**Candidate Name: Sheraz Chaudhry**

**Date: 05/25/25**

**Role: Senior Java Engineer - 322291**

**Basic Programming Skills:**

Please complete at least 2 of the below:

BTW AI will likely solve these incorrectly/incompletely and/or produce a poor solution that would not be extensible. It’s OK to use any tool you like.

**#1** Write a Java console program that implements a basic search engine

to search:

declaration of independence us constitution magna Carta.

For example, the user can type:

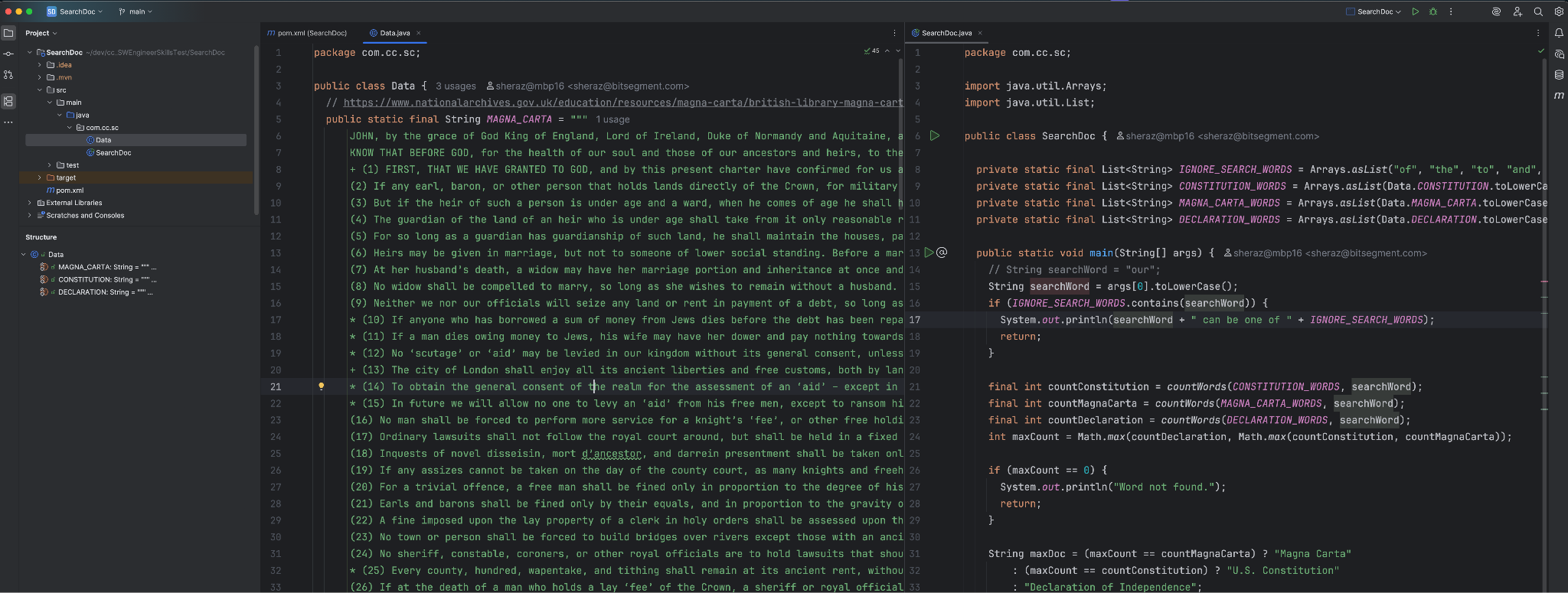
java SearchDoc our and it would print the word our occurs x times in Magna Carta

-because our occurs in the Magna Cart the most often.

**1.1** The program must exclude the following common words "of", "the", "to", "and”, “for".

I have created a separate Data.java. It contains 3 public constants ***CONSTITUTION, MAGNA\_CARTA*** and ***DECLARATION***.These contain texts of "Declaration of Independence" , "US constitution" and "Magna Carta".

