log4j, we have multiple methods like debug(), info() , error() which can be called based on some condition and after that this Log API will print only that types of logs.

So, whenever we need to change logs from Info level to Debug level, we can easily do that by switching the call from info() to debug() as shown in example below:

```
static Logger logger = Logger.getLogger(MyClass.class.getName()); // Creating logger instance

logger.info("info"); // Suppose , initially it was printing info logs

logger.setLevel(Level.DEBUG); From now on, only debug, info, warn, error and fatal logs will be printed.
But trace logs will not get printed.
logger.debug("debug");
logger.error("error");
logger.trace("trace");//it will not get printed.
```

6.What is the Log4j log level hierarchy order?



	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
OFF							
FATAL	X						
ERROR	X	X					
WARN	X	X	X				
INFO	X	X	X	X			
DEBUG	X	X	X	X	X		
TRACE	X	X	X	X	X	X	
ALL	X	X	X	X	X	X	X

From above diagram, we can see that for WARN, FATAL, ERROR and WARN are visible.
And for OFF, nothing will be visible.

7.What is a Daemon thread? How it works?
Daemon thread acts like service providers for other threads running in the same process.
Daemon threads will be terminated by JVM when there are no other threads running, it includes main thread of execution as well.

To specify that a thread is a Daemon thread, call the setDaemon() method with the argument true.

To determine if a thread is a daemon thread, use the accessor method isDaemon().
Daemon threads are used to provide background support to the user threads.

Example of a daemon thread is Garbage Collection thread. gc() method is defined in System class that is used to send request to JVM to perform garbage collection.

```
public class DaemonThread extends Thread{

   public DaemonThread(){
      setDaemon(true);
   }

   public void run(){

      System.out.println("Is this thread Daemon? - "+isDaemon());
   }

   public static void main(String[] args){

      DaemonThread dt = new DaemonThread();
      dt.start();
   }
}
```

8.Describe two code cases where Race condition occur in java?
Two code scenarios where race condition occur are:

- Check and Act race condition
- Read. Modify, Update race condition

Check and Act race condition:

In this, we take example of creating Singleton instance.

```
if(instance == null){

   return class.getInstance();
}
```

Here, in this code, Suppose we have two threads T1 and T2. When T1 crosses if check, then it goes inside and CPU switches to T2. Now, it T1 is taking more time to create instance, then T2 also checks if statement and find instance as null. It also goes inside if() check and creates another instance.

So 2 instances will be created for Singleton instance.

9.How does Time complexity for get() and put() methods in HashMap is O(1)?

Whenever we search for a key in HashMap or HashSet, it follows these steps:

- Calculate hashcode for the key using their own hash() method.
- This key acts as a m/m address and it is used to find array index/bucket location.
- Then entries in LinkedList are compared.

As, m/m address from array can be get in one step, that's why it is O(1).

But in case, there are 1000 entries in LinkedList in a bucket, then it is not O(1). Then it is based on number of entries in LinkedList. So, in worse case, it is O(n).

10.What is the difference between map and flatMap in Java 8?

map() method is used to map an object or entry in stream to some other value.
While flatMap() method , first applies map() on the entries and then flatten the result.

e.g.:
Suppose we have a string array with entries :
String[] strArray = {"12" , "46"};

And we have to find all the permutations of these strings.

So output with map() method will be : [12, 21] , [46, 64].

While output with flatMap() method will be : [12,21,46,64]

Let's take another example:

```
List<Integer> evens = Arrays.asList(2,4,6,8);
List<Integer> odds = Arrays.asList(1,3,5,7);
List<Integer> primes = Arrays.asList(5,7,11,13,17);

List numbers = Stream.of(evens, odds, primes).flatMap(list-> list.stream()).collect(Collectors.toList());

System.out.println("List = "+numbers);
```

```
O/P: 2,4,6,8,1,3,5,7,5,7,11,13,17
```

Note: This output is possible only with use of flatMap() method.

11.What change has been done in HashMap performance in java 8 and how does that change is implemented?

In HashMap, a customized LinkedList is used for each entry in underlying bucket. From Java 8, if length of LinkedList gets increased to some threshold, then this LinkedList is converted to Binary Search Tree, which increases performance by reducing search time.

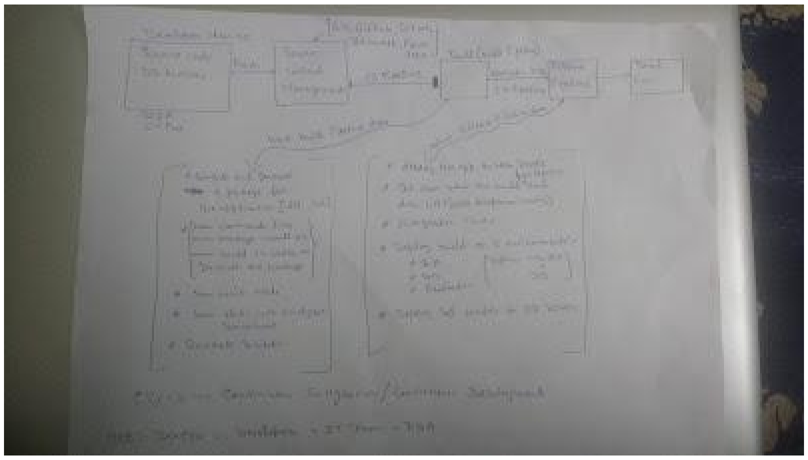
This Binary search tree is constructed using hashcode values of keys. If the hashcode values are different then they are arranged in left or right subtrees. But if the hashcode value is same, in that case there is no such performance improvement.

12:What is the difference between PUT and PATCH?

PUT is used to completely replace a resource at given URI.
While PATCH is used just for partial updation of a given resource at given URI.

Another difference is that while using PUT, we have to send the full payload as the request, while using PATCH , we can only send the parameters which we want to update.

13.Draw CI/CD pipeline process that you follow in your company.



Explanation:

First part is Developer Machine where all the source code and Database scrips are written.
Now all these source code/DB scripts files are pushed to SCM[Source Control Management] like Git/GitHub/GitLab/Azure Repo/Bitbucket etc.

From there it goes to Build pipeline where all this code is compiled and generate a package file for the application [.dll or .jar].

Build Pipeline:

From here CD pipeline starts.

To build the package files, commands like "mvn package install" or "mvn build" are run .

Also all the unit tests run here.
It also runs static code analyzer tool like Sonarqube.
It also includes configuration for generating scripts.

Release Pipeline:

Here built package files from build pipeline are deployed on multiple servers like QA server, Development server and Production server.
All the Integration tests run here.

QA team takes the build from here and run User Acceptance Testing.
Also deploys SQL scripts on Database server.

14.What is the upper limit for a payload to pass in the POST method?

GET appends data to the service URI. But it's size should not exceed the maximum URL length. However POST doesn't have any such limit.

So , theoretically , a user can pass unlimited data as the payload to POST method. But, if we consider a real use case, then sending POST with large payload will consume more bandwidth. It will take more time and present performance challenges to the server. Hence, a user should take action accordingly.

15.Suppose our REST API requires an int value and we send String "abcd" in query parameter , then what response we get?

We get error : Not found.

"timestamp": "2019-01-17T18",
"status" : 404,
"error" : "Not found",

"message" : "JSON parse error: Cannot deserialize value of type int from String \"abcd\": not a valid integer value;
nested exception is com.fasterxml.jackson.databind.exc.InvalidFormatException:

16.What are Spring Core annotations?

Spring core annotations are following:

@Qualifier

@Autowired

@Required

@ComponentScan

@Configuration

@Bean

@Lazy

@Value

17. Explain Spring Data JPA vs Hibernate

JPA: Java Persistence API which provide specification for creating, deleting, persisting and data management from java objects to relations [tables] in database.

Hibernate: There are various providers which implement JPA. Hibernate is one of them . So , we have other provider as well e.g.: Eclipse Link etc.

Spring Data JPA: This is another layer on top of JPA which spring provides to make coding easier.

Java Interview @ WDTS [Walker Digital Table Systems]
1.What is Reentrant Lock?
ReentrantLock is a mutually exclusive lock with the same behavior as the intrinsic/implicit lock accessed via synchronization.

ReentrantLock, as the name suggests , possesses reentrant characteristics. That means , a thread that currently owns the lock can acquire it more than once without any problem.

ReentrantLock forces one thread to enter critical section and queues all other threads to wait for it to complete.

Basically, java 5 introduces the concept of a lock. Lock and ReadWriteLock are interfaces in java 5.

And their implementations are ReentrantLock and ReentrantReadWriteLock.

2.How to handle service fallback in MicroServices?

We can use Circuit breaker pattern implementation Netflix Hystrix for handling fallback in microservices.

Actually, in Microservices based arhitecture, there are many applications running in different processes on different machines. And any of these service may be down at some point of time. So to avoid sending all requests to this misroservice, Circuit breaker pattern can be used.

3.What are REST Call End Points?

e.g.: https://api.github.com/users/zellwk/repos?sort=pushed

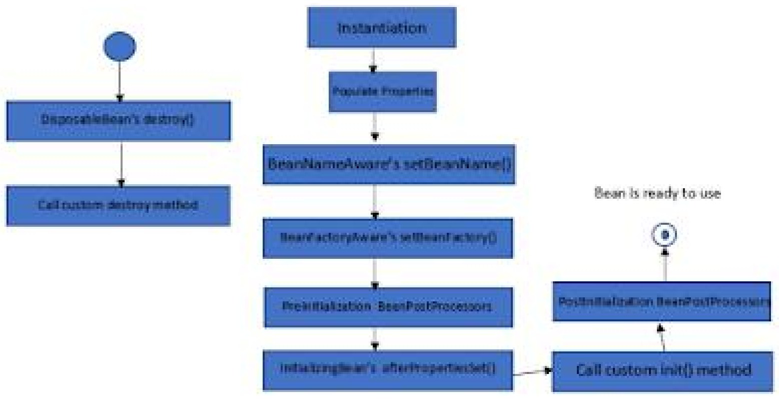
In above URL, https://api.github.com is root endpoint

/users/zellwk/repos is path which determines the resource we request for.

The last part of an endpoint is query parameters. Query parameters give us the option of modifying the request with key-value pairs. They always begin with a question mark [?]. Each parameter pair is then separated with an ampersand [&] , like this:

?query1=value1&query2=value2

4.Explain Lifecycle of Spring Bean.



Instantiation
Populate Properties
BeanNameAware's setBeanName()
BeanFactoryAware's setBeanFactory()
PreInitialization BeanPostProcessors
InitializingBean's afterPropertiesSet()
Call custom init() method
PostInitialization BeanPostProcessors
Bean is ready to use
DisposableBean's destroy()
Call custom destroy method

5.Explain the steps of sending a request from browser and handling it at server side?

Steps :

- First request from browser [client side] will be received by DispatcherServlet
- DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request.
- So now, request transfers to the Controller and then controller will process the request by executing appropriate methods and returns ModelAndView object back to the DispatcherServlet.
- Now DisptacherServlet send the view name to the ViewResolver to get the actual view page.
- Finally DispatcherServlet will pass the Model object to the view page to display the result.

6. Name all the HTTP verbs?

HTTP verbs are:

- **GET**
- **PUT**
- **POST**
- **DELETE**
- **HEAD**
- **PATCH**

7. What is a DispatcherServlet?
DispatcherServlet handles all HTTP requests and responses.

It is front controller in Spring MVC based applications. DisptacherServlet uses it's own WebApplicationContext which is a child of ApplicationContext created by ContextLoaderListener.

8.When we override hashcode() method, then how to retrieve the actual default hash code value for that object?
Just use System.identityHashCode(object);
The value returned by default implementation of hashcode() is called identity hash code.

Identity hashcode is usually the integer representation of the memory address.

Hashcode of an object is a 32-bit signed int that allows an object to be managed by hash-based data structure.

9.Write a logic of two threads printing question and answer one-by-one.

```
class Chat{

   boolean flag = false;

   public synchronized void Question(String message){
      if(flag){
         try{
            wait();
         }
         catch(InterruptedException ie){
             ie.printStackTrace();
         }
      }

      System.out.println(message);
      flag = true;
      notify();
   }

   public synchronized void Answer(String message){
      if(!flag){
         try{
            wait();
         }
         catch(InterruptedException ie){
            ie.printStackTrace();
         }
      }

      System.out.println(message);
      flag = false;
      notify();
   }
}


class Question implements Runnable{

   Chat chat;
   String[] str = {"Hi", "How are you?", "I'm also doing fine"};

   public Question(Chat c1){
      this.chat = c1;
      new Thread(this, "Question").start();
   }


  public void run(){

     for(int i = 0; i< str.length; i++){
        c1.question(str[i]);
      }
```

```
    }

}


class Answer implements Runnable{

    Chat chat;
    String[] str = {"Hi", "I'm good", "Great"};

    public Answer(Chat c1){
        this.chat = c1;
        new Thread(this, "Answer").start();

    }

    public void run(){

        for(int i=0; i<str.length; i++){
            c1.answer(str[i]);
        }
    }

}


public class Test{

    Chat chat =  new Chat();
    new Question(chat);
    new Answer(chat);
}
```

Technical Architect interview @ Xebia
1.What is distributed transaction? What architectures are available for performing distributed transactions?

Simple Transaction:

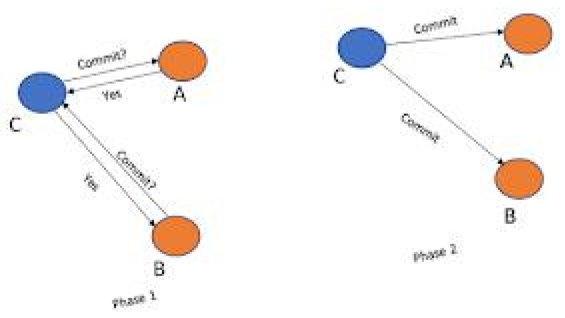
A transaction is a logically atomic unit of work which may span multiple database queries. They ensure locks over resources they acquire in order to maintain the consistency of the database.
All this is done with the help of ACID properties.

Distributed Transaction:

A distributed transaction would do what a transaction would do but on multiple databases.
In a distributed scenario, the architecture would be split into services like handling service, payment gateway service etc who would house a respective database and all the actions would be performed in the respective databases.

One way of doing is the Two phase commit.
In Two phase commit, we have a controlling node which houses most of the logic and we have a few participating nodes on which the actions would be performed.
Prepare Phase:
In this phase, controlling node would ask all the participating nodes, if they are ready to commit. The participating nodes would then respond in yes or no.

Commit Phase:
Then, if all the nods have replied in affirmative, the controlling node would ask them to commit, else if one node replies in negative, it'll ask them to rollback.



Drawbacks of Two Phase Commit:
- Whole logic gets concentrated in single node and if that node goes down, the whole system fails.
- The whole system is bound by slowest node, since any ready node will, have to wait for response from the slower node which is yet to confirm it's status.

Another approach is Saga Pattern:
Saga pattern is one of the ways by which we ensure data consistency in a distributed architecture. But there is absence of atomicity in Saga pattern.

For example , in the e-commerce website, we first do the transaction pertaining to the customer selection. Once that is complete, we start selecting a product and so on. So all these constituent transactions together will be known as a Saga.

A successful Saga looks something like this:
- Start Saga
- Start T1
- End T1
- Start T2
- End T2
- Start T3
- End T3
- End Saga

But things do not go straight. Sometimes, we might not be in position to perform a transaction in the middle of the saga. At that point, the previously successful transactions would've already committed. So, apart from not continuing with the Saga, we also need to undo whatever changes we may have already committed. For this, we apply compensatory transactions.
Thus for each transaction Ti, we implement a compensatory transaction Ci, which tries to semantically nullify Ti.

It's not always possible to get back to the same state. e.g. if Ti involved in sending the email, we can't really undo that. So, we send a corrective email which semantically undoes Ti. So a failed saga looks something like this.

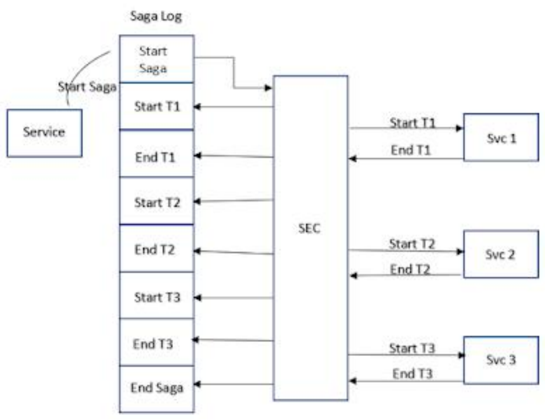
- Begin Saga
- Start T1

- End T1
- Start T2
- Abort Saga
- Start C2
- End C2
- Start C1
- End C1
- End Saga

There can be various ways of implementing Saga pattern, but to actually implement it in a scalable manner ,we can introduce a central process aka Saga Execution Controller or SEC.

SEC is merely a process that handles the flow of execution of transactions or compensatory transactions. It helps us centrally locate the logic of execution.

