**JPMC Cohort Software Engineer II & III**

**Interview Guide**

[**Overview 3**](#_j5m3krzeqohb)

[**Coding Assessment Questions 4**](#_xsiqurmbs97g)

[1. Pairs 4](#_mxr592nwho7e)

[2. Chess Tournament 5](#_qj41ln61xwy6)

[3. Server Selection 7](#_iir1aajobxes)

[4. The Organized Shop 9](#_3o9jri3w0kng)

[5. System and Strings 11](#_c7zgndcypint)

[6. Minimum Total Cost 13](#_oqa02b87md98)

[7. Substring Removal 15](#_w2elnajnctwg)

[**Super Day Questions 17**](#_l57cq9irg5ci)

[Behavioral Interview 17](#_u0d68530zk3u)

[Coding Interview 18](#_kpz5i12dywrx)

[Analysis & Improvement 18](#_8r806fbliebh)

[Code Snippet 18](#_w0jjuy32vcts)

[Improvements 22](#_j43s7kv4t6hg)

[1. Static Initialization of API\_SERVERS 22](#_abydri1mut1t)

[2. Thread Safety and Volatility 22](#_u9qowdjoicdv)

[3. LinkedList Usage 23](#_4ttw2l9o6mj0)

[4. Error Handling in call\_url 23](#_r8nn4mgpiqs2)

[5. Inconsistent Error Handling 23](#_ss0q6m4x19s6)

[6. String Comparison 24](#_2oh6f81nad6q)

[7. Redundant Initialization 24](#_rrupyrnc6spi)

[8. Inefficient Looping 25](#_qxaefynks9r5)

[9. ApiResponse Class 25](#_3hknyytmk1n5)

[10. Method Documentation 26](#_dayln6j1lmq3)

[11. Consider Using a Logger 26](#_sxzmix3mtd4p)

[12. Space and Time Complexity Analysis 26](#_jbfa1tab2kzi)

[13. Resource Management & Exception Handling 26](#_tms3h26t8emj)

[14. Other 28](#_dicr1n9anaik)

[Algorithm Question 29](#_x2wiuhw9iqg9)

[Count Substrings Having Distinct Characters 29](#_hndmfak8nnin)

[Systems Design 31](#_d68teq157cu2)

[AI Parking System 31](#_klz70rbsqwby)

## Overview

JPMC Cohort hiring is standardized hiring where each candidate will do a coding assessment screen (generally hackerrank), once that is cleared they will proceed to a “Super Day” which is JPMC’s version of virtual onsite day.

The Super Day consists of three rounds: Coding, Systems Design, and Behavioral.

## Coding Assessment Questions

### Pairs

Consider two arrays of integers, a[n] and b[n]. What is the maximum number of pairs that can be formed where a[i] > b[j]? Each element can be in no more than one pair.

Find the maximum number of such possible pairs.

**Example:**

N = 3, A = [1,2,3], B = [1,2,1]

Two ways the maximum number of pairs can be selected:

* {a[1], b[0]} = {2,1} and {a[2], b[2]} = {3,1} are valid pairs
* {a[1], b[0]} = {2,1} and {a[2], b[1]} = {3,2} are valid pairs.

No more than 2 pairs can be formed, so return 2

**Answer:** Question\_1\_Pairs

| **public** **static** **int** **findNumOfPairs**(List<Integer> a, List<Integer> b) {   **int** counter = 0;  Collections.sort(a);  Collections.sort(b);  **int** i = a.size() - 1, j = b.size() - 1;  **while** (i >= 0 && j >= 0) {  **if** (a.get(i) > b.get(j)) {  counter++;  i--;  j--;  } **else** {  j--;  }  }   **return** counter;  } |
| --- |

### Chess Tournament

The city of Hackerland organized a chess tournament for its citizens.