It looks like this:



**package com.cc.sc;**

**import java.util.Arrays;**

**import java.util.List;**

**public class SearchDoc {**

**private static final List<String> *IGNORE\_SEARCH\_WORDS* = Arrays.*asList*("of", "the", "to", "and", "for");**

**private static final List<String> *CONSTITUTION\_WORDS* = Arrays.*asList*(Data.*CONSTITUTION*.toLowerCase().split("\\W+"));**

**private static final List<String> *MAGNA\_CARTA\_WORDS* = Arrays.*asList*(Data.*MAGNA\_CARTA*.toLowerCase().split("\\W+"));**

**private static final List<String> *DECLARATION\_WORDS* = Arrays.*asList*(Data.*DECLARATION*.toLowerCase().split("\\W+"));**

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

**// String searchWord = "our";**

**String searchWord = args[0].toLowerCase();**

**if (*IGNORE\_SEARCH\_WORDS*.contains(searchWord)) {**

**System.*out*.println(searchWord + " can be one of " + *IGNORE\_SEARCH\_WORDS*);**

**return;**

**}**

**final int countConstitution = *countWords*(*CONSTITUTION\_WORDS*, searchWord);**

**final int countMagnaCarta = *countWords*(*MAGNA\_CARTA\_WORDS*, searchWord);**

**final int countDeclaration = *countWords*(*DECLARATION\_WORDS*, searchWord);**

**int maxCount = Math.*max*(countDeclaration, Math.*max*(countConstitution, countMagnaCarta));**

**if (maxCount == 0) {**

**System.*out*.println("Word not found.");**

**return;**

**}**

**String maxDoc = (maxCount == countMagnaCarta) ? "Magna Carta"**

**: (maxCount == countConstitution) ? "U.S. Constitution"**

**: "Declaration of Independence";**

**System.*out*.printf("occurs %d times in %s.\n", maxCount, maxDoc);**

**}**

**private static int countWords(List<String> words, String searchWord) {**

**return (int) words.stream()**

**.filter(word -> word.contains(searchWord))**

**.count();**

**}**

**}**

**#2** Often times at our company external systems submit orders via REST API. Example payload for submitting an order is below.

An outletId is the id of the physical location that sells products. That is, an outlet id represents a physical location at a google maps location for restaurant or store front that sells our products.

[https://www.google.com/maps/dir/33.7704658,-](https://www.google.com/maps/dir/33.7704658,-84.40676/Moe's+Southwest+Grill,+1450+Ernest+W+Barrett+Pkwy+NW,+Kennesaw,+GA+30152/@33.8757647,-84.6562564,11z/data=!3m1!4b1!4m9!4m8!1m1!4e1!1m5!1m1!1s0x88f515c0fc91e969:0xd35d2fb924bdc235!2m2!1d-84.5930317!2d33.9956548?entry=ttu)

[84.40676/Moe's+Southwest+Grill,+1450+Ernest+W+Barrett+Pkwy+NW,+Kennesaw,+GA+30152/@33.87 57647,-](https://www.google.com/maps/dir/33.7704658,-84.40676/Moe's+Southwest+Grill,+1450+Ernest+W+Barrett+Pkwy+NW,+Kennesaw,+GA+30152/@33.8757647,-84.6562564,11z/data=!3m1!4b1!4m9!4m8!1m1!4e1!1m5!1m1!1s0x88f515c0fc91e969:0xd35d2fb924bdc235!2m2!1d-84.5930317!2d33.9956548?entry=ttu)

[84.6562564,11z/data=!3m1!4b1!4m9!4m8!1m1!4e1!1m5!1m1!1s0x88f515c0fc91e969:0xd35d2fb924b dc235!2m2!1d-84.5930317!2d33.9956548?entry=ttu](https://www.google.com/maps/dir/33.7704658,-84.40676/Moe's+Southwest+Grill,+1450+Ernest+W+Barrett+Pkwy+NW,+Kennesaw,+GA+30152/@33.8757647,-84.6562564,11z/data=!3m1!4b1!4m9!4m8!1m1!4e1!1m5!1m1!1s0x88f515c0fc91e969:0xd35d2fb924bdc235!2m2!1d-84.5930317!2d33.9956548?entry=ttu)

It’s possible that the client caller from external systems could fail to get the response due to network issues. That is the calling external system receives a socket exception, and the receiving coke system creates the order. This is problematic because the calling external system will retry, and subsequent calls to coke system creates a duplicate order.