Another important constituent in order to implement our form of saga pattern is Saga log.
Just like a database log, it's a basic, durable but distributed source of information.Every event that SEC executes is logged in Saga log. A good example of that could be a Kafka logic.



What if our SEC fails down. The solution for that is , we can launch another SEC and it will start from where the first SEC left off.

2.How to write custom annotation?

Java annotations are mechanism for adding metadata information to our source code.
Annotations were added in java 5.
Annotations offer an alternative to the use of XML descriptors and marker interfaces.

We can attach them to packages, classes, interfaces and methods but they have no effect on the execution of the program.

@interface keyword is used to declare an annotation.

Class level annotation:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.Type)
public @interface JsonSerializable{

}
```

In above code, I have defined a class level annotation. @Target and @Retention are used to define metadata [target and scope respectively].

ElementType.TYPE means, it is applied on types (classes).
RetentionPolicy.RUNTIME means, annotation has runtime visibility,

Field Level annotation:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface JsonElement{
   public String key() default "";
}
```

This annotation declares one String parameter with name key and an empty string as the default value.

Method Level annotation:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Init{

}
```

When creating custom annotation for methods, keep in mind that the methods (on which these annotation are applied) cannot have any parameter and cannot throw any exception.

Now, we can simply apply these annotations on classes, fields and methods as shown below:

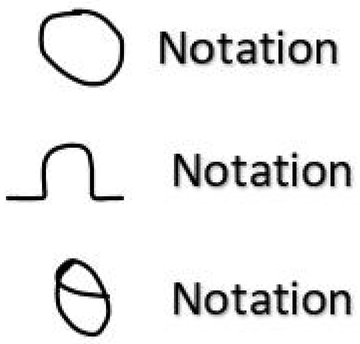
```
@JsonSerializable
public class Student{

   @JsonElement
   public String name;

   public String age;

   @Init
   private void getName(){

   }

}
```

This getName() method is called before serialization. And only name field be serialized.
And as we have made this method private, so that we can't initialize our object by calling it manually.

3.What are Asymptotic notations? What is their use?

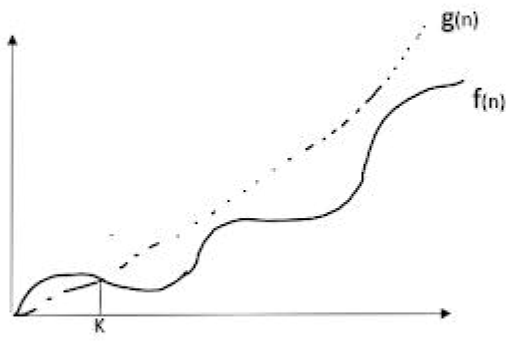
Asymptotic notations are used to calculate running time complexity of an algorithm.

Below are the commonly used asymptotic notations:



Big Oh notation , O:
The notation O(n) is the formal way to express the upper bound of an algorithm's running time.
It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

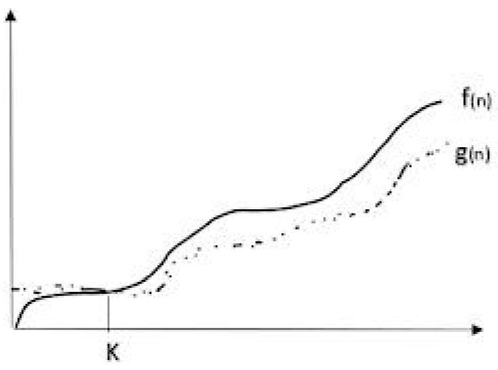
For example , for a function f(n)
o(f(n)) = {g(n) : there exists c > 0 and n theta such that f(n) <= c.g(n) for all n > n theta. }



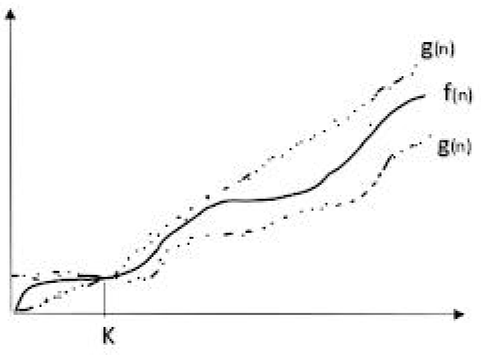
Omega Notation:
The notation Omega(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

For example, for a function f(n):

$\Omega(f(n)) \geq \{ g(n)$: there exists $c > 0$ and n_0 such that $g(n) \leq c.f(n)$ for all $n > n_0$. }



Theta Notation:
The notation theta(n) is the formal way to express both the lower bound and upper bound of an algorithm's running time. It is represented as follows:



4.Write some HTTP error codes with their meaning.

1XX : Informational
- 100 : Continue

2XX : SUCCESS
- 200 : OK
- 201 : Created
- 202 : Accepted
- 204 : No Content

3XX : Redirection

4XX : Client Error
- 400 : Bad Request
- 401 : Unauthorized
- 402 : Payment Required
- 403 : Forbidden
- 404 : Not Found

5XX : Server Error

- 500 : Internal Server Error
- 501 : Not Implemented
- 502 : Bad Gateway
- 503 : Service Unavailable [Website's server is simply not available]

5.What are the different ways of managing REST API versioning? And how do you manage the versioning for some update() method for the new client?

There are multiple ways of managing the REST API versioning:

- Through a URI path: We include the version number in the URI path of the endpoint.
e.g.: api/v1/shares
- Through query parameters: We pass the version number as a query parameter with a specified name e.g.: api/shares?version=1
- Through custom HTTP headers: We define a new header that contains the version number in the request.
- Through a content negotiation: The version number is included in the "Accept" header together with the accepted content type.

Suppose we have an update method with endpoint given below. Initially, it was accessible under /v1.0 path. Now, it is available under /v1.1/{id} path.

```
@PutMapping("/v1.0")
public ShareOld update(@RequestBody ShareOld share){

   return (ShareOld)repository.update(share);

}

@PutMapping("/v1.1/{id}")
public ShareOld update(@PathVariable("id") long id,  @RequestBody ShareOld share){

   return (ShareOld)repository.update(share);

}
```

And if have a GET mapping that remains same for both versions, then , we can write GET mapping as:

```
@GetMapping("/v1.0/{id}", "/v1.1/{id}")
public Share findByIdOld(@PathVariable("id") long Id){

   return (Share)repository.findById(id);
}
```

In this, as we have 2 different versions of our API, we need to create two Docket objects using api() method call.
e.g.:

```
@Bean
public Docket swaggerShareApi10(){

   return new Docket(DocumetationType.SWAGGER_2)
                .groupName("share-api-1.0")
                .select()                          .apis(RequestHandlerSelectors.basePackage("pl.piomin.service
s.versioning.controller"))
.paths(regex("/share/v1.0.*"))
.build()
.apiInfo(new ApiInfoBuilder().version("1.0").title("Share API").description("Documentation Share API
v1.0"));

}
@Bean
public Docket swaggerShareApi11(){

   return new Docket(DocumetationType.SWAGGER_2)
                .groupName("share-api-1.1")
                .select()                  .apis(RequestHandlerSelectors.basePackage("pl.piomin.services.versio
ning.controller"))
.paths(regex("/share/v1.1.*"))
.build()
.apiInfo(new ApiInfoBuilder().version("1.1").title("Share API").description("Documentation Share API
v1.1"));

}
```

Now, when we launch Swagger UI, it shows us the dropdown displaying both versions and we can easily switch between them.

6.What is the difference between MongoDB and Cassandra?

- MongoDB is free and open source , cross platform , document oriented database system. While Cassandra is open source, distributed and decentralized , column-oriented database system.
- MongoDB does not have triggers while Cassandra has triggers.
- MongoDB has secondary indexes while Cassandra has restricted secondary indexes.
- Cassandra uses a selectable replication factor while MongoDB uses a master-slave replication factor.
- MongoDb is used when we need to store data in JSON style format in some documents which consists of key-value pairs. While Cassandra is used as decentralized database for big data.

7.What are the steps for using MongoDB in java project?

Steps for using MongoDB are described below:

- Need to add following dependencies in build.gradle file:
- compile group: 'org.mongodb', name: 'mongodb-driver', version: '3.11.0'
- compile group: 'org.mongodb' name: 'bson', version: '3.11.0'
- compile group: 'org.mongodb' name:'mongodb-driver-core', version: '3.11.0'
- compile group: 'org.mongodb' name:'mongo-java-driver', version: '3.11.0'
- Specify the mongo DB URL and mongo DB name in application.yml file.
- Create MongoClient instance using MongoCredential, MongoClientOptions.Builder and a list of Mongo server addresses.
- Create MongoCollection bean.
- Create a class implementing HealthIndicator interface overriding the health() method.
- Now in the DAO implementation class, call MongoCollection methods such as find(), insertOne(), countDocuments(), count(), bulkWrite() etc.

Coding interview questions in java

1.Write the code for custom BlockingQueue in java.

There are two custom implementations of BlockingQueue: Using Synchronization and using Lock and Condition objects.

Using synchronization:

```
public class CustomBlockingQueue{

    int putPtr, takePtr;
    Object[] items;
    int count = 0;

    public CustomBlockingQueue(int length){

        items = new Object[length];
    }

    public void synchronized put(Object item){

        while(count == items.length()){
            try{
              wait();
            }
            catch(InterruptedException e){

            }
            items[putPtr] = item;
            if(++putPtr == items.length)
                putPtr = 0;

            count++;
            notifyAll();

        }

    }

    public void synchronized take(){

        while(count == 0){
            try{
                wait();
            }
            catch(InterruptedException e){

            }
        }
        Object item = items[takePtr];
        if(++takePtr == items.length)
            takePtr = 0;

        --count;
        notifyAll();
    }

}
```

Using Lock and Condition object:

```
public class CustomBlockingQueue{
```

```
final Lock lock = new ReentrantLock();
final Condition notFull = lock.newCondition();
final Condition notEmpty = lock.newCondition();

public Object[] items = new Object[100];
int putPtr, takePtr, count;


public void put(Object item) throws InterruptedException{
    lock.lock();
    try{
        while(count == items.length)
            notFull.await();
        items[putPtr] = item;
        if(++putPtr = items.length)
            putPtr = 0;
        ++count;
        nonEmpty.signal();

    }
    finally{
        lock.unlock();
    }


}

public Object take() throws InterruptedException{
    lock.lock();
    try{
        while(count == 0)
```

```
            notEmpty.await();
          Object x = items[takePtr];
          if(++takePtr == items.length)
            takePtr = 0;
          count--;
          notFull.signal();
          return x;

        }
        finally{
          lock.unlock();

        }
    }
}
```

2. Write the code for custom ArrayList in java.

```
public class CustomArrayList{

    public Object[] items;

    public int ptr;

    public CustomArrayList(){
        items = new Object[100];
    }

    public void add(Object item){
        if(items.length - ptr == 5){
            increaseListSize();
        }
        items[ptr++] = item;

    }

    public Object get(int index){
        if(index < ptr)
            return items[index];
        else{
            throw new ArrayIndexOutOfBoundsException();
        }

    }

    private void increaseListSize(){
        items = Arrays.copyOf(items, items.length*2);
    }
}
```

3.Write the code for implementing own stack.

```
public class CustomStack{

    private in maxSize;
    private long[] stackArray;
    private int top;

    public CustomStack(int size){
        maxSize = s;
        stackArray = new long[size];
        top= -1;
    }

    public void push(long l){
        stackArray[++top] = l;

    }

    public long pop(){
        return stackArray[top--];

    }

    public long peek(){
        return stackArray[top];

    }

    public boolean isEmpty(){
        return (top == -1);
    }
}
```

4.Write the code for implementing own Queue.

```
public class CustomQueue{

    private int[] arrQueue;
    private int front , rear;

    public CustomQueue(int size){

        arrQueue[size] = new int[];
        front = rear = -1;
    }

    public void insert(int item){

        if(rear == -1){
            front = rear = 0;
        }
        else{
            rear++;
        }

        if(arrQueue.length == rear)
            increaseQueueSize();

        arrQueue[rear] = item;

    }



    public int remove(){
        if(arrQueue.length == 0)
            throw new NoSuchElementException();

        int elem =  arrQueue[front];

        if(front == rear){
            front = rear = -1;
        }
        else
            front++;
        return elem;
    }


    public void increaseQueueSize(){
        arrQueue = Arrays.copyOf(arrQueue, arrQueue.length*2);


    }


}
```

Capgemini Java 3 to 8 Years Experience Interview
1. Tell me about yourself ?

Ans : Give your brief introduction.

2. Explain about your current project?

Ans : It's quite easy to describe your projects and your key role on this project. But, be careful and get ready about the functionality when you are describing your working module / part of the project. Show your confidence that you have done the major part and you can face challenges in future.

3. How many types of literals are there in JAVA ?

Ans : The literals means the value you are assigning to variable. You can specify the below types of literal in java.As per the primitive data types(int,short,long,float,double,boolean,char ,etc there is respective literal. Some literal needs to be ended with a specific character.Read More.

```
        long var=20L; //specify L or l for long literal
int var=20; // If not mentioned any character then its can be short or int
char var='A';
float var=10.44f; //specify f or F for float literal
double var=10.44;
boolean var=true; //or false
```

4. What is meant by Garbage collection ?

Ans : Garbage collection is a automatic feature of java for cleaning the unused object from heap. It helps to developer for releasing the reserved memory without any extra effort by developer. It helps developer to save time and extra mental tension for handling object allocation in memory. When there is no reference to an object found, it will clean that object from memory . You can run the garbage collection explicitly by using System.gc() .

5. Diffference between string s= new string (); and string s = "Hi Dude"; ?

Ans : Both statements are different to each other. Always ‘new’ keyword is used to create object.

String s=new String(); // This statement creates new object in heap. S is the object here.
String s=”Hi Dude” ; // This statement do not create object, its creating reference and its storing in String Constant Pool.S is the reference here.

6. What is singleton class? where is it used ?

Ans : View Answer

7. What is the difference between JSP and Servlets ?

Ans : As simple JSP is pre-compiled but Servlets are not. JSP is specially use for displaying/populating the data on browser. If we take an example of MVC architecture JSP plays the role of View (V). But, Servlets are used to handle the request and process the business logic. In MVC architecture Servlets are knows as Controller (C) .

8. What is the difference in using request.getRequestDispatcher() and context.getRequestDispatcher()?

Ans : Both are taking String parameter, but request.getRequestDispatcher() will dispatch the request inside the application. Where as context.getRequestDispatcher() will dispatch the request outside the context also. If you are using absolute path for dispatching then both are similar.

9. How the JSP file will be executed on the Server side ?

Ans : First its converted to java file. Then its compiled by java compiler and creates the .class file.Now once you get the .class file you can execute it. Internally a JSP converts into respective servlet.

Conversion => Compilation => Execution

10. What is ActionServlet ?

Ans : ActionServlet provides the controller in struts application with MVC (Model-View-Controller) Model 2. ActionServlet is a sub class of javax.servlet.http.HttpServlet.It has few methods like doGet(),doPost(),destroy(),etc.

11. What is Struts Validator Framework ?
Ans : Struts provides a convenient way to validate. Basically we are using two xml configuration file for configuring such as Validator.xml and validation-rule.xml . validation-rule.xml defines the rule of validation like number format validation,email validation. Apart from this ActionForm having validate() method which we can implement for validating the form data.

12. What is the difference between the Session and SessionFactory in hibernate ?

Ans : Session and SessionFactory are playing a prominent role in hibernate. SessionFactory is used to create Session and its created once during starting of the application. You can have only one SessionFactory per application. SessionFactory is also called 2ndlevel cache. But, Session could be many per application. Session is being created by using SessionFactory object.Session is called 1st level cache.

13. What is HQL ?

Ans : HQL stands for Hibernate Query Language. Its fully object oriented and quite similar with SQL.It supports association and joins for effective entity relationship.

Java Interview @ Fresher Level
1.What is Polymorphism and many types of polymorphism are there?

It is one of the OOPS concepts that allows us to perform single action in different ways.
Polymorphism is the capability of a method to do different things based on the object that it is acting upon.

Types of Polymorphism:

- Runtime Polymorphism or Dynamic Polymorphism [Method Overloading]
- Compile time polymorphism or static polymorphism [Method Overloading]

2.What is static binding and dynamic binding?

Static binding or Early binding:

The binding which can be resolved at compile time by compiler is known as static or early binding. The binding of static , private and final methods is compile-time. Because these methods can't be overridden and the type of the class is determined at the compile time.

Dynamic binding or Late binding:

When compiler is not able to resolve call/binding at compile time, such binding is known as dynamic binding or late binding.
Method overriding is the perfect example of dynamic binding.

3.What are the use of super keyword?

The super keyword refers to the object of immediate parent class.

Use of super keyword:

1). To access the data members of parent class when both the child and parent class have member with same name.
2). To explicitly call no-arg and parameterized constructor of parent class.
3). To access the method of parent class when child class has overridden that method.

4.What is the difference between implements Runnable and extends Thread?

There are few differences which are described as below:

- Inheritance Option
- Loosely-coupled
- Functions overhead
- Thread represents "how a thread of control runs". And Runnable represents "what a Thread runs".

Inheritance Option: Means, If class implements Runnable interface, it can extends one class as well. But if class extends Thread, then it cannot extend any other class.

Loosely-coupled: Runnable object can be used in multiple threads. And a Runnable can be changed independently.

Functions Overhead: If we extends Thread class just to run a task, then there will be overhead of all Thread functions.

5.What is difference between user thread and daemon thread?

When we create a thread in java application, it is called user thread.
Daemon thread is a thread that acts as a service provider for other threads running in the same process. e.g. GC.
Daemon thread is terminated by JVM, when there is no other thread running.

6.Why will we take Daemon thread in our application?

If we take a user thread and if it is polling some websites continuously, then if we exit the app, then user thread keeps running and it will not let JVM shutdown.

But, if we make it a daemon thread , then this thread will be terminated by JVM, when there is no other thread running.

7.What do you understand about thread priority?

Every thread has a priority. Usually higher priority thread gets precedence in execution , but it depends upon Thread Scheduler implementation that is OS dependent.

We can specify thread priority using setPriority() method. We can use getPriority() method to get priority.

Thread priority is an int whose value varies from 1 to 10 where 1 is the lowest priority thread and 10 is the highest priority thread.

There are 3 variables defined in Thread class for priority:

- MIN_PRIORITY
- NORM_PRIORITY : This is the default priority. It has value 5. Default priority of a thread depends upon priority of parent thread.
- MAX_PRIORITY

8.What is thread scheduler and time slicing?

Thread scheduler is OS service that allocates CPU time to the available threads.
Time slicing is the process of divide the available CPU time to the available runnable threads.

9.Write down the definition of Iterable interface.