There are *n* participants numbered 1 to *n* where the participant has potential denoted by *potential[i]*. The potential of each player is distinct. Initially, all players stand in a queue in order from the 1st to the player. In each game, the first 2 participants of the queue compete and the participant with a higher potential wins the game. After each game, the winner remains at the beginning of the queue and plays with the next person from the queue and the losing player goes to the end of the queue. The game continues until a player wins *k* consecutive games.

Given the potential of the participants and the deciding factor *k*, find the potential of the winning player.

**Example:**

Consider *n = 4* participants have *potential = [3,2,1,4]* and *k = 2*.

* Initial position of participants: [1,2,3,4]
* Participants 1 and 2 compete. Their potentials are 3 and 2. Player 1 wins due to the higher potential. Player 1 stays at the front of the queue and player 2 moves to the back. Now their positions are [1,3,4,2]
* Participants 1 and 3 compete. Their potentials are 3 and 1. 1 wins a second consecutive game. Since k = 2, player 1 has won enough consecutive games.

Return player 1’s potential, 3.

**Answer**: Question\_2\_Chess\_Tournament

| **public** **static** **int** **getPotentialOfWinner**(List<Integer> potential, **long** k) {  Queue<Integer> queue = **new** ArrayDeque<>(potential);  **int** winner = -1;  **long** kCount = k;   **int** p1 = -1, p2 = -1;  **while** (**true**) {  **if** (p1 == -1) {  p1 = queue.poll();  }  **if** (p2 == -1) {  p2 = queue.poll();  }  **if** (p1 > p2) {  queue.add(p2);  **if** (winner != p1) {  kCount = k;  }  winner = p1;  p2 = -1;  } **else** {  queue.add(p1);  **if** (winner != p2) {  kCount = k;  }  winner = p2;  p1 = -1;  }   **if** (--kCount == 0) **return** winner;  } |
| --- |

### Server Selection

Some developers want to deploy their application on different servers with a load balancer in the front. There are *n* servers to choose from where the number of requests that can be handled by the server is *server[i]*. The number of requests served by any server is a power of 2 e.e. 1,2,4,8,16,...etc.

Given the array *server* and an integer *expected\_load*, find the minimum number of servers that must be chosen such that the total sum of requests served by all the chosen servers is exactly equal to the *expected\_load*. If there is no combination of servers that can serve exactly *expected\_load* requests, report -1 as the answer.

**Example:**

Suppose *n = 4, servers = [1,1,2,4], and expected\_load = 3.*

It is optimal to choose the first and the third or the second and the third servers serving a total of 1 + 2 = *expected\_load*  = 3 requests. Return the minimum number of servers needed, 2.

**Answer**: Question\_3\_Server\_Selection

| **public** **static** **int** **getMinServers**(**int** expectedLoad, **int**[] server) {  *// Sort the server array in descending order*  *// Since Arrays.sort does not support reverse order on primitives, we will sort normally*  Arrays.sort(server);   **int** minServers = 0;  **int** totalRequests = 0;  **int** i = server.length - 1; *// Start from the largest element*   *// Iterate through the sorted servers to find the minimum number of servers needed*  **while** (totalRequests < expectedLoad && i >= 0) {  **if** (totalRequests + server[i] <= expectedLoad) {  totalRequests += server[i];  minServers++;  }  i--;  }   *// If the total requests match the expected load, return the number of servers*  **if** (totalRequests == expectedLoad) {  **return** minServers;  } **else** {  **return** -1;  }  } |
| --- |

### The Organized Shop

The owner of HackerMail loves organized items. A row of items is *organized* if the parity (even or odd) is different for each adjacent stack of items. To organize the row, half of the items in any stack can be removed. This can happen as many times and on as many stacks as is required. Determine the minimum number of operations needed to organize a row.

More formally, given an array *items[i]* of integer of length *n,* the array is organized if each x less than *n - 1, items[x]* mod 2 != *items[x + 1]* mod 2. A mod B is the remainder of A divided by B. In one operation, the owner can choose an element and divide it by 2. That is, if one chooses index x, then do *items[x] = floor(items[x]/2)*. The goal is to return the minimum number of operations that one