You can see how this would create a problem for store manager at the outlet when their delivery arrives, and the delivery is for double their order, or triple (because there were 3 retries).

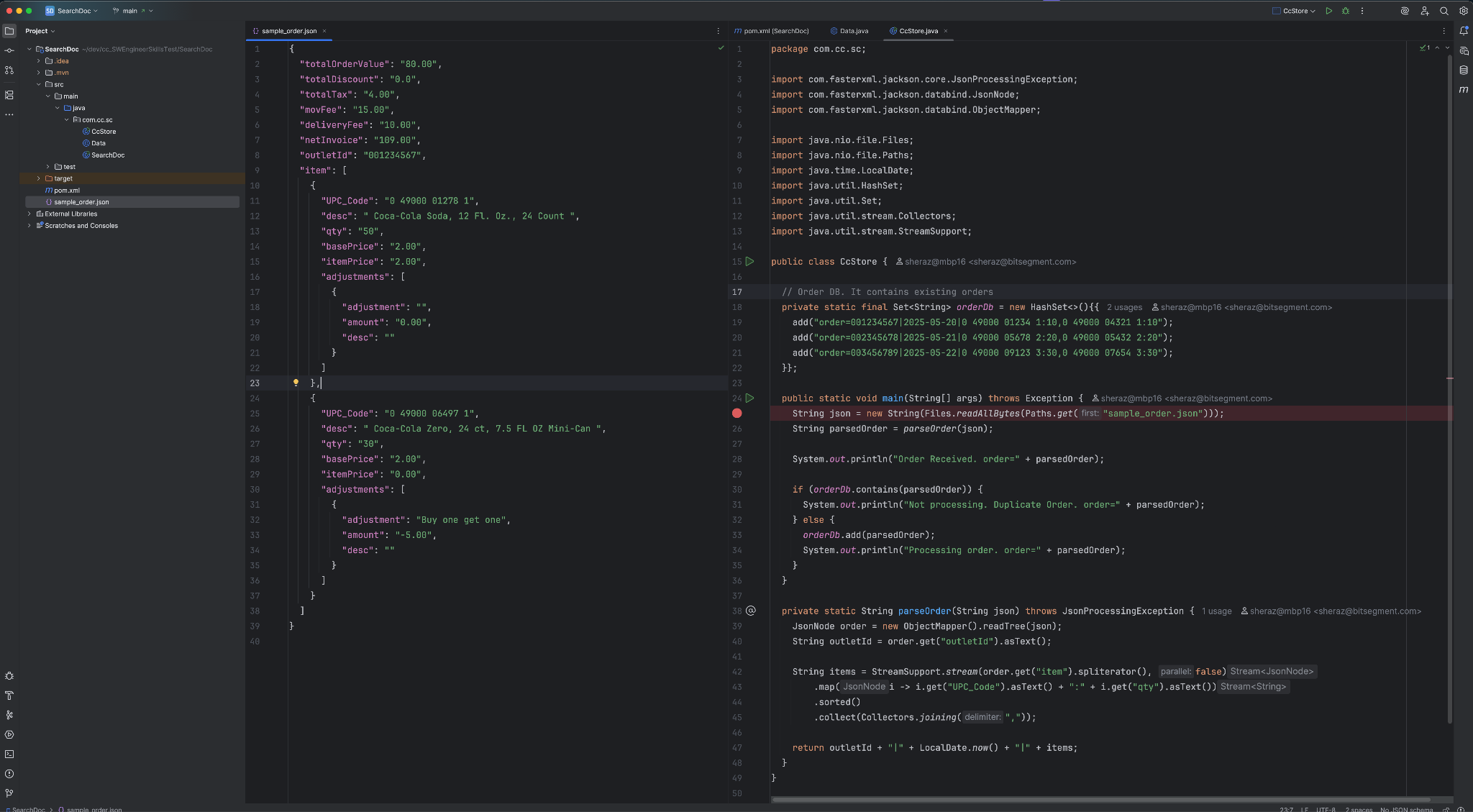
Assume you are writing an API that must process 1,000,000 orders a day and you are facing this problem.