```
public interface Iterable{

  public Iterator<T> iterator();
}
```

10.How does HashSet work?

When we create HashSet instance, it internally creates instance of HashMap in its constructor. So when we add a duplicate key, that key is compared in HashMap entries. Actually, it is put in HashMap.

So, when we call add(3), then the value 3 is stored as key in HashMap and some dummy value [new Object] is stored as value.

So, in HashSet add() method is defined as:

```
public boolean add(E e){

        return map.put(e, PRESENT) == null;
}
```

Note: map.put() returns null, if it adds key-value pair. If key already exists , it returns value.
So, if key is not present in HashMap, HashMap returns null and add() method of HashSet returns true.Else it returns false.

11.What copy technique is originally used by HashSet clone() method?

There are two copy techniques in every object oriented programming language: deep copy and shallow copy.

To create a clone or copy of the Set object, HashSet internally uses shallow copy in clone() method, the elements themselves are not cloned. In other words, a shallow copy is made by copying the reference of the object.

12.What is the difference between HashSet and TreeSet?

There are multiple differences between HashSet and TreeSet which are described below:

- Ordering of the elements
- Null Value: TreeSet doesn't allow null value.
- HashSet implements Set interface while TreeSet implements NavigableTreeSet interface.
- HashSet uses equals() method for comparison while TreeSet uses compareTo() method for comparison.

Note: Underlying data structure of TreeSet is Red-Black Tree which is a BST and thus is sorted. For it to be sorted, it uses comparator. The default comparator is not null safe, that's why TreeSet doesn't allow null value.

So, when we call TreeSet.add(null); , It compiles but at runtime, it throws NullPointerException.

13.How fail fast iterator come to know that the internal structure is modified?

Iterator read internal data structure (object array) directly. The internal data structure (i.e. object array) should not be modified while iterating through the collection.

To ensure this, it maintains an internal flag "mods". Iterator checks the mods flag , whenever it gets the next value (using hasNext() method and next() method). Value of mods flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException.

14.What is fail safe iterator?

Fail safe iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.
Any structural modification done to the iterator affects the copied data structure. So , original data structure remains structurally unchanged.
Hence, no ConcurrentModificationException is thrown by the fail safe iterator.

15.What is the difference between Collections and Collection?

java.util.Collections is a class which contains static methods only and most of the methods throw NullPointerException if object or class passed to them is null.

java.util.Collection is an interface. Which is the base interface for all other collections like List, Map, Set etc.

16.What is the difference between Iterator and ListIterator?

We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.

Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.

17.How many ways are there to traverse or loop Map, HashMap , TreeMap in java?

We have 4 ways to traverse:
- Take map.keySet() and loop using foreach loop
- Take map.keySet() and loop using Iterator
- Take map.entrySet() and loop using foreach loop
- Take map.entrySet() and loop using iterator.

18.Name the types of SQL databases.

There are multiple SQL databases. Few are listed below:
- MySQL
- SQL Server
- Oracle
- Postgres

19.What is the difference between MySQL and SQL Server?

There are multiple differences between MySQL and SQL server databases:
- MySQL is an open source RDBMS [owned by Oracle] , whereas SQL Server is a microsoft product.
- MySQL supports more programming languages than supported by SQL server. e.g.: MySQL supports Perl, Scheme, Tcl, Eiffel etc. which are not supported by SQL server.
- Multiple storage engine[InnoDB, MyISAM] support makes MySQL more flexible than SQL Server.
- While using MySQL, RDBMS blocks the database while backing up the data. And the data restoration process is time-consuming due to execution of multiple SQL statements. Unlike MySQL, SQL server does not block the database while backing up the data.
- SQL server is more secure than MySQL. MySQL allows database file to be accessed and manipulated by other processes at runtime. But SQL server does not allow any process to access or manipulate it's database files or binaries.
- MySQL doesn't allow to cancel a query mid-execution. On the other hand, SQL server allows us to cancel a query execution mid-way in the process.

20.Why should we never compare Integer using == operator?

Java 5 provides autoboxing/unboxing.

So, we store int to wrapper class Integer. But we should not use == operator to compare Integer objects. e.g.:

```
Integer i = 127;
Integer j = 127;
```

i == j will give true.

```
Integer ii =  128;
Integer jj = 128;
```

ii == jj will give false.

It is so because, Integer.valueOf() method caches int values ranging from -127 to 127. So, between this range, it will return same object.
After that, it will create new object.

Note:
```
Integer i = 127;
Integer j = new Integer(128);
```

Now, == operator will give false. As , new operator will create a new object.

21.Why to use lock classes from concurrent API when we have synchronization?

Lock classes in concurrent API provide fine-grained control over locking.

Interfaces: Lock, ReadWriteLock
Classes: ReentrantLock, ReentrantReadWriteLock

22.What are the methods for fine-grained control?

ReentrantLock provides multiple methods for more fine-grained control:
- isLocked()
- tryLock()
- tryLock(long milliseconds, TimeUnit tu)

The tryLock() method tries to acquire the lock without pausing the thread. That is, if the thread couldn't acquire the lock because it was held by another thread, then it returns immediately instead of waiting for the lock to be released.

We can also specify a timeout in the tryLock() method to wait for the lock to be available:

```
lock.tryLock(1, TimeUnit.SECONDS);
```

The thread will now pause for one second and wait for the lock to be available. If the lock couldn't be acquired within 1 second , then the thread returns.

23.What is ReentrantReadWriteLock?

ReadWriteLock consists of a pair of locks - one for read and one for write access. The read lock may be held by multiple threads simultaneously as long as the write lock is not held by any thread.

ReadWriteLock allows for an increased level of concurrency. It performs better compared to other locks in applications where there are fewer writes than reads.

24.What is the difference between Lock and synchronized keyword?

Following are the differences between Lock and synchronized keyword:

- Having a timeout trying to get access to a synchronized block is not possible. Using lock.tryLock(long timeout, TimeUnit tu), it is possible.
- The synchronized block must be fully contained within a single method. A lock can have it's calls to lock() and unlock() in separate methods.

25.Why to use Executor framework?

We can use executor framework to decouple command submission from command execution.

Executor framework gives us the ability to create and manage threads.

There are 3 interfaces defined in executor framework:

- Executor
- ExecutorService
- ScheduledExecutorService

26.How many groups of collection interfaces are there?

There are two groups of collection interfaces:

- Collection
    - List
        - ArrayList
        - Vector
        - LinkedList
    - Queue
        - LinkedList
        - PriorityQueue
    - Set
        - HashSet
        - LinkedHashSet
        - SortedSet
            - TreeSet
- Map
    - HashMap
    - HashTable
    - SortedMap
        - TreeMap

27.What is the definition of Iterable interface?

```
public interface Iterable<T>{

   public Iterator<T> iterator();
}
```

Monitoring the performance of microservice

Question 1:

How do you monitor the performance of microservice?

Answer:

For monitoring the performance of microservice, I have used OpenCensus and Zipkin tools.

OpenCensus is an open-source project started by google that can emit metrics and traces when it integrates within application code to give us a better understanding of what happens when the application is running.

So, means we have to instrument our application. Instrumentation is how our application produce the events that will help us to have a better understanding of a problem when debugging in production.

For using OpenCensus, we need to integrate it into our code.

Instead of having to manually write code to send traces and metrics on the application while it is running, we just use the OpenCensus libraries.

These libraries exist in several programming languages. By using these libraries as frameworks, we can collect commonly used predefined data.

The data that OpenCensus collects is divided into two groups : metrics and traces.

But how can we see the metrics and traces that OpenCensus collects? By using exporters.

Exporters are the mechanism that we use to send those metrics to other sources (also known as backends) like Prometheus, Zipkin, Stackdriver, Azure Monitor etc.

Metrics: These are the data points that tell us about what is happening in the application e.g. : latency in a service call or user input.

Traces: These are the data that can show how the initial call propagates through the system. Traces help you to find exactly where the application might be having problems.

How does OpenCensus help you?

We can tag every request with an unique ID. Having a unique identifier helps us to correlate all the events involved in each user call, creating a single context.

Once we have the context, OpenCensus helps us expand it by adding traces. Each trace that OpenCensus generates will use the propagated context to help you to visualize request's flow in the system. Having traces allows us to know information like how much time each call took and where exactly the system needs to improve. From all the calls generated, we might identify that the user input has data that we didn't consider and that's what causing a delay in storing data in the cache.

Configure OpenCensus:

Maven Configuration:

```
<dependency>

   <groupId>io.opencensus</groupId>
```

```
    <artifactId>opencensus-api</artifactId>
    <version>${opencensus.version}</version>
</dependency>

<dependency>
    <groupId>io.opencensus</groupId>
    <artifactId>opencensus-impl</artifactId>
    <version>${opencensus.version}</version>
</dependency>

<dependency>
    <groupId>io.opencensus</groupId>
    <artifactId>opencensus-exporter-trace-zipkin</artifactId>
    <version>${opencensus.version}</version>
</dependency>
```

Inside main() method:

```
//1. Configure exporter to export traces to Zipkin.
ZipkinTraceExporter.createAndRegister("http://localhost:9411/api/v2/spans", "tracing-to-zipkin-service");
```

OpenCensus can export traces to different distributed tracing stores (such as Zipkin, Jeager, StackDriver trace). We configure OpenCensus to export to zipkin, which is listening on localhost port 9411 and all of the traces from this program will be associated with a service name tracing-to-zipkin-service.

Configure Sampler:

Configure 100% sample rate, otherwise few traces will be sampled.

```
TraceConfig traceConfig = racing.getTraceConfig();
TraceParams activeTraceParams = traceConfig.getActiveTraceParams();
traceConfig.updateActiveTraceParams(activeTraceParams.toBuilder().setSampler(Samplers.alwaysSample()).build());
```

Using the Tracer:

To start a trace, we first need to get a reference to the tracer. It can be retrieved as a global singleton.

```
Tracer tracer = Tracing.getTracer();
```

Create a Span:

To create a span in a trace, we used the tracer to start a new span. A span must be closed in order to mark the end of the span. A scoped span implements AutoCloseable, so when used within a try block in java 8, the span will be closed automatically when existing the try block.

```
try(Scope scope = tracer.spanBuilder("main").startScopedSpan()){

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

        doWork(i);

    }


}
```

Using Zipkin:

Zipkin is a java based distributed tracing system to collect and lookup data from distributed systems.

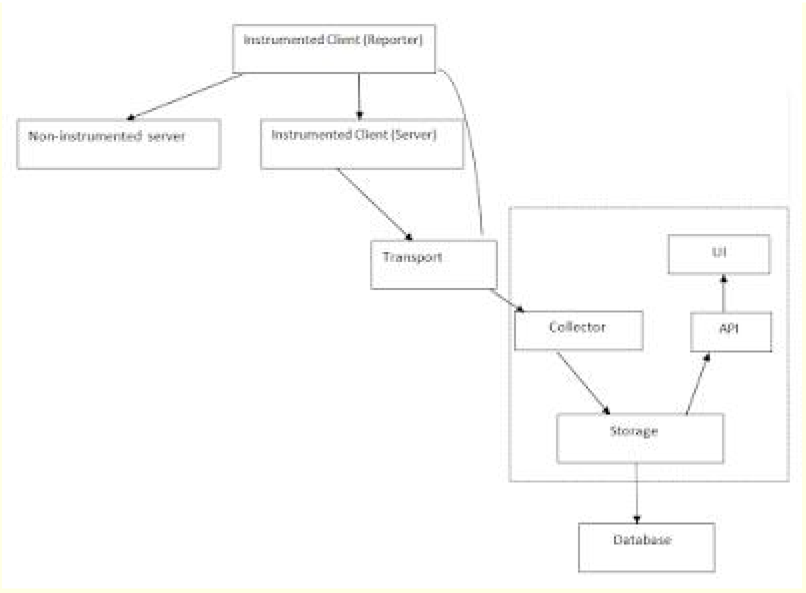
Too many things could happen when a request to an HTTP application is made. A request could include a call to a database engine, to a cache server or any other dependency like another microservice. That's where a service like Zipkin can come in handy.

Zipkin is an open source distributed tracing system based on Dapper's paper from google. Dapper is google's system for its system distributed tracing in production.

Zipkin helps us find out exactly where a request to the application has spent more time.

Whether it is an internal call inside the code or an internal or external API call to another service, we can instrument the system to share a context. Microservices usually share context by correlating requests with a unique ID.

Zipkin Architecture:



Zipkin Components:

- Collector
- Storage
- Search
- Web UI

Using Zipkin:

- Need to add dependency for Zipkin in pom.xml and also add dependency for Zipkin UI.

```
<dependency>

   <groupId>io.zipkin.java</groupId>

   <artifactId>zipkin-server</artifactId>

   <version>2.11.7</version>

</dependency>

<dependency>

   <groupId>io.zipkin.java</groupId>

   <artifactId>zipkin-autoconfigure-ui</artifactId>

   <version>2.11.7</version>

</dependency>
```

- Add @EnableZipkinServer annotation on main Springboot application.

Open application.properties in resources and mention following properties:

- spring.application.name = zipkin-server

- server.port = 9411 (default port)
- spring.main.allow-bean-definition-overriding = true (Used when some other bean with same name has already been defined.)
- Management.metrics.web.server.auto-time-requests = false (If prometheus says, there are to meters with same name but different tags names)

Java Technical Interview @ CandelaLabs
Question1:

What is DispatcherServlet and why it is used for?

Answer:

The DispatcherServlet is the implementation of front controller design pattern that handles all incoming web requests to a spring MVC application.

A front controller pattern is a common pattern in web applications whose job is to receive all requests and route it to different components of application for actual processing.

In Spring MVC, the DispatcherServlet routes web requests to spring MVC controllers.

DispatcherServlet uses handler mapping like @RequestMapping annotation to find correct controller to process a request.

It s also responsible for delegating logical view name to ViewResolver and then sending the rendered response to the client.

Question 2:

What is the root application context in spring MVC? How is it loaded?

Answer:

In Spring MVC, the context loaded using ContextLoaderListener is called the root application context which belongs to the whole application. While the one initialized using DispatcherServlet is actually specific to that servlet.

Technically, spring MVC allows multiple DispatcherServlet in a spring MVC web application and so are multiple contexts , each specific for the specific servlet.

Question 3:

How many types of Dependency injection types are there in Spring?

Answer:

Spring documentation strictly defines only twp types of injection : constructor and setter injection.

However there are more ways to inject a dependency like a field injection , lookup method injection.

Constructor Injection: Enforcing immutability:

In this, a dependent class has a constructor , where all dependencies are set. These will be provided by spring container according to xml, java or annotation based configuration.

e.g.:

```
class Service1{
}
class Service2{
}
class DependentService{
    private Service1 ser1;
    private Service2 ser2;
    public DependentService(Service ser1, Service2 ser2){
        this.ser1 = ser1;
        this.ser2 = ser2;
    }
}
public class Main{
    public static void main(String[] args){
        ApplicationContext container = new ClassPathApplicationContext("spring.xml");
        (DependentService) container.getBean("dependentService");
    }
}
```

spring.xml is:

