CITI (03-Feb-2021) (1 hour 30 min around)

Java -

1. What are the OOPS concept? And their example?
2. Overriding vs overloading?
3. Is it possible to widen access modifiers in overriding? What about exception?

* The overriding method must not have more restrictive access modifier.
* If the overridden method is has default access, then the overriding one must be default, protected or public.
* If the overridden method is protected, then the overriding one must be protected or public.
* If the overridden method is public, then the overriding one must be only public.



1. Why Runtime exception can be thrown from overridden method but not broader checked exception?

Its because of runtime Polymorphism (Abstraction) (One of the central concepts of OOP is using abstract types, and that all subtypes may be treated as the abstract type)

The reason you can't introduce broader behaviour is that if the method from the abstract type (super class or interface) doesn't throw an Exception and you refer to your object as that type, you'd get unexpected behaviour:

**Alpha alpha = new Beta();**

**// At this point, the compiler knows only that we have an Alpha**

**alpha.myMethod();**

If Alpha's myMethod() doesn't throw an Exception, but Beta's *does*, we could get an unexpected Exception in the above code.

Anyways Runtime Exception does not concern compiler its programmers’ fault but Checked Does

1. Can we override static methods? No – Why?

Because its part of class not instance

1. Where to use array list vs linked list?
2. How does HashMap internally work?
3. What happens when hash collision occurs?
4. If my HashMap is working slow, what could be the reason?
5. What is Semaphore?

*A semaphore controls access to a shared resource through the use of a counter. For example we what a class to connect with max 10 connections not more than that at a time*

1. What is Executors Framework? How does it work?
2. How synchronized block works? (monitors)? Class level and object level locks?

**Object Level Locks −** It can be used when you want non-static method or non-static block of the code should be accessed by only one thread.

public class ClassLevelLockExample {

   public void classLevelLockMethod() {

      synchronized (ClassLevelLockExample.class) {

         //DO your stuff here

      }

   }

}

**Class Level locks −** It can be used when we want to prevent multiple threads to enter the synchronized block in any of all available instances on runtime. It should always be used to make static data thread safe.

public class ObjectLevelLockExample {

   public void objectLevelLockMethod() {

      synchronized (this) {

         //DO your stuff here

      }

   }

}

1. What design pattern you used?
2. What is Façade design pattern?
3. Should the HashMap key object be immutable? Why?

*Because If I change the key then hashcode of that key will change in map and I won’t able to find the value.*

1. How to make a class immutable?
2. If one class say Employee has account list, then how to make the employee class immutable?

you can turn the List immutable by decorating it using the Collections class:

list = Collections.unmodifiableList(list);

1. Abstract class vs interface? Where to use one over another?

**Abstract classes** should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated **classes**.

An **abstract class** allows **you to** create functionality that subclasses **can** implement or override. An **interface** only allows **you to** define functionality, not implement it. And whereas a **class can** extend only **one abstract class**, it **can** take advantage of multiple **interfaces**

1. Can abstract class have constructor? What is the use of it if we cannot instantiate it?

The main **purpose** of the **constructor** is to initialize the newly created object. In **abstract class**, we have an instance variable, **abstract** methods, and non-**abstract** methods. We need to initialize the non-**abstract** methods and instance variables, therefore **abstract classes** have a **constructor ,IT will be called from subclass via super**

Spark -

1. How do you get SparkContext/Session in your program?

**SparkConf** sparkConf = new SparkConf().setAppName("Word Count Demo").setMaster("local");

**JavaSparkContext** jsc = **new** JavaSparkContext(sparkConf);

**SparkSession** spark = SparkSession

.*builder*()

.appName("Java Spark SQL basic example")

.config("spark.master", "local")

.getOrCreate();

SparkContext sc = spark.sparkContext();

What is SparkContext? - SparkContext (aka **Spark context**) is the entry point to Spark for a Spark application. It [sets up internal services](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext-creating-instance-internals.html) and establishes a connection to a [Spark execution environment (deployment mode)](https://mallikarjuna_g.gitbooks.io/spark/content/spark-deployment-environments.html).

Once a [SparkContext instance is created](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html" \l "creating-instance) you can use it to [create RDDs](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html#creating-rdds), [accumulators](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html#creating-accumulators) and [broadcast variables](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html#broadcast), access Spark services and [run jobs](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html#runJob) (until SparkContext is [stopped](https://mallikarjuna_g.gitbooks.io/spark/content/spark-sparkcontext.html#stop)).

A Spark context is essentially a client of Spark’s execution environment and acts as the master of your Spark application (don’t get confused with the other meaning of [Master](https://mallikarjuna_g.gitbooks.io/spark/content/spark-master.html) in Spark, though).

1. What all are the context used in your current spark job? (spark,sql,hive context etc)
2. Where does spark conf reside? (file path of spark-default.conf ?)

The default Spark properties file is [$SPARK\_HOME/conf/spark-defaults.conf](https://mallikarjuna_g.gitbooks.io/spark/content/spark-properties.html#spark-defaults-conf) that could be overriden using spark-submit's [--properties-file command-line option](https://mallikarjuna_g.gitbooks.io/spark/content/spark-submit.html#properties-file).

<https://mallikarjuna_g.gitbooks.io/spark/content/spark-properties.html>

1. What all are the properties in that file?

spark.submit.deployMode setting can be client or cluster

# Example:

# spark.master spark://master:7077

# spark.eventLog.enabled true

# spark.eventLog.dir hdfs://namenode:8021/directory

# spark.serializer org.apache.spark.serializer.KryoSerializer

# spark.driver.memory 5g

# spark.yarn.historyServer .address=http:

# spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"

1. What is your cluster configurations?
2. How much your current Aggregation job takes time?
3. What is executor’s memory used in your cluster?
4. What is RDD? What is Dataframe and Dataset?
5. To create DataFrame don’t need to define object type? – No
6. How does Spark fast then MapReduce? - In memory
7. Which resource manager you are using? Why did you use YARN ? – yarn is better suited for Hadoop cluster, and if you already have hadoop cluster then yarn is better choice.
8. What is the purpose of Default and static method in java 8.

*Java 8 allows the interfaces to have default and static methods. The reason we have default methods in interfaces is to allow the developers to add new methods to the interfaces without affecting the classes that implements these interfaces.*

*For example, if several classes such as A, B, C and D implements an interface XYZInterface then if we add a new method to the XYZInterface, we have to change the code in all the classes(A, B, C and D) that implements this interface.  imagine if there are hundreds of classes implementing an interface then it would be almost impossible to change the code in all  This is why in java 8, we have a new concept “default methods”. These methods can be added to any existing interface and we do not need to implement these methods in the implementation classes mandatorily, thus we can add these default methods to existing interfaces without breaking the code.*

*We can say that concept of default method is introduced in java 8 to add the new methods in the existing interfaces in such a way so that they are backward compatible. Backward compatibility is adding new features without breaking the old code.*

***Static methods****in interfaces are similar to the default methods except that we cannot override these methods in the classes that implements these interfaces.*