It’s safe to assume that the same outlet would not place the same logical order twice on the same day of the year. Orders would be considered logically the same if they have the same mix of merchandise and quantity, other values in the payload could be different.

How would you write an API to prevent duplicates, (it must support the payload below)?

Write a simple java main that demonstrates you can parse the payload below and apply the key programming concepts needed to prevent duplicate orders.

I have created a separate sample\_order.json file it contains a sample order



**package com.cc.sc;**

**import com.fasterxml.jackson.core.JsonProcessingException;**

**import com.fasterxml.jackson.databind.JsonNode;**

**import com.fasterxml.jackson.databind.ObjectMapper;**

**import java.nio.file.Files;**

**import java.nio.file.Paths;**

**import java.time.LocalDate;**

**import java.util.HashSet;**

**import java.util.Set;**

**import java.util.stream.Collectors;**

**import java.util.stream.StreamSupport;**

**public class CcStore {**

**// Order DB. It contains existing orders**

**private static final Set<String> *orderDb* = new HashSet<>(){{**

**add("order=001234567|2025-05-20|0 49000 01234 1:10,0 49000 04321 1:10");**

**add("order=002345678|2025-05-21|0 49000 05678 2:20,0 49000 05432 2:20");**

**add("order=003456789|2025-05-22|0 49000 09123 3:30,0 49000 07654 3:30");**

**}};**

**public static void main(String[] args) throws Exception {**

**String json = new String(Files.*readAllBytes*(Paths.*get*("sample\_order.json")));**

**String parsedOrder = *parseOrder*(json);**

**System.*out*.println("Order Received. order=" + parsedOrder);**

**if (*orderDb*.contains(parsedOrder)) {**

**System.*out*.println("Not processing. Duplicate Order. order=" + parsedOrder);**

**} else {**

***orderDb*.add(parsedOrder);**

**System.*out*.println("Processing order. order=" + parsedOrder);**

**}**

**}**

**private static String parseOrder(String json) throws JsonProcessingException {**

**JsonNode order = new ObjectMapper().readTree(json);**

**String outletId = order.get("outletId").asText();**

**String items = StreamSupport.*stream*(order.get("item").spliterator(), false)**

**.map(i -> i.get("UPC\_Code").asText() + ":" + i.get("qty").asText())**

**.sorted()**

**.collect(Collectors.*joining*(","));**

**return outletId + "|" + LocalDate.*now*() + "|" + items;**

**}**

**}**

How would you check for duplicates in a real system that scales?

There can be few options you can use.

**1. Distributed Async Processing:**

Make a distributed system by utilizing Kafka. By using Kafka, order receiving service will not slow down.

Create an Order Service that uses any RDBMS DB and stores unique orders for the last 24 hours. This service will help detect duplicate orders.

Delivery service will deliver the order.

Order request -> order queue -> Order Service (filter duplicate) -> Delivery service

**2. Redis Cache:**

Create a Redis cache that has TTL of 24 hours. Check for duplicates and in the redis cache. If duplicate is not found then store order in redis cache and deliver the product.

Assume that the same outlet would not intentionally place the same order for all the same products and quantities twice in one day.

{

"totalOrderValue": "80.00",

"totalDiscount": "0.0",

"totalTax", "4.00",

"movFee", "15.00",

"deliveryFee", "10.00",

"netInvoice", "109.00",

“outletId”: “001234567”,

"item": [

{

"UPC\_Code": "0 49000 01278 1",

"desc": " Coca-Cola Soda, 12 Fl. Oz., 24 Count ",

"qty": "50",

"basePrice": "2.00",

"itemPrice": "2.00",

"adjustments": [

{

"adjustment": "",

"amount": "0.00",

"desc": ""

},

{

" UPC\_Code": "049000064971 ",

"desc": " Coca-Cola Zero, 24 ct, 7.5 FL OZ Mini-Can ",

"qty": "30",

"basePrice": "2.00",

"itemPrice": "0.00", >>> Quantity\* Base price – sum (Adjustments) "adjustments": [