```
<?xml version = "1.0" encoding = "utf-8"?>
    // bean xmlns namespaces goes here.....
    <bean id = "service1" classs ="eample.service1"/>
    <bean id = "service2" class = "example.service2"/>
    <bean id="deoenddentService" class = "example.DepenentService">
        <constructor-arg type = "example.Service1" ref = "service1"/>
        <constructor-arg type = "example.Service2" ref = "service2"/>
    </bean>
```

Constructor injection using annotation:

```
@Service
class Service1{
}
@Service
class Service2{
}
@Service
class DependentService{
    private final Service1 ser1;
    private final Service2 ser2;
    @Autowired
    public DependentService(Service1 ser1, Service2 ser2){
        this.ser1 = ser1;
        this.ser2 = ser2;
    }
}
public class Main{
    public static void main(String[] args){
        ApplicationContext container = new ClassPathAppplicationContext("spring.xml");
        (DependentService) container.getBean("dependentService");
    }
}
```

Now in spring.xml file:

we do not need to add any <bean>.

Just add <context:component-scan base-package= "example"/>

Drawback of this approach:

We need to use @Qualifier annotation to tell spring, which implementation of a specific interface to use , in case we are using two implementations of an interface.

Higher chance to have circular dependencies.

Note: We can have many more constructors in or class, but only one of them should qualify for dependency injection.

Advantage of this approach: Can be combined with setter injection and field injection.

Setter injection:

With XML:

```
class Service1{
}
class Service2{
}
class DependentService{
   private Service1 ser1;
   private Service2 ser2;
   public DependentService(){
   }
   public void setService1(Service1 ser1){
      this.ser1 = ser1;
   }

   public void setService2(Service2 ser2){
      this.ser2 = ser2;
   }


}
```

XML is:

```
<bean id = "service1" class = "example.Service1"/>
<bean id= "service2" class = "example.Service2"/>
<bean id = "dependentService" class = "example.DependentService">
   <property name="service1" ref="service1"/>
   <property name="service2" ref="service2"/>
</bean>
```

Setter Injection with annotations:

```
class Service1{
}
class Service2{
}
```

```
class DependentService{
    private Service1 ser1;
    private Service2 ser2;
    public DependentService(){
    }
    @Autowired
    public void serService1(Service1 ser1){
        this.ser1 = ser1;
    }
     @Autowired
     public void setService2(Service2 ser2){
        this.ser2 = ser2;
    }
}
```

Note: We can combine setter injection and constructor injection.

Field Injection:

```
@Service
class DependentService{
    @Autowired
    private Service1 ser1;
    @Autowired
    private Service2 ser2;
}
```

Advantages:

- Easy to use. No constructors or setters required.
- Can be easily combined with the constructor and /or setter approach.

Disadvantages:

- A number of dependencies can reach dozens until we notice that something went wrong in the design.

Question 4:

What are java Microservices frameworks?

Answer:

There are multiple java microservices frameworks:

- Spring and Spring boot
- DropWizard
- Eclipse MicroProfile

Spring and Spring Boot:

The framework propagates building the application into a jar and running it on the embedded Tomcat server , making it a perfect combination with Docker to manage a virtualized deployment environment.

DropWizard:

Like Spring boot, DropWizard application is packages into jar file with the jetty server embedded.

DropWizard integrates the tried-and-tested java libraries into a fully functional platform: Jersey for REST, Jackson for JSON, Freemarker for template and Mustache for java based UIs.

DropWizard ships with no built-in dependency injection solution, but integrations exist for Guice and Dagger.

Java Technical interview @ Candela Labs - Round 2
Question 1:

Is the DispatcherServlet instantiated via Application context?

Answer:

No,the DispatcherServlet is instantiated by Servlet containers like tomcat or jetty. We must define the Dispatcher Servlet into the web.xml file.

We also include load-on-startup tag = 1, which means DispatcherServlet is instantiated when you deploy the spring MVC application to Tomcat or any other servlet container.

During instantiation, it looks for a file servlet-name-context.xml and then initializes beans defined in this file.

Question 2:

Do we need spring-mvc.jar in our classpath or is it part of spring-core?

Answer:

The spring-mvc.jar is not part of spring-core, which means that if we want to use spring MVC framework in the java project, we must include spring-mvc.jar in the application's classpath.

In a java web application, spring-mvc.jar is usually placed inside the /WEB-INF/lib folder.

Question 3:

How Spring MVC framework works? How HTTP request is processed?

Answer:

- Client sends a request to a URL.
- That request hit the web container e.g. Tomcat or Jetty. Then it looks into web.xml and find the servlet or filter which is mapped to that URL.
- It then delegates the control to servlet or filter to process the request. Web container(Tomcat) is responsible for creating servlet and filter instances and invoking their various lifecycle methods e.g. init(), service() and destroy(). In the case of HTTP request, HTTPServlet handles that and depending upon the HTTP request method, various do...() methods are invoked by container e.g. doGet(), doPost() to process GET request and POST request.
- To enable spring MVC, we need to declare the DispatcherServlet from spring MVC jar into web.xml. This servlet listens for a URL pattern * as shown below in web.xml, which means all requests are mapped to DispatcherServlet.
- Then DispatcherServlet then passes the request to a specific controller depending on the URL requested. It uses handler mapping [@RequestMapping annotation] or spring mvc configuration file to find out the mapping of request URL to difference controllers. Controller classes are also identified using @Controller and @RestController annotations.
- After processing the request , controller returns a logical view name and model to DispatcherServlet and it consults view resolvers until an actual view is determined to render the output.
- DispatcherServlet then contacts the chosen view e.g. Freemarker or JSP with model data and it renders the output.
- This rendered output is returned to the client as HTTP response. On it's way back, it can pass to any configured filgter as well e.g. Spring security filter chain or filters configured to convert the response to JSON or XML.

Question 4:

Which cache you have used and explain the steps to use that cache?

Answer:

There are multiple types of caches:

- Caffeine
- HazelCast
- Redis
- EHCache
- CouchBase

I have used Redis cache.
Redis is an open source in-memory data structure store used as a database, cache and message broker.

It supports data structures such as strings, hashes, lists, set, sorted set and streams.

Redis has built-in replication.

Redis cache limits:

On 32-bit machine,it can store upto 3 GB of data. When cache size reaches the memory limit, old data is removed to make place for new one.

Annotations used:

- @Cacheable : To cache some values

- @Cacheput : To update the cache
- @CacheEvict : To empty the cache

Maven Dependency:

```
<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>
```

application properties:

```
#Redis config:

spring.cache.type=redis

spring.redis.host-localhost

sprint.redis.port=6379
```

@EnableCaching annotation on SpringBootApplication main class.

```
import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository{

}


@RestController

public class UserController{

   private final Logger log = LoggerFactory.getLogger(getClass());

   private final UserRepository userRepository;

   @Autowired

   public UserController(userRepository userRepository){

      this.userRepository = userRepository;

   }


   @Cacheable(value="users", key="#userId", unless="#result.followers<12000")

   @RequestMapping(value"/{userId}", method=RequestMethod.GET)

   public user getUser(@PathVariable String userId){
        log.info("Getting user with ID {}", userId);

        return userRepository.findOne(Long.valueOf(userId));

    }

}
```

Question 5:

What will be the output of below code?

```
public class StringClass{

    public static void main(int[] args){


        final String str = "First String is computer";

        String str1 = str.replaceAll("First", "Last");

        System.out.println(str);

    }

}
```

Answer:

It says, main type is not found. Because JVM requires a main method with String[] as arguments.

In the same code, if we try replacing str1 with str, then compiler shows exception that str is declared as final and cannot be changed.

Question 6:

What happens internally when we invoke main method?

Answer:

The purpose of main method in java is to be program execution start point.

When we run java.exe,it makes some JNI calls. These calls load the DLL that is really the JVM.

It will parse the command line arguments , creates a new string array in the JVM to hold these arguments , parses the class name that we specify as containing main() method, uses JNI calls to find the main() method, then invokes the main() method passing the parameters list.

Java Technical Architect interview @ Brillio
Question 1:

What is the difference between @Autowired, @Inject and @resource annotations?

Answer:

Resource and Inject belong to java extension package: javax.annotation.Resource and javax.inject.Inject package.

@Autowired annotation belongs to org.springframework.beans.factory.annotation package.

Each of these annotations can resolve dependencies either by field injection or by setter injection.

@Inject and @Autowired have same execution paths:

- Match by Type
- Match by Qualifier
- Match by name

While @Resource has different execution path:

- Match by name
- Match by type
- Match by Qualifier

e.g.:

```
public class FieldResourceInjectionIntegrationTest{

   @Resource(name="namedFile")

   private File defaultFile;

   @Test

   public void givenResourceAnnotation(){

      assertNotNull(defaultFile);

      assertEquals("nameFile.txt", defaultFile.getName());

   }

}
```

This configuration will resolve dependencies using the match-by-name execution path. The bean namedFile must be defined in the ApplicationContextTestResourceNameType application context.

```
@Configuration

public class ApplicationContextTestResourceNameType{

   @Bean(name="namedFile")

   public File namedFile(){

      File namedFile = new File("namedFile.txt");

      return namedFile;

   }

}
```

Failure to define the bean[with that name "namedFile"] in the application context will result in a org.springframework.beans.factory.NoSuchBeanDefinitionException being thrown.

Note: But if we use @Annotation or @Inject, in that case, it works, because execution path of @Annotation and @Inject contains "Match by Type" first.

Question 2:

What are Spring bean scopes?

Answer:

Spring bean scopes are:

- Singleton
- Prototype
- Request
- Session
- GlobalSession
- Application
- WebSocket

Singleton: Only one bean is created with same id per spring container.
But if, we have dependencies as prototype beans, then injection happens when the singleton object is instantiated, so injection just happens once.

Thus the prototype bean will always be the same instance even though it was defined with a prototype scope.

Prototype: Spring container will create a new instance of the object described by the bean each time it receives a request. Prototype bean should be used in environment where there is session management(Stateful) and use singleton scope beans when working on environments without session management(Stateless).

Request: Spering container will create a new instance of the object defined by the bean each time it receives an HTTP request.

Session: Spring container will create a new instance for each HTTP session.

Global Session: It is same like Session , but it works for Portlet based applications.

Question 3:

How can we connect to multiple databases in hibernate?

Answer:

We need to use separate configuration file for connecting to a specific databases, so two configuration files are required to connect to two different databases.

To configure mysql database:

hibernate-mysql.cfg.xml

To configure oracle database:

hibernate-oracle.cfg.xml

In details, mysql configuration will be like this:

```
<?xml version="1.0" encoding="utf-8"?>

// DOCTYPE here......

<hibernate-configuration>
```

```
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">PASSWORD</property>
        <property name="hibernate.connection.url">jdbc:mysql://locallhost:3306/db_name</property>
        <property name="hibernate.connection.username">USERNAME</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <mapping class="domain.EmployeeMySql"></mapping>
    </session-factory>
</hibernate-configuration>
```

In details, oracle configuration will be like this:

```
<?xml version="1.0" encoding="utf-8"?>
// DOCTYPE here......
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property name="hibernate.connection.password">PASSWORD</property>
        <property name="hibernate.connection.url">jdbc:oracle:thin:db_name</property>
        <property name="hibernate.connection.username">USERNAME</property>
        <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="domain.EmployeeOracleSql"></mapping>
    </session-factory>
</hibernate-configuration>
```

And code should be like this:

MySql configuration:

```
private static SessionFactory sessionAnnotationFactory;
sessionAnnotationFactory = new Configuration().configure("hibernate-mysql.cfg.xml");
Session session = sessionAnnotationFactory.openSession();
```

Oracle configuration:

```
private static SessionFactory sessionAnnotationFactory;
sessionAnnotationFactory = new Configuration().configure("hibernate-oracle.cfg.xml");
Session session = sessionAnnotationFactory.openSession();
```

Question 4:

What is the output of below code?

```
class A{
   void methodOne(Double d){
   }
   void methodTwo(Integer I){
   }
class B extends A{
   @Override
   void methodOne(double d){
   }
   @Override
   void methodTwo(Integer I){
   }
}
}
```

Answer:

Compile time error for not overriding proper method.

Question 5:

What is the effect when a transient mapped object is passed onto a session's save method?

Answer:

It will change its state from transient to persistent.

Java Technical interview @ThalesGroup : Round -1

Question 1:

Is the following source code is valid override?

```
class MyClass{
   void add(int i, int j){
   }
}
```

```
class Subclass extends MyClass{
   public void add(int i, int j){
   }
}
```

Answer:

Yes, the source code is valid override, as we can increase the visibility of overridden method in subclass.

Question 2:

Is the following source code is valid override?

```
class MyClass{
   private void add(int i, int j){
   }
}
class SubClass extends MyClass{
   public void add(int i, int j){
   }
}
```

Answer:

Yes, answer is same as for question 1.

But we cannot use below code for above code:

MyClass myClass = new SubClass();

myClass.add(1,2); // add() is private , cannot call.

Question 3:

Is the following source code is valid override?

```
class MyClass{
   static void add(int i, int j){
    }

}



class SubClass extends MyClass{

    static void add(int i, int j){

    }

}
```

Answer:

It is valid, but static methods cannot be overridden. It is called method hiding.

So, when we use below code:

MyClass myClass = new SubClass();

myClass.add(1,2); // Calls add() method of class MyClass.

Question 4:

What are different types of caches in hibernate?

Answer:

Hibernate uses two different caches for objects:

- First-Level cache: It is associated with session object.
- Second-Level cache It is associated with SessionFactory object.

By default, hibernate uses first-level cache on a per transaction basis. Hibernate uses this cache to reduce number of queries it needs to generate within a given transaction.

Question 5:

How does hibernate second-level cache work?

Answer:

Hibernate always tries to first retrieve the objects from session[first-level-cache] and if it fails, then it tries to retrieve them from second-level cache. If this fails again, the objects are directly loaded from the database.

Hibernate's static initialize() method which populates a proxy object, will attempt to hit the second-level cache before going to the database. The hibernate class provides static methods for manipulation of proxies.

Question 6:

What is version checking in hibernate?

Answer:

Version checking is used hibernate when more than one thread try to access same data.

e.g.:

User A tries to update a row in table, in the same time , user B also tries to update the row in a table and click on update after making changes. Now user A clicks on update. This way, changes made by user B are gone.

It is called stale object updation. And in hibernate , we can avoid stale object updation using version checking.

How it works:

Check the version of the row when you are updating the row. Get the version of the row, when we are fetching the row from the table. At the time of updation, just fetch the version number and match with our version number. This way, we can prevent stale object updation.

Question 7:

What is the general contract between hashcode() and equals() method?

Answer:

- Whenever hashcode is invoked on the same object multiple times, it should return same integer value during the execution of a program, provided no info in equals() method is changed. This integer need not be same from one execution of a program to another execution of the same program.
- If two objects are equal according to equals() method, then their hashcodes must be same.
- It is not required that if two objects are unequal , then their hashcodes must be different. However programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

Question 8:

Suppose we have a real time big application and one service is calling another service for fetching real time data. And another service is slow somehow, then what approach will be used to ensure that first service should bet real time data for each request?

Answer:

We can implement a messaging queue between these two services and store every message in that queue and second service can easily consume these messages according to it's speed.

Java Technical interview @ ThalesGroup : Round-2

Question 1:

What are the tools available for testing web services?

Answer:

- SOAP UI Tool
- Poster for Firefox Browser
- The Postman extension for Chrome

Question 2:

Which java API helps in developing a RESTful web service?

Answer:

There are many frameworks and libraries that a developer can use to create Restful web services in java.

For example , the JAX-RS library is standard way to develop a REST web service.

Also Jersey is another most popular implementations of JAX-RS which offers more than what the specs recommended. There are others like RESTEasy, RESTLet and Apache CFX.

Question 3:

What is the difference between REST and RESTful?

Answer:

REST based services/architecture vs RESTFul services/architecture:

To differentiate or compare these two, we should know what is REST.

REST [Representational State Transfer]: Is basically an architectural style of development having some principles:

- It should be stateless
- It should access all the resources from the server using only URI.
- It does not have inbuilt encryption
- It does not have session
- For performing CRUD operations , it should use HTTP verbs, such as GET, POST, PUT, DELETE, PATCH.
- It should return the result only in JSON format or XML , atom etc. (lightweight data)

REST based services follow some of the above principles and not all.

RESTFUL services means, it follows all the above principles.

Question 4:

Write a sample GET request using Jersey.

Answer:

@GET

@Path("/{empId}")

Employee getEmployee(@PathParam("empId") String empId, @QueryParam("empName") String empName)

Question 5:

Is Spring initializer the only way to create Spring boot projects?

Answer:

No

We can use two approaches:

- The first one is start.spring.io
- The other one is : Setting up a project manually

Setting up a maven project manually:

- In Eclipse, Use File-> New Maven project to create a new project
- Add dependencies
- Add the maven plugins
- Add the Spring boot Application class

Question 6:

Why do we need spring-boot-maven-plugin?

Answer:

Spring-boot-maven-plugin provides a few commands which enable us to package the code as a jar or run the application.

Command in spring-boot-maven-plugin are:

- Spring-boot: run - Runs the spring boot application
- Spring-boot : repackage - Repackages the jar/war to be executable
- Sprint-boot: start and spring-boot: stop - To manage the lifecycle of spring boot application.
- Spring-boot: build-info - Generates build information that can be used by the Actuator.

Question 7:

How can I enable auto reload of Spring application with spring boot?

Answer:

We can enable auto reload of spring application by using spring boot developer tools.

Adding spring boot developer tools to the project is very simple.

Add this dependency to pom.xml:

```
<dependency>

   <groupId> org.springframework.boot</groupId>

   <artifactId>spring-boot-devtools</artifactId>

   <scope>runtime</scope>

</dependency>
```

Restart the application.

Question 8:

What is the use of Embedded servers?

Answer:

Think about what you would need to be able to deploy the application on a virtual machine:

- Step 1: Install Java
- Step 2: Install the web/application server
- Step 3: Deploy the application war

What if we want to simplify it?
How about making the server a pat of the application?

We just need a virtual machine with java installed and we would be able to directly deploy the application on the virtual machine.

This idea is the genesis of Embedded servers.

When we create an application deployable , we would embed the server (for example tomcat or jetty) inside the deployable.

Embedded server is when the deployable unit contains the binaries of the server (example tomcat.jar)

Question 9:

How can we use profiles to configure environment specific configuration with spring boot?

Answer:

Profile is nothing but a key to identify an environment.

In this example, we will use two profiles

- dev
- prod

The default application configuration is present in application.properties. Let's consider an example.

application.properties

- basic.value= true
- basic.message= Dynamic Message
- basic.number= 100

We would want to customize the application.properties for dev profile. We would need to create a file with name application-dev.properties and override the properties that we would want to customize.

application-dev.properties

```
basic.message: Dynamic Message in DEV
```

Similarly you can configure properties for prod profile.

- application-prod.properties
- basic.message: Dynamic Message in Prod

Once you have profile specific configuration, you would need to set the active profile in an environment.

There are multiple ways of doing this:

- Using -Dspring.profiles.active=prod in VM Arguments
- Use spring.profiles.active=prod in application.properties

Question 10:

Which file is required for application configuration with spring boot?

Answer:

The file name is application.properties.

application.properties can reside anywhere in classpath of the application.

Question 11:

How does spring boot reduces configuration?

Answer:

A Web Application using Spring MVC and JPA (Hibernate):

- Pom.xml
- Configure Service/DAO Layer Beans using JavaConfig
- Application.properties file
- Log4j.properties file
- Configure Spring MVC Web Layer Beans
- Register Spring MVC FrontController Servlet DispatcherServlet
- Create JPA Entity
- Create Spring Data JPA repository
- Create a SpringMVC Controller
- Create a Thymeleaf View /WEB-INF/views/index.html

Using Spring Boot:

- Pom.xml
- Application.properties file
- Create JPA Entity
- Create Spring Data JPA repository
- Create Thymeleaf view
- Create SpringBootEntryPoint class

Java Technical interview @ Thales Group: Round-3

Question 1:

What are the differences between save() and persist() methods?

Answer:

- persist() is defined in JPA. save() is defined in hibernate.
- persist() return type is void. save() returns serializable object[generated id].

- persist() doesn't execute insert statement outside transaction boundary. But save() can execute insert statement inside/outside transaction boundary.
- We can persist detached object using save() and it will crate a new row in the table. We cannot persist a detached object using persist() and it will throw PersistentObjectException.

Working of persist() method:
When it is called on transient instance, then that instance's state is changed to persistent. No record will be created in DB until until commit or flush is called on session object or session is closed.

When it is called on persistent object, then nothing happens.

When it is called on detached instance, then it throws PersistentException.

persist() method has void return type.

Note: An instance will go to detached state, when evict() is called on session instance, or session is closed.

Example:

Person p = new Person();

p.setName("abcd");

session.persist(p);

//Here when persist() is called, the person object has transitioned from transient to persistent state.

The object is in the persistent context now, but not yet saved to the db. The generation of INERT statement will occur only upon committing the transaction, flushing or closing the session.

Working of save() method:
The save() method is an original hibernate method that does not conform to the JPA specification.

It's purpose is basically the same as persist(), but it has different implementation details.

The documentation for this method strictly states that it persists the instance, "first assigning a generated identifier". The method is guaranteed to return the serializable value of this identifier.

Person p = new Person();

p.setName("abcd");

long id = (long) session.save(p);

The effect of saving an already persisted instance is the same as with persist(). Difference comes when we try to save a detached instance.

Person p = new Person();

p.setName("abcd");

long id = (long) session.save(p);

session.evict(p);

long id2 = (long)session.save(p);

The id2 variable will differ from id1. The call of save() on a detached instance creates a new persistence instance and assigns it a new identifier, which results in a duplicate record in a database upon committng or flushing.

Working of update() method:

As with *persist* and *save*, the *update* method is an “original” Hibernate method that was present long before the *merge* method was added. Its semantics differs in several key points:

- it acts upon passed object (its return type is *void*); the *update* method transitions the passed object from *detached* to *persistent* state;
- this method throws an exception if you pass it a *transient* entity.

```
 import org.hibernate.*;

import org.hibernate.cfg.*;

public class ClientLogicProgram{

public static void main(String... args)

{

     Configuration cfg = newConfiguration();

cfg.configure("hibernate.cfg.xml");

 SessionFactoryfactory = cfg.buildSessionFactory();

      Session session1 = factory.openSession();

      Product p=null;   //Transient state..

      Object o=session1.get(Product.class, new Integer(1001));

      p=(Product)o;   //now p is in Persistent state..

      session1.close();

 p.setPrice(36000);            // p is in Detached state

      Session session2=factory.openSession();

      Transaction tx=session2.beginTransaction();

        session2.update(p);   // now p reached to Persistent state

tx.commit();
```

```
    session2.close();

factory.close();

}

}
```

Question 2:

What are spring transaction isolation levels?

Answer:

Isolation level defines how the changes made to some data repository by one transaction affect other simultaneous concurrent transactions. And also how and when that changed data becomes available to other transactions.

When we define a transaction using the spring framework , we are also able to configure in which isolation level that same transaction will be executed.

Usage example:

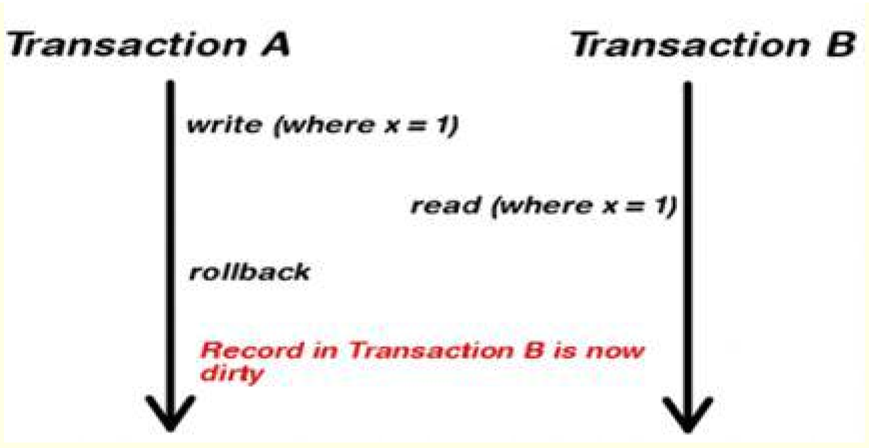
Using the @Transactional annotation, we can define the isolation level of a spring managed bean transactional method. This means that the transaction in which this method is executed will run with that isolation level.

Isolation level in a transactional method:

```
@Autowired

private TestDAO testDAO;

@Transactional (isolation = Isolation.READ_COMMITTED)

public void someTransactionalMethod(User user){

   // interact with testDAO

}
```

READ_UNCOMMITTED:

READ_UNCOMMITTED isolation level states that a transaction may read data that is still uncommitted by other transactions. This constraint is very relaxed in what matters to transactional concurrency , but it may led to some issues like dirty reads.

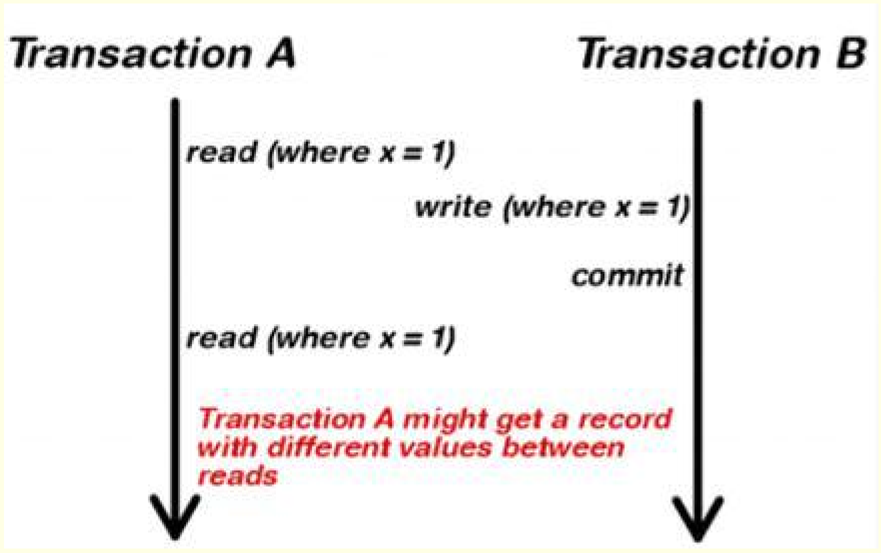


Note: READ_UNCOMMITTED is also vulnerable to non-repeatable reads and phantom reads.

READ_COMMITTED:

This level states that a transaction cannot read data that is not yet committed by other transactions.

This means that dirty read is no longer an issue , but even this way other issues may occur.

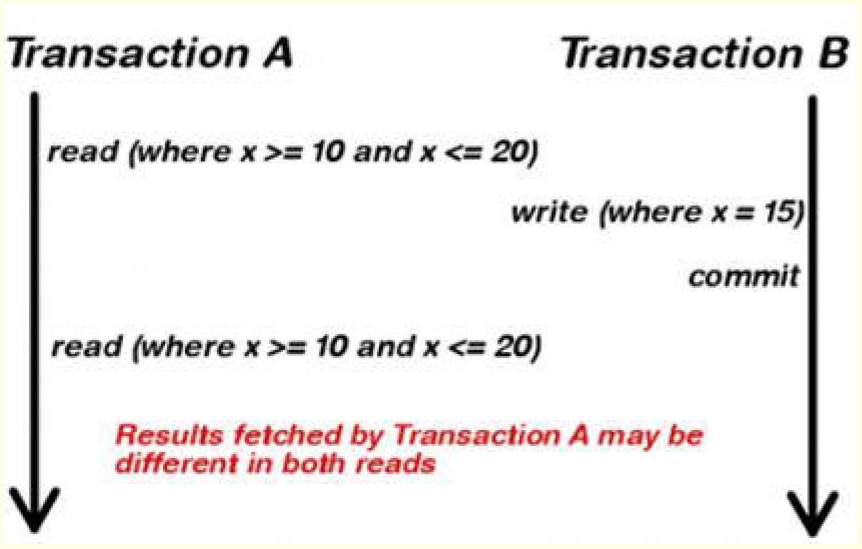


In this example **Transaction A** reads some record. Then **Transaction B** writes that same record and commits. Later **Transaction A** reads that same record again and may get different values because **Transaction B** made changes to that record and committed. This is a **non-repeatable read**.

Note: READ_COMMITTED is also vulnerable to **phantom reads**.

REPEATABLE_READ

REPEATABLE_READ isolation level states that if a transaction reads one record from the database multiple times the result of all those reading operations must always be the same. This eliminates both the **dirty read** and the **non-repeatable read** issues, but even this way other issues may occur.


Transaction A
Transaction B
read (where x >= 10 and x <= 20)
write (where x = 15)
commit
read (where x >= 10 and x <= 20)
Results fetched by Transaction A may be different in both reads

In this example **Transaction A** reads a **range** of records. Meanwhile **Transaction B** inserts a new record in the same range that **Transaction A** initially fetched and commits. Later **Transaction A** reads the same range again and will also get the record that **Transaction B** just inserted. This is a **phantom read**: a transaction fetched a range of records multiple times from the database and obtained different result sets (containing phantom records).

SERIALIZABLE:

This is the most restrictive of all isolation levels. Transactions are executed with locking at all levels (read, range and write locking) so they appear as if they were executed in a serialized way.

This leads to a scenario where none of the issues mentioned above may occur. But in the other way, we do not allow transaction concurrency and consequently introduce a performance penalty.

Interview questions on Singleton, Serialization and enum

Question 1:

Which classes are candidates of Singleton?

Answer:

Any class which we want to be available to the whole application and only one instance should be available is a candidate for singleton.

e.g.
Popup utility class in GUI application. If we want to show popup with message , we can have only one instance of this class and we just call show() method with message.

Question 2:

What is lazy and early loading in Singleton?

Answer:

When we create singleton instance using static final at the time of class loading, it is called early loading. And when singleton instance is created when the client requests , it is called lazy loading.

Question 3:

Give any Singleton example from Java development kit?

Answer:

- java.lang.Runtime
- java.awt.Desktop

Question 4:

How to customize java serialization of an object?

Answer:

We can use readObject(ObjectInputStream ois) and writeObject(ObjectOutputStream oos) methods of java.io.ObjectInputStream and java.io.ObjectOutputStream classes.

readObject(ObjectInputStream) throws ClassNotFoundException and IOException.

writeObject(ObjectOutputStream) throws IOException.

We can use writeUTF(), writeInt(), writeLong() and readUTF(), readInt() and readLong() methods defined in ObjectOutputStream and ObjectInputSteam classes.

In this, next question becomes:

Why readObject(ObjectInputStream ois) throws ClassNotFoundException?

Answer:

Because readObject(ObjectInputStream) is used on deserialization and there may be the case that class is not found. e.g. On client side, we have a class Message in client package and on server side , we have same class in server package, then while deserializing, class will not be found because package client is not found.

But, it is not the case with writeObject(ObjectOutputStream). That is why this method does not throw ClassNotFoundException.

Question 5:

We use serialVersionUID in serialized classes. Why this ID is made static?

Answer:

During deserialization, it is checked that whether both sender and receiver of serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of senders class, then InvalidClassException is thrown. That check is made before creating the object and by comparing serialVersionUID. As no object is present at that time, that is why it is static.

Code Scenario:

```
public class Account implements  Serializable{

        String strUserName = "Anish";

        transient String pwd = "abcd";

        transient String onlyTest = "test";

        private void writeObject(ObjectOutputStream oos) throws Exception{

                oos.defaultWriteObject();

                 String epwd = "12345"+  pwd;

                 String etest = "54321" + onlyTest;

                 oos.writeUTF(epwd);

                 oos.writeUTF(etest);

        }

         private  void  readObject(ObjectInputStream ois) throws Exception{

                 ois.defaultReadObject();

                 Account acc = null;

                 String epwd = (String) ois.readUTF();
                 String etest = (String) ois.readUTF();

                 pwd = epwd;

                 onlyTest = etest;

        }

}

public static void main(String[] args){



                 Account account = new Account();

                 try {

                         FileOutputStreamfos
= new FileOutputStream("C:\\Users\\Lenovo\\TestFiles\\a.txt");

                         ObjectOutputStream oos = new ObjectOutputStream(fos);

                         oos.writeObject(account);

                        FileInputStream fis = new FileInputStream("C:\\Users\\Lenovo\\TestFiles\\a.txt");

                         ObjectInputStream ois = new ObjectInputStream(fis);

                         Account account1 = (Account)ois.readObject();

                         oos.close();

                         fos.close();

                         ois.close();

                         fis.close();

                 }

                 catch(Exception e) {

                 }

}
```

Above example code is for customized serialization and deserialization.

Customized serialization is required when we have to send some secure information over the network in encrypted form. So,in this example we are sending transient pwd in serialized form. And then deserializing it.

Points to note down:

1. We have to write readObject() and writeObject() methods.
2. Whatever , we need to make secure , we need to write it in ObjectOutputStream specifically. Other variables will be written by a call to defaultWriteObject(). Same thing applies to when reading object using readObject() method. If we don't call defaultWriteObject() , then other variables will not be written.
3. Order of writing and reading the variables must be same.

Question 6:

Is it mandatory to declare enum constants with UPPERCASE letters?

Answer:

No,but using uppercase is a good coding practice.

Question 7:

Can enum types have fields, method and constructors?

Answer:

Yes.

Example:

```
enum EnumClass{

  A, B, C;   // Enum contants

  int i;   // Enum can have fields

  private EnumClass(){

  // Enums can have constructors

  }

  void method(){

    //Enum can have methods

  }

}
```

Question 8:

What is the access type of enum constructor?

Answer:

Enum constructors are private by default. Only private constructors are allowed in enum types. That's why we cannot instantiate enum types using new operator.

Question 9:

How enum constants work?

Answer:

Enum constants are created only once for the whole execution. All enum constants are created when you refer any enum constant first time in your code. While creating each enum constant, corresponding constructor is called.