{

"adjustment": "Buy one get one",

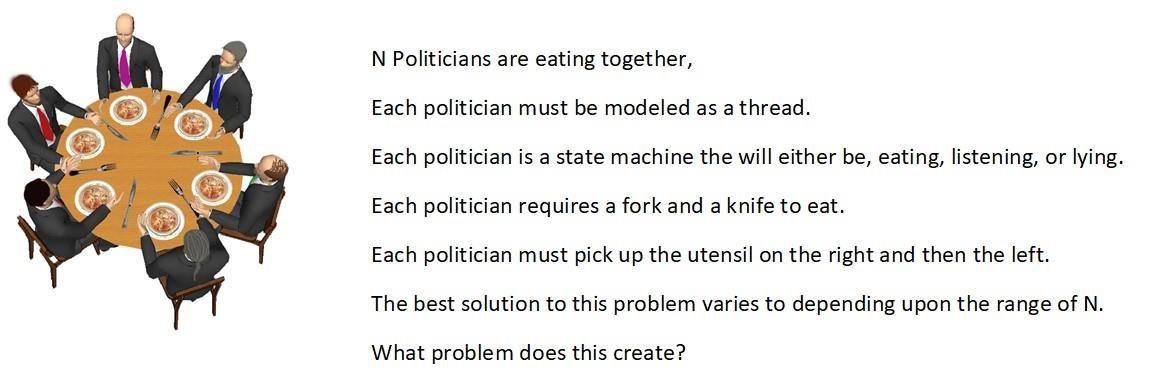
"amount": "-5.00",

"desc": ""

}

] }

**#3** Politicians are eating in very pre-covid arrangement.



What are some programming keywords you would use to solve this problem.

Write a simple java console program that solves this problem. Design your solution assuming N is from 3 – 20.

How might you consider changing your solution for accommodate N = 10000.

The best solution to this problem varies to depending upon the range of N.

**More Advanced skills:**

Complete as many as you can in **4 hours**:

Hint: If you’re just going to copy and paste an AI answer you probably shouldn’t bother. If you have done it before you know.

**#1 My SQL transaction query.**

There is a transaction table in in production that has

* a surrogate key named id that auto increments. - - A name which is a string < 254> - A create date.
* A Status that gets updated by various processes on 100 threads.

There are roughly 5 million records a day that are being added to this table (at an unsteady state… the ingress is highly burstable), and the junior developer who created this table didn’t make any indexes, and no data has been purged from this table in 2 years.

You need to perform a select distinct name on the most recent records to debug a problem in production.

How can you safely do this without impacting performance?

Not creating indexes will take a too long and create a performance risk that would have to be scheduled for a later outage window.

**#2 My SQL transaction clean up?**

You need to clean the mess that was left for you in #1.

The prod system that you are using must be up 24/7/365. You don’t get an outage window.

You need to delete all the transaction data older than 1 month and add indexes. How do you do it without impacting production performance?

**#3 My SQL truncate vs delete.**

What is the difference between truncate and delete?

When do you use one vs the other?

When performing batch integrations, its often common to use truncate and select together. Give an example of when you would to this?

How does this relate back to #2 above?

**#4 You have a java process running on a Linux EC2 in production that is running out of memory daily in production even though there are 16 GB allocated to the JVM.**

You cannot take production down, and you cannot replicate the problem in other environments.

How would you approach this problem?

What steps would you take to mitigate this issue in production?

What tools would you use to capture data?

What tools would you use to analyze the data?

Get as specific as you possible can with java command line arguments, tools and screen shots.

Write an example java program that will generate an oom (out of memory) to prove your method will work.

Again, you need to keep prod up 24/7/365 you can’t take prod down for more than 5 mins at time.

**#5 You have multithreaded java application where the threads appear to dead lock in production.**

You believe it is deadlocked because the app stops processing requests, and CPU is zero, memory is normal, and the java process is running.

This seems to happen every 24 -48 hours on no recurring schedule.

You can’t reproduce in lower environments. What do you do to solve this problem?

What kind of commands do you run?

What kind of tools do you use?

Again, you need to keep prod up 24/7/365 you can’t take prod down for more than 5 mins at time.

Alter your solution to #3 in basic skills test to prove you can make it deadlock, take screen shots of your thread dumps.