```
enum Enums

{

   A, B(10), C("ccc", 20);

   //No-arg private constructor

   private Enums()

   {

      System.out.println(1);

   }

   //Private constructor taking one argument


   private Enums(inti)

   {

      System.out.println(2);

   }
```

```
    //Private constructor taking two arguments

    private Enums(String s, intj)

    {

        System.out.println(3);

    }

}


public class MainClass

{

    public static void main(String[] args)

    {

        Enums en = Enums.B;   //All enum constants are created while executing this statement.

        //While creating each enum constant, corresponding constructor is called

        Enums en2 = Enums.C;   //No enum constant is created here.

        Enums en3 = Enums.A;   //No enum constant is created here.

    }

}
```

Question 10:

Can we declare enum constants anywhere?

Answer:

No. These must be declared first ahead of fields, constructors and methods(if any).

```
enum Enums{

    int i;

    A, B, C;   // Compile time error, enum constants must be declared first.

}
```

Java Technical Architect interview @ Wipro

Question 1:

How to join three threads in java? Suppose we have three threads T1, T2, T3.

We want that T2 should run after T1 and T3 should run after T2.

Answer:

```
public class ThreadDemo{

   private static class ParallelTask implements Runnable{

       private Thread predecessor;

       @Override

       public void run(){

           if(predecessor != null){

               try{

                   predecessor.join();

               }

               catch(InterruptedException ie){

                   ie.printStackTrace();

               }

           }

       }

       public void setPredecessor(Thread t){

           this.predecessor = t;

       }

   }

   public static void  main(String[] args){
       ParallelTask task1 = new ParallelTask();

       ParallelTask task2 = new ParallelTask();

       ParallelTask task3 = new ParallelTask();



       final Thread t1 = new Thread(task1, "Thread-1");

       final Thread t2 = new Thread(task2, "Thread-2");

       final Thread t3 = new Thread(task3, "Thread-3");

       task2.setPredecessor(t1);

       task3.setPredecessor(t2);



       //Now start all the threads.

       t1.start();

       t2.start();

       t3.start();

   }

   }
```

Question 2:

When a class is initialized?

Answer:

There are multiple ways in which a class is initialized.

- An instance of class is created using either new operator or using reflection using Class.forName() method.
- A static method of class in invoked.
- A static member of class is accessed , which is not a constant variable.

How class is initialized in java?

1). Classes are initialized from top to bottom so field declared on top initialized before field declared in bottom.

2). Super class is initialized before subclass or derived class in java.

3). If class initialization is triggered due to access of static field , only class which has declared static field is initialized and it doesn't trigger initialization of super class or sub class even if static field is referenced by type of subclass , sub interface or by implementation class of interface.

4). interface initialization in java doesn't cause super interfaces to be initialized.

5). static fields are initialized during static initialization of class while non-static fields are initialized when instance of class is created. It means static fields are initialized before non static fields in java.

6). non static fields are initialized by constructors in java. Subclass constructor implicitly call super class constructor before doing any initialization, which guarantees that non-static or instance variables of super class is initialized before subclass.

```
public class Fruit{

    public static String field = "field";

    static{

        System.out.println("static block of fruit");

    }

    {

        System.out.println("Non-static block of fruit");

    }

}

public class Apple extends Fruit{

    static{

        System.out.println("static block of apple");

    }

    {

        System.out.println("Non-static block of apple");

        GreenApple a = null; // Subclass of Apple

        System.out.println("comparing = "+ (this == a));
    }



}
```

main() method:

Apple apple = new Apple();

O/P:

static block of Fruit

static block of Apple

Non-static block of Fruit

Non-static block of Apple

Now, if I have access static field of Fruit class using Apple class, then

O/P: static block of Fruit

Question 3:

What are the ways to make an object eligible for Garbage collection?

Answer:

- Nullifying the reference variable.
- Re-assigning the reference variable.
- Object created inside method

Question 4:

What are the ways for JVM to run garbage collector?

Answer:

We have to ways:

- using System.gc();
- Using Runtime.getRuntime().gc()

Both ways are equivalent . And there is no guarantee that any of the above two methods will definitely run garbage collector.

Question 5:

Which component calls finalize() method?

Answer:

The finalize() method gets called by garbage collector not JVM. Although garbage collector is one of the module of JVM.

Question 6:

What is ClassLoader in java?

Answer:

ClassLoader in java is a class which is used to load class files in java.

Java code is compiled into class file by javac compiler and JVM executes java program, by executing byte codes written in class file.

ClassLoader is responsible for loading class files from file system, network or any other source. There are three default class loader used in java : Bootstrap, Extension and System or Application class loader.

Java class loaders are used to load classes at runtime.

Classloader in java works on three principles:

- Delegation
- Visibility
- Uniqueness

Delegation principle forward request of class loading to parent class loader and only loads the class , if parent is not able to find or load class.

Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child.

Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent.

Note: It is possible to write our own ClassLoader and violate the Delegation and Uniqueness principle. But, it is not beneficial.

Question 7:

How to load class explicitly in java?

Answer:

Using Class.forName(classname);

Using Class.forName(classname, boolean , classloader);

Class is loaded by calling loadClass() method of java.lang.ClassLoader class which calls findClass() method to locate bytecodes for corresponding class. If findClass() doesn’t find the class , then it throws java.lang.ClassNotFoundException and if it finds, it calls defineClass() to convert bytecodes into a .class instance which is returned to the caller.

Java Technical Interview @ Wipro: Round-2

Question 1:

What is N+1 problem in hibernate?

Answer:

N+1 problem is a performance issue in object relational mapping that fires multiple select queries (N+1 to be exact , where N = number of records in table) in database for a single select query at application layer.

Hibernate provides multiple ways to catch and prevent this problem.

What is N+1 problem?

To understand N+1 problem, lets consider a scenario. Let's say we have a collection of user objects mapped to tab_users table in database and each user has collection of roles mapped to tab_roles table using a joining table tab_user_roles. At the ORM level , a user has many to many relationship with role.

```
@Entity

@Table(name="tab_users")

public class User{

   @Id

   @GeneratedValue(strategy=GenerationType.AUTO)

   private long id;

   private String name;

   @ManyToMany(fetch=FetchType.LAZY)

   private Set<Role> roles;

}



@Entity

@Table(name="tab_roles")
```

```
public class Role{

   @Id

   @GeneratedValue(strategy = GenerationType.AUTO)

   private long id;

   private String name;

}
```

Now let's say we want to fetch all users from users table and print roles for each one. Very naive Object Relational implementation will be:

First get All users:

```
select * from tab_users;
```

Then get roles for each user:

```
select * from tab_user_roles where user_id=<userid>;
```

So, we need one select for user and N additional selects for fetching roles for each user, where N is total number of users. This is a classic N+1 problem in ORM.

How to identify it?

Hibernate provides tracing option that enables SQL logging in the console/logs. Using logs, we can easily see if hibernate is issuing N+1 queries for a given call.

Enabling SQL logging in application.yml:

```
spring:

   jpa:

      show-sql: true

      database-platform: org.hibernate.dialect.H2Dialect

      hibernate:

         ddl-auto:create

         use-new-id-generator-mappings: true

      properties:
```

```
      hibernate:

          type: trace
```

Enables SQL logging in trace.

We have to enable this too in order to show sql queries in logs.

N+1 Resolution:

Hibernate and spring data provide mechanism to solve the N+1 ORM issue.

At SQL level, what ORM needs to achieve to avoid N+1 is to fire a query that joins the two tables and get the combined results in single query.

Spring data JPA approach:

If we are using spring data JPA, then we have two options to achieve this - using EntityGraph or using select query with fetch join.

```
public interface UserRepository extends CrudRepository<User, long>{

    List<User> findAllByRolesIn(List<Role> roles);

    @Query("select p from User p Left Join Fetch p.roles")

     List<User> findWithoutNPlusOne();

    @EntityGraph(attributePaths= {"roles"})

    List<User> findAll();
```

N+1 queries are issued at database level.

Using left join fetch , we resolve the N+1 problem.

Using attributePaths, Spring data JPA avoids N+1 problem.

Hibernate approach:

If it is pure hibernate, then the following solution will work.

HQL Query:

"from User u join fetch u.roles roles roles"

Hibernate Criteria API:

```
Criteria criteria = session.createCriteria(User.class);

criteria.setFetchMode("roles", FetchMode.EAGER);
```

Under the hood, all these approaches work similar and they issue a similar database query with left join fetch.

Question 2:

Write code to implement ThreadPool in java.

Answer:

```
 import java.util.concurrent.LinkedBlockingQueue;

public class ThreadPool{

    private final int nThreads;

    private final PoolWorker[] threads;

    private final LinkedBlockingQueue queue;

    public ThreadPool(int nThreads){

        this.nThreads = nThreads;

        queue = new LinkedBlockingQueue();

        threads = new PoolWorker[nThreads];

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

            threads[i] = new PoolWorker();

            threads[i].start();

        }

    }

    public void execute(Runnable task){

        synchronized(queue){

            queue.add(task);

            queue.notify();

        }
    }

    private class PoolWorker extends Thread{

        public void run(){

            Runnable task;

            while(true){

                synchronized(queue){

                    while(queue.isEmpty()){

                        try{

                            queue.wait();

                        }

                        catch(InterruptedException ie){


                        }

                    }

                    task = queue.poll();

                }

                try{

                    task.run();

                }

                catch(RuntimeException e){

                }


            }

        }

    }
```

}

Question 3:

What is CountDownLatch?

Answer:

CountDownLatch is used to make sure that a task waits for other threads to finish before it starts.

e.g.

Consider a server where the main task can only start when all the required services have started.

Working of CountDownLatch:

When we create an object of CountDownLatch, we specify number of threads it should wait for. All such threads are required to countdown by calling CountDownLatch.countDown(), once they are completed or ready to do the job. As soon as count reaches 0, the waiting task starts running.

Question 4:

What is CyclicBarrier?

Answer:

java.util.concurrent.CyclicBarrier

CyclicBarrier is used to make threads wait for each other. It is used when different threads process a part of computation and when all threads have completed the execution, the result needs to be combined in the parent thread.

In other words, a CyclicBarrier is used when multiple threads carry out different subtasks and the output of these subtasks need to be combined to form the final output.

After completing it's execution, threads call await() method and wait for other threads to reach the barrier. Once all the threads have reached, the barriers then give the way for threads to proceed.

Technical Architect interview @ DoctorAnywhere

Question 1:

How does the communication happen in microservice architecture?

Answer:

In microservice architecture, client and services can communicate through many different types of communication, each one targeting a different scenario and goals. Initially, those types of communications can be classified in two axes.

The first axis defines if the protocol is synchronous or asynchronous.:

Synchronous Protocol: HTTP is a synchronous protocol. The client send a request and waits for a response from the service. That's independent of the client code execution that could be synchronous(thread is blocked) or asynchronous(thread is not blocked and the response will reach to a callback eventually).

Asynchronous protocol: Other protocols like AMQP use asynchronous messages. The client code or message sender usually doesn't wait for a response. It just sends the message as when sending a message to a RabbitMQ queue or any other message broker.The second axis defines if the communication has a single receiver or multiple receivers:

Single receiver: Each request must be processed by exactly one receiver or service. An example of this communication is the command pattern.

Multiple receivers: Each request can be processed by zero to multiple receivers. This type of communication must be asynchronous. An example is the publish/subscribe mechanism used in patterns like event-driven architecture.

This is based on an event-bus interface or message broker when propagating data updates between multiple microservices through events. It is usually implemented through a service bus like Azure Service bus by using topics and subscriptions.

A microservice-based application will often use a combination of these communication styles. The most common type is single-receiver communication with a synchronous protocol like HTTP/HTTPS.

Microservices also typically use messaging protocols for asynchronous communication between microservices.

Note:

Ideally, we should try to minimize the communication between the internal microservices. The fewer communications between microservices, the better. But sometimes, we need to implement communication between the microservices and critical rule here is that the communication between the microservices should be asynchronous.

If we implement synchronous communication between microservices, we have an architecture that will not be resilient when some microservices fail.

If our microservice needs to raise an additional action in another microservice, if possible do not perform that action synchronously. Instead do it asynchronously (using asynchronous messaging or integration events).

And finally if our initial microservice needs data that is originally owned by other microservices, do not rely on making synchronous requests for that data. Instead , replicate or propagate that data(only the attributes we need) into the initial service's database by using eventual consistency.

Note: Duplicating some data across several microservices is not an incorrect design.

When a client uses request/response communication, it assumes that the response will arrive in a short time, typically less than a second or few seconds at most.

For delayed responses, you need to implement asynchronous communication based on messaging patterns and messaging technologies.

Question 2:

How to implement asynchronous message-based communication?

Answer:

Asynchronous messaging and event-driven communication are critical when propagating changes across multiple microservices and their related domain models.

A solution is eventual consistency and event-driven communication based on asynchronous messaging.

When using messaging, processes communicate by exchanging messages asynchronously. A client makes a command or a request to a service by sending it a message.If the service needs to reply, it sends a different message back to the client. Since it is a message based communication, the client assumes that the reply will not be received immediately and that there might be no response at all.

A message is composed by a header (metadata such as identification or security information) and a body. Messages are usually sent through asynchronous protocols like AMQP.

A rule we should follow is that to use only asynchronous messaging between the internal services and to use synchronous communication (such as HTTP) only from the client apps to the front-end services(API gateways plus the first level of microservices).

There are two kinds of asynchronous messaging communication: single receiver message-based communication and multiple receivers message-based communication.

Single-receiver message-based communication is especially well suited for sending asynchronous commands from one microservice to another.

Once we start sending message-based communication (either with commands or events), we should avoid mixing message-based communication with synchronous HTTP communication.

In multiple receiver message-based communication, we use a publish-subscribe communication and for that we can use an event bus interface to publish events to any subscriber.

Asynchronous event-driven communication:

When using asynchronous event-driven communication, a microservice publishes an integration event when something happens within it's domain and another microservice needs to be aware of it, like a price change in a product catalog microservice.

Additional microservices subscribe to the events so they can receive them asynchronously. When that happens, the receivers might update their own domain entities, which can cause more integration events to be published. This publish/subscribe system is usually performed by using an implementation of an event bus.

Question 3:

You have two sticks and matchbox. Each stick takes exactly an hour to burn from one end to the other.

The sticks are not identical and do not burn at a constant rate. As a result, two equal lengths of the stick would not necessarily burn in the same amount of time. ow would you measure exactly 45 minutes by burning these sticks.

Answer:

0 minutes - Light stick 1 on both sides and stick 2 on one side.

30 minutes - Stick 1 will be burnt out. Light the other end of stick 2.

45 minutes - stick 2 will be burnt out.

Core Java Interview Questions: Part-1

Question 1:

What is polymorphism and what are it's types?

Answer:

It is one of the OOPS feature that allows us to perform a single action in different ways.

Polymorphism is the capability of a method to do different things based on the object that it is acting upon.

Types of polymorphism are:

- Runtime polymorphism or Dynamic polymorphism
- Compile time polymorphism or Static polymorphism

Question 2:

What are early binding and late binding?

Answer:

Static binding or early binding:

The binding which can be resolved at compile time by the compiler is known as static or early binding.

The binding of static , private and final methods is compile time.Because these methods cannot be overridden and the type of the class is determined at compile time.

Dynamic binding or late binding:

When compiler is not able to resolve call/binding at compile time, such binding is known as dynamic binding or late binding.

Question 3:

What is the use of super keyword?

Answer:

The super keyword refers to the objects of immediate parent class.

Use of super keyword:

- To access the data members of parent class when both parent and child class have members with same name.
- To explicitly call the no-arg and parameterized constructor of parent class.
- To access the methods of parent class when child class has overridden that method.

Question 4:

What are the static and non-static methods in Thread class?

Answer:

Static methods in thread class are:

- yield()
- activeCount()
- currentThread()
- sleep()

Non-static methods in thread class are:

- start()
- run()
- join()
- getName()
- setName()
- getPriority()
- setPriority()
- interrupt()
- getState()

Question 5:

What is thread scheduler and Time slicing?

Answer:

Thread scheduler is OS service that allocates CPU time to the available running threads.

Time slicing is the process to divide the available CPU time to the available running threads.

Question 6:

How can we make sure that main() is the last thread to finish in java program?

Answer:

We can call join()[non-static method] method on threads to make sure all threads created by java program are dead before finishing the main() function.

Overloads of join allow a programmer to specify a waiting period. However, as with sleep(), join() is dependent on OS for timing, so we should not assume that join() will wait exactly as long as you specify.

Like sleep(), join() responds to an interrupt by exiting with an InterruptedException.

join() is a final method in java and we cannot override it.

Question 7:

What Thread class's sleep() and yield() methods are static?

Answer:

These methods work on currently executing thread. These are made static to avoid confusion to the programmer who might think , they can invoke these methods on some non-running threads.

Question 8:

How do we achieve thread safety?

Answer:

There are multiple mechanisms to achieve thread safety:

- Synchronization
- Atomic concurrent classes
- Concurrent locks
- volatile keyword
- Immutable classes
- Thread safe classes

Question 9:

What is ThreadLocal?

Answer:

Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it's variables, so if the variable is not thread safe, we can use synchronization, but if we want to avoid synchronization, we can use ThreadLocal variables.

Every thread has it's own ThreadLocal variable and they can use it's get() and set() methods to get the default value or change it's value local to thread. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread.

Question 10:

What are the changes in java 10?

Answer:

There are multiple new changes that have been added in java 10.

- var keyword
- Unmodifiable collection enhancements
- G1GC performance improvements
- New JIT compiler [created in pure java]
- Alternate memory devices for allocating heap
- Application class data sharing

Question 11:

What is pagination?

Answer:

Pagination in web application is a mechanism to separate a big result set into smaller chunks. Providing a fluent pagination navigator could increase both the value of the website for search engines and enhance user experience through minimizing the response time.

The famous example of pagination is google's search result page. Normally,the result page is separated into several pages.

The spring framework provides an out of the box feature for pagination that needs the number of pages and number of elements per page. This is very useful when we would implement static pagination which can offer a user to choose between different pages.

Core Java Interview Questions: Part-2

Question 1:

What is the point of having a private constructor?

Answer:

We need a private constructor in two cases:

- When we nee singleton pattern
- When we have only static methods, then we do not need any object.

Question 2:

What is difference between ClassNotFoundException and NoClassDefFoundError.

Answer:

ClassNotFoundException occurs when:

- The forName() method of class tries to load a class.
- The findSystemClass() method in ClassLoader tries to load a class.
- The loadClass() method in ClassLoader tries to load a class.

But no class is found.

NoClassDefFoundError occurs when:

- The class was compiled successfully but couldn't be integrated in JAR file, so at runtime , class is not found.
- When ClassLoader tries to load a class , it ran into an error reading the class definition. This typically happens when the class in question has static blocks and members which use a class that is not found by ClassLoader. So, to find the culprit, look at the code of your class in question and look for code using static blocks or static members.
- When code is compiled on one machine and run on another, so in this case, we may forget to set the classpath properly and this error occurs.
- Any start-up script is overriding classpath environment variable.
- This error comes under Throwable -> Error -> LinkageError -> NoClassDefFoundError

Question 3:

What is a copy constructor?

Answer:

A copy constructor is a constructor which takes a single parameter and which is of type class.

Use case:

It is used to create another object which is a copy of the object that it takes as a parameter.

But, the newly created copy is actually independent of the original object.

Question 4:

What is a WeakHashMap?

Answer:

It is defined in java.util.WeakHashMap.

It is a java collection that contains weak references. This class stores keys [not values] as weak references. If the key is garbage collected, then the entry from the WeakHashMap is also removed.

Question 5:

Is WeakHashMap synchronized?

Answer:

No, it is not synchronized. But we can have synchronized version using Collections.synchronizedMap() method.

Question 6:

Does WeakHashMap allow null keys and values?

Answer:

Yes, it allows null keys and values.

Question 7:

How does key in WeakHashMap work?

Answer:

Key stored in WeakHashMap works as a referent of a weak reference. When the weak reference are garbage collected, then the key and it's value are removed from WeakHashMap.

Question 8:

How WeakHashMap is implemented?

Answer:

The value objects in WeakHashMap are held by ordinary strong references. Tus care should be taken to ensure that value objects do not strongly refer to their own keys, either directly or indirectly., since that will prevent the keys from being discarded. Note that a value object may refer indirectly to it's key via the WeakHashMap itself, that is, a value object may strongly refer to some other key object whose associated value object, in turn strongly refers to the key of the first value object.

One way to deal with is to wrap values themselves within weak references before inserting and then unwrapping upon each get.

Question 9:

What are the differences between Heap and Stack?

Answer:

There are multiple differences between heap and stack. These are as described below:

- Heap is used to store objects [Objects created anywhere in the class, inside method or in any block of code]. Only exception is the string literal which are stored mainly in permgen area until java 7 in which string pool was moved to heap. Stack space is used to store local variables, method frames and call stack.
- Heap space in java is much bigger than stack space.
- Heap space is shared by threads. But each thread has it's own stack space. And local variables created in thread are not visible to other threads.
- We can resize heap space using -Xms and -Xmx to specify starting and maximum heap memory in java. Similarly, we can use -Xss to specify stack size of individual threads in JVM.
- Order: In heap, objects can be create and stored in any order. But stack memory is structured as LIFO, where method calls are stored as Last In First out order. That is why we can use recursion in java.
- When a heap gets filled, we get java.lang.OutOfMemoryError: java Heap space. When a stack memory gets filled up, then java.lang.StackOverflowError is thrown.

Question 10:

How we can get total memory, free memory and maximum memory in java?

Answer:

Using Runtime class methods , we can get total memory, free memory and maximum memory in java.

- Runtime.getRuntime().freeMemory()
- Runtime.getRuntime().maxMemory()
- Runtime.getRuntime().totalMemory()

Question 11:

What are loading, linking and initialization mechanism in class loader in java?

Answer:

Loading: This component handles the loading of the .class files from the hardware system into the JVM memory and stores the binary data (such as fully classified class name, immediate parent class-name, information about the methods, variables and constructors etc.) in the method areas.

For every loaded .class file, JVM immediately creates an object on the heap memory of type java.lang.class. Do remember, even though the developers call a class multiple times, only one class object is always created. There are three main types of classloaders:

- Bootstrap ClassLoader
- Extension ClassLoader
- Application or System ClassLoader

Linking: This process performs the linking of a class or interface. As this component involves the allocation of new data structure, it may throw the OutOfMemoryError and performs three important activities:

- Verification
- Preparation
- Resolution

Initialization: This component performs the final phase of the class loading where all the static variables are assigned the original values and the static blocks are executed from the parent to the child class.

This process requires careful synchronization as JVM is multithreaded and some threads may try to initialize the same class or interface at the same time.

Question 12:

Where to use ClassLoader in java?

Answer:

Popular example of classloader is Applet ClassLoader which is used to load class by Applet. Since applets are loaded mostly from internet rather than local file system, by using separate ClassLoader , we can load same class from multiple sources and they will be treated as different class in JVM.

Question 13:

What is the maximum size of java array?

Answer:

The maximum size of java array is VM dependent.

Though array size will be Integer.MAX_VALUE - num.

Where num can e anything from 2 to 5.

e.g.:

If maximum size of java array for a VM is Integer.MAX_VALUE - 5, then creating a java array with size Integer.MAX_VALUE - 4 will throw java.lang.OutOfMemoryError.

Interview questions on spring

Question 1:

List all the spring core , stereotype annotations, spring boot annotations and spring MVC and WEB annotations.

Answer:

- @Qualifier
- @Autowired
- @Required
- @ComponentScan
- @Configuration
- @Bean
- @Lazy
- @Value

Spring Framework Stereotype annotations:

- @Component
- @Controller
- @Service
- @Repository

Spring Boot Annotations:

- @EnableAutoConfiguration
- @SpringBootApplication

Spring MVC and REST[WEB] Annotations:

- @Controller
- @RequestMapping
- @CookieValue
- @CrossOrigin
- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping
- @PatchMapping
- @ExceptionHandler
- @InitBinder
- @Mappings and @Mapping
- @MatrixVariable
- @PathVariable
- @RequestAttribute
- @RequestBody
- @RequestHeader
- @RequestParam
- @RequestPart
- @ResponseBody
- @ResponseStatus
- @ControllerAdvice
- @RestController
- @RestControllerAdvice
- @SessionAttribute
- @SessionAttributes

Question 2:

What is the use of @Required annotation?

Answer:

It is required when we want to ensure that all the required properties have been set.

In Spring, there is **dependency-check attribute** and using this we can only check whether the properties has been set or not. But we can't check if their value is set to null or non-null.

Using @Required annotation we can check if values are set to non-null.

It is used on setter methods.

While using @Required, we should register **RequiredAnnotationBeanPostProcessor** class that checks if all the bean properties with the @Required annotation has been set.

applicationContext.xml ->

```
<bean id="manager"  class="com.howtodoinjava.demo.factory.EmployeeFactoryBean">

   <!-- <property name="designation" value="Manager" /> -->

</bean>

 <bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />
```

Note: If any property with @Required annotation have not been set, a **BeanInitializationException** will be thrown by this bean post processor.

In this way, you can use @Required annotation and RequiredAnnotationBeanPostProcessor class to verify that on context initialization, all the required bean properties have been set properly.

Question 3:

What is the use of @Value annotation?

Answer:

This annotation is used to assign default value to variables and method arguments. We can read Spring environment variables and System variables as well.

@Value only takes String values.

```
@Value("true")

private boolean defaultBoolean;

@Value("10")

private int defaultInt;
```

```
@Value("Test")

public void printValues(String s, String v) // Both s and v values will be test.



@Value("Test")

public void printValues(String s, @Value("Data") String v) // Here s will be Test and v will be Data.
```

Question 4:

What are available spring framework modules?

Answer:

- Spring Core module
- Spring MVC module
- Spring Context [J2EE]
- Spring DAO module
- Spring ORM module
- Spring AOP

Question 5:

What are the different features of spring framework?

Answer:

There are multiple features in spring framework.

- **Lightweight:** Spring is lightweight when it comes to size and transparency.
- **Inversion of control (IOC):** The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion Of Control.
- **Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.
- **Container:** Spring Framework creates and manages the life cycle and configuration of the application objects.
- **MVC Framework:** Spring Framework's MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.

- **Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework. Spring's transaction support can be also used in container less environments.
- **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.

Question 6:

What are the different components of a spring application?

Answer:

A Spring application, generally consists of following components:

- Interface: It defines the functions.
- Bean class: It contains properties, its setter and getter methods, functions etc.
- Spring Aspect Oriented Programming (AOP): Provides the functionality of cross-cutting concerns.
- Bean Configuration File: Contains the information of classes and how to configure them.
- User program: It uses the function.

Question 7:

What do we mean by dependency injection?

Answer:

In Dependency Injection, we do not have to create your objects but have to describe how they should be created.

We don't connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

Question 8:

What are the benefits of using IOC in spring?

Answer:

Some of the benefits of IOC are:

- It will minimize the amount of code in your application.
- It will make your application easy to test because it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.
- It promotes loose coupling with minimal effort and least intrusive mechanism.
- It supports eager instantiation and lazy loading of the services.

How Kafka Consumer consumes messages from Kafka Topic

Kafka Consumers: Reading data from Kafka

Applications that need to read data from Kafka use a KafkaConsumer to subscribe to Kafka topics and receive messages from these topics.

Kafka Consumer Concepts

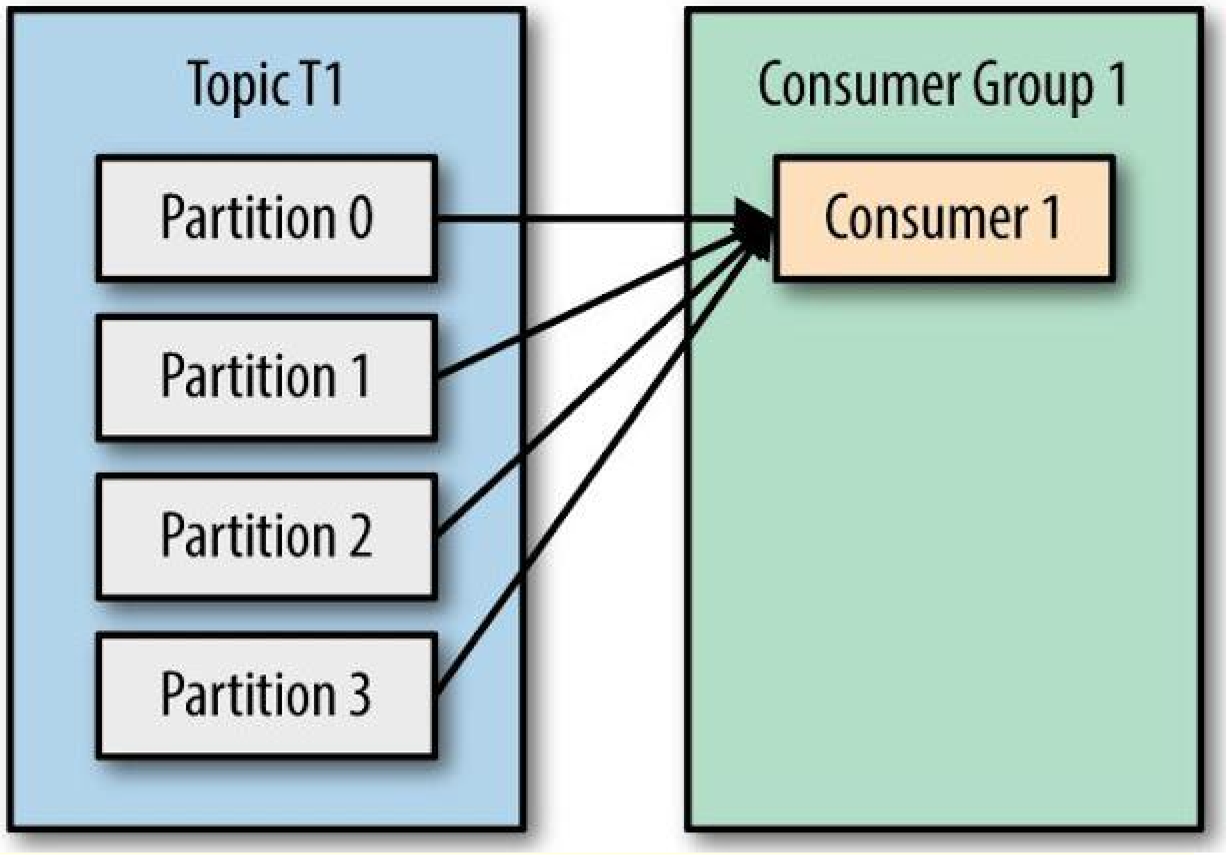
In order to understand how to read data from Kafka, you first need to understand its consumers and consumer groups. The following sections cover those concepts.

Consumers and Consumer Groups

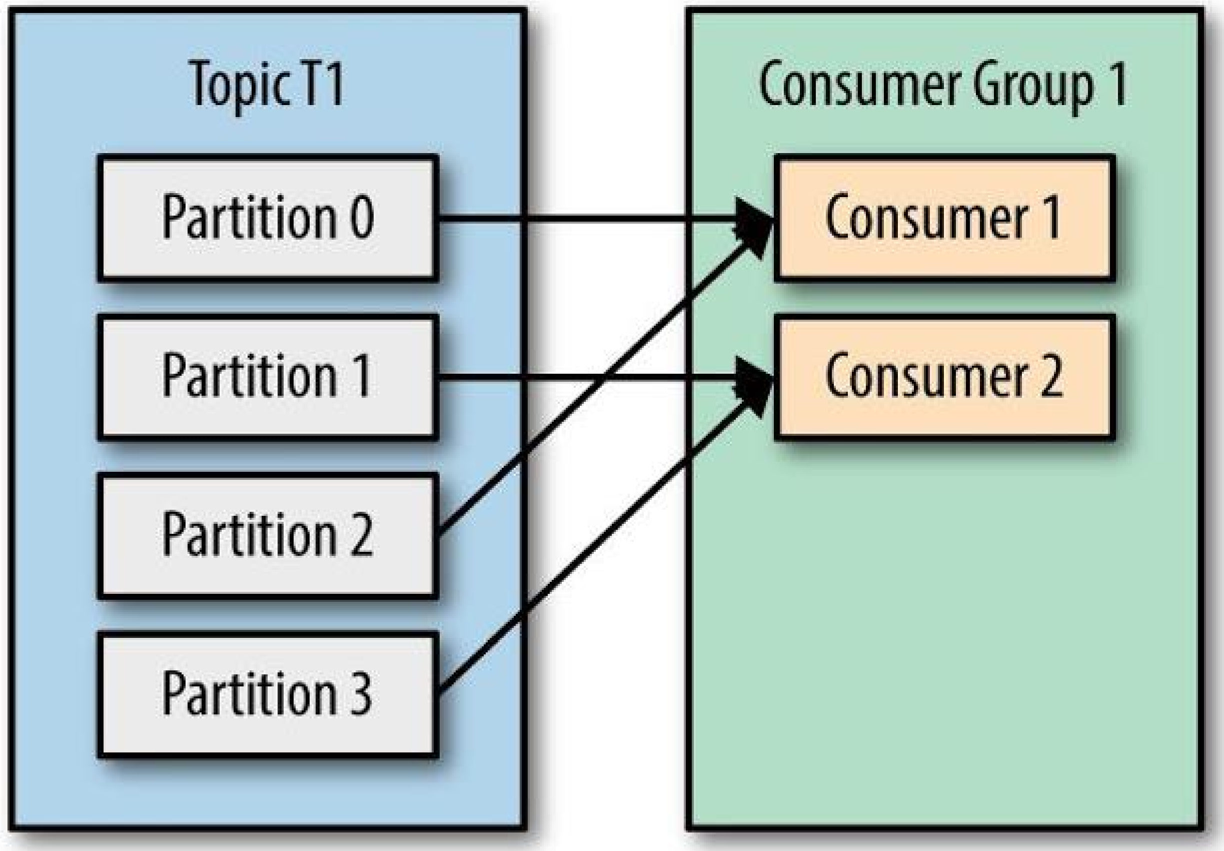
Suppose you have an application that needs to read messages from a Kafka topic, run some validations against them, and write the results to another data store. In this case your application will create a consumer object, subscribe to the appropriate topic, and start receiving messages, validating them and writing the results. This may work well for a while, but what if the rate at which producers write messages to the topic exceeds the rate at which your application can validate them? If you are limited to a single consumer reading and processing the data, your application may fall farther and farther behind, unable to keep up with the rate of incoming messages. Obviously there is a need to scale consumption from topics. Just like multiple producers can write to the same topic, we need to allow multiple consumers to read from the same topic, splitting the data between them.

Kafka consumers are typically part of a consumer group. When multiple consumers are subscribed to a topic and belong to the same consumer group, each consumer in the group will receive messages from a different subset of the partitions in the topic.

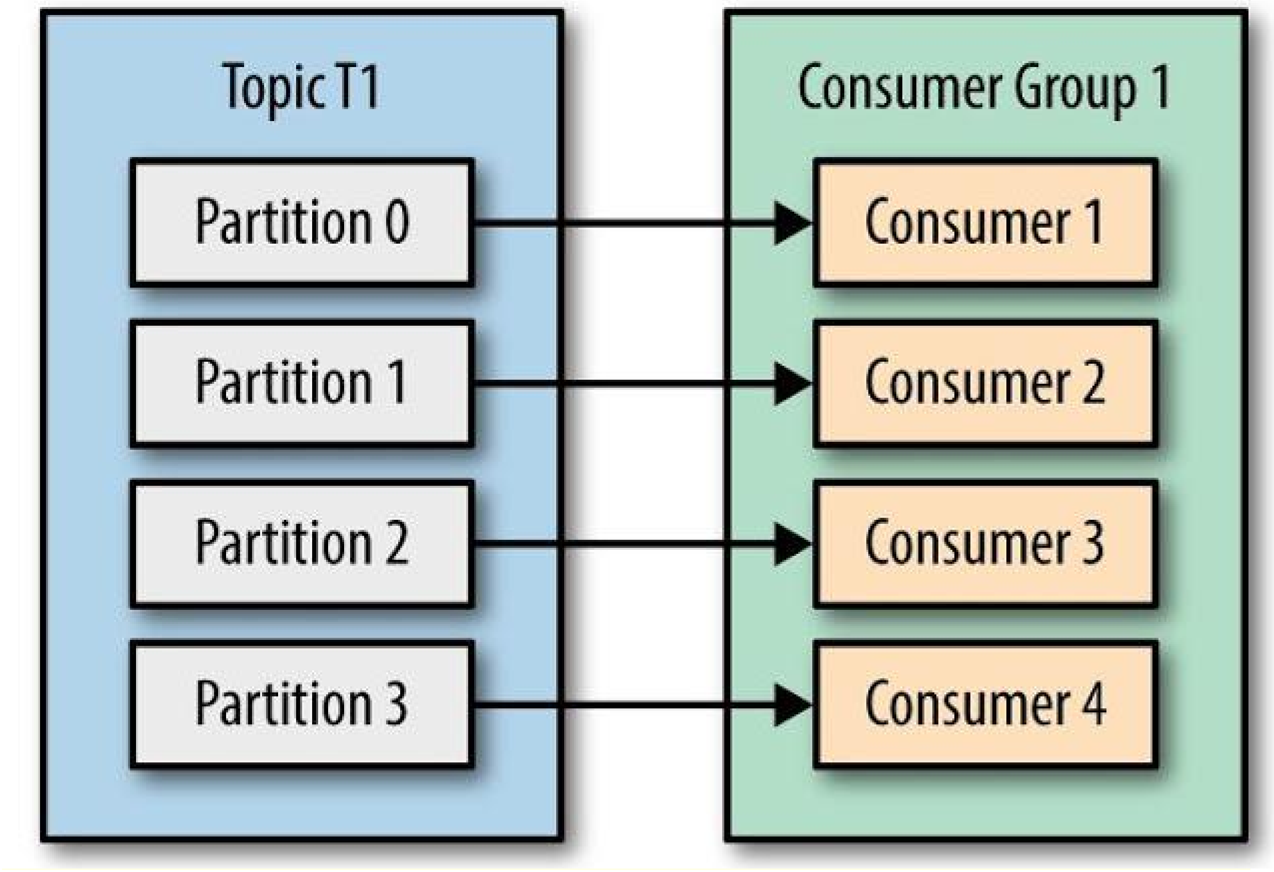
Let's take topic T1 with four partitions. Now suppose we created a new consumer, C1, which is the only consumer in group G1, and use it to subscribe to topic T1. Consumer C1 will get all messages from all four T1 partitions.



If we add another consumer, C2, to group G1, each consumer will only get messages from two partitions. Perhaps messages from partition 0 and 2 go to C1 and messages from partitions 1 and 3 go to consumer C2.



If G1 has four consumers, then each will read messages from a single partition.



If we add more consumers to a single group with a single topic than we have partitions, some of the consumers will be idle and get no messages at all.

In addition to adding consumers in order to scale a single application, it is very common to have multiple applications that need to read data from the same topic. In fact, one of the main design goals in Kafka was to make the data produced to Kafka topics available for many use cases throughout the organization.

In those cases, we want each application to get all of the messages, rather than just a subset. To make sure an application gets all the messages in a topic, ensure the application has its own consumer group. Unlike many traditional messaging systems, Kafka scales to a large number of consumers and consumer groups without reducing performance.

In the previous example, if we add a new consumer group G2 with a single consumer, this consumer will get all the messages in topic T1 independent of what G1 is doing.

Consumer Groups and Partition Rebalance

As we saw in the previous section, consumers in a consumer group share ownership of the partitions in the topics they subscribe to. When we add a new consumer to the group, it starts consuming messages from partitions previously consumed by another consumer. The same thing happens when a consumer shuts down or crashes; it leaves the group, and the partitions it used to consume will be consumed by one of the remaining consumers. Reassignment of partitions to consumers also happen when the topics the consumer group is consuming are modified (e.g., if an administrator adds new partitions).

Moving partition ownership from one consumer to another is called a *rebalance*. Rebalances are important because they provide the consumer group with high availability and scalability (allowing us to easily and safely add and remove consumers), but in the normal course of events they are fairly undesirable.

During a rebalance, consumers can't consume messages, so a rebalance is basically a short window of unavailability of the entire consumer group. In addition, when partitions are moved from one consumer to another, the consumer loses its current state; if it was caching any data, it will need to refresh its caches—slowing down the application until the consumer sets up its state again. Throughout this chapter we will discuss how to safely handle rebalances and how to avoid unnecessary ones.

The way consumers maintain membership in a consumer group and ownership of the partitions assigned to them is by sending *heartbeats* to a Kafka broker designated as the *group coordinator* (this broker can be different for different consumer groups). As long as the consumer is sending heartbeats at regular intervals, it is assumed to be alive, well, and processing messages from its partitions. Heartbeats are sent when the consumer polls (i.e., retrieves records) and when it commits records it has consumed.

If the consumer stops sending heartbeats for long enough, its session will time out and the group coordinator will consider it dead and trigger a rebalance. If a consumer crashed and stopped processing messages, it will take the group coordinator a few seconds without heartbeats to decide it is dead and trigger the rebalance.

During those seconds, no messages will be processed from the partitions owned by the dead consumer. When closing a consumer cleanly, the consumer will notify the group coordinator that it is leaving, and the group coordinator will trigger a rebalance immediately, reducing the gap in processing. Later in this chapter we will discuss configuration options that control heartbeat frequency and session timeouts and how to set those to match your requirements

How does the process of assigning partitions to brokers work?

When a consumer wants to join a group, it sends a JoinGroup request to the group coordinator. The first consumer to join the group becomes the group *leader*. The leader receives a list of all consumers in the group from the group coordinator (this will include all consumers that sent a heartbeat recently and which are therefore considered alive) and is responsible for assigning a subset of partitions to each consumer. It uses an implementation of PartitionAssignor to decide which partitions should be handled by which consumer.

Creating a Kafka Consumer:

The first step to start consuming records is to create a KafkaConsumer instance. Creating a KafkaConsumer is very similar to creating a KafkaProducer—you create a Java Properties instance with the properties you want to pass to the consumer. We will discuss all the properties in depth later in the chapter.

To start we just need to use the three mandatory properties: bootstrap.servers, key.deserializer, and value.deserializer.

The first property, bootstrap.servers, is the connection string to a Kafka cluster. The other two properties, key.deserializer and value.deserializer, are similar to the serializers defined for the producer, but rather than specifying classes that turn Java objects to byte arrays, you need to specify classes that can take a byte array and turn it into a Java object.

The following code snippet shows how to create a KafkaConsumer:

```
Properties props = new Properties();

props.put("bootstrap.servers", "broker1:9092,broker2:9092");

props.put("group.id", "CountryCounter");

props.put("key.deserializer",

    "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer",

    "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer =

    new KafkaConsumer<String, String>(props);
```

We assume that the records we consume will have String objects as both the key and the value of the record. The only new property here is group.id, which is the name of the consumer group this consumer belongs to.

Subscribing to Topics

Once we create a consumer, the next step is to subscribe to one or more topics. The subscribe() method takes a list of topics as a parameter, so it's pretty simple to use:

```
consumer.subscribe(Collections.singletonList("customerCountries"));
```

The Poll Loop

At the heart of the consumer API is a simple loop for polling the server for more data. Once the consumer subscribes to topics, the poll loop handles all details of coordination, partition rebalances, heartbeats, and data fetching, leaving the developer with a clean API that simply returns available data from the assigned partitions.

```
try {

    while (true) {

        ConsumerRecords<String, String> records = consumer.poll(100);

        for (ConsumerRecord<String, String> record : records)

        {

            log.debug("topic = %s, partition = %d, offset = %d,"

                customer = %s, country = %s\n",

                record.topic(), record.partition(), record.offset(),

                record.key(), record.value());



            int updatedCount = 1;

            if (custCountryMap.countainsKey(record.value())) {

                updatedCount = custCountryMap.get(record.value()) + 1;

            }

            custCountryMap.put(record.value(), updatedCount)

            JSONObject json = new JSONObject(custCountryMap);

            System.out.println(json.toString(4));

    }
}

}finally{

    consumer.close();

}
```

This is indeed an infinite loop. Consumers are usually long-running applications that continuously poll Kafka for more data. We will show later in the chapter how to cleanly exit the loop and close the consumer.

This is the most important line in the chapter. The same way that sharks must keep moving or they die, consumers must keep polling Kafka or they will be considered dead and the partitions they are consuming will be handed to another consumer in the group to continue consuming. The parameter we pass, poll(), is a timeout interval and controls how long poll() will block if data is not available in the consumer buffer. If this is set to 0, poll() will return immediately; otherwise, it will wait for the specified number of milliseconds for data to arrive from the broker.

poll() returns a list of records. Each record contains the topic and partition the record came from, the offset of the record within the partition, and of course the key and the value of the record. Typically we want to iterate over the list and process the records individually.

Processing usually ends in writing a result in a data store or updating a stored record. Here, the goal is to keep a running count of customers from each county, so we update a hashtable and print the result as JSON. A more realistic example would store the updates result in a data store.

CAP Theorem and external configuration in microservices

Question 1:

What is CAP Theorem?

Answer:

- **Consistency** — A guarantee that every node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.
- **Availability** — Every non-failing node returns a response for all read and write requests in a reasonable amount of time. The key word here is every. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.

- **Partition Tolerant** — The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

The C and A in ACID represent different concepts than C and A in the CAP theorem.

The CAP theorem categorizes systems into three categories:

- CP (Consistent and Partition Tolerant) — At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition.
- CA (Consistent and Available) — CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.
- AP (Available and Partition Tolerant) — These are systems that are available and partition tolerant but cannot guarantee consistency.

Question 2:

How to do externalized configuration in microservices?

Answer:

Why do we need this?

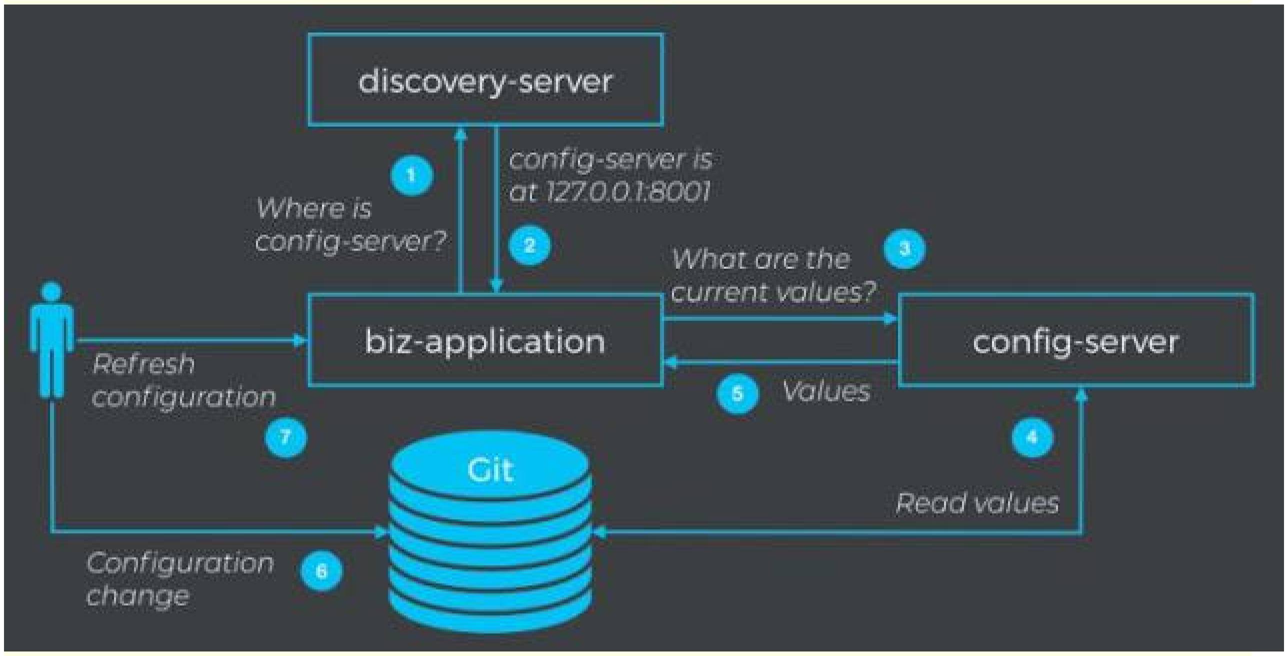
With microservices, applications are split into several services (microservices), each one usually running in a separate process. Each process can be deployed and scaled independently, meaning that there may be several replicas of the a microservice running at a certain time.

Let’s say we want to modify the configuration for a microservice that has been replicated a hundred times (one hundred processes are running). If the configuration for this microservice is packaged with the microservice itself, we’ll have to redeploy each of the one hundred instances. This can result in some instances using the old configuration, and some using the new one.

Moreover, sometimes microservices use external connections which, for example, require URLs, usernames, and passwords. If you want to update these settings, it would be useful to have this configuration shared across services.

How does it work?

Externalized configuration works by keeping the configuration information in an external store, such as a database, file system, or environment variables. At startup, microservices load the configuration from the external store. During runtime, microservices provide an option to reload the configuration without having to restart the service.



Implementing a configuration server with Spring Cloud Config

There are many ways to implement externalized configuration. Netflix's Archaius and Spring Cloud offer ready-to-use and well-tested solutions. Cloud services and platforms such as AWS and Kubernetes offer similar services, as well. The demo application uses Spring Cloud Config which includes both the server and the client part of the equation.

Use the Spring Initializr to create a new Spring Boot application named config-server and include the **Eureka Discovery**, **Config Server**, **Actuator** (optional), and Config Server dependencies:

Open up the ConfigServerApplication class and activate the discovery client and the configuration server by using the following annotations:

```
@SpringBootApplication

@EnableConfigServer

@EnableDiscoveryClient

public class ConfigServerApplication {

  ...
}
```

Remove the application.properties file and create a new application.yml file with the following content

```
server.port: 8101

spring:
  application.name: config-server
  cloud.config:
    server.git:
      uri: https://github.com/alejandro-du/vaadin-microservices-demo-config.git
      default-label: master

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8001/eureka/
    registryFetchIntervalSeconds: 1
  instance:
    leaseRenewalIntervalInSeconds: 1




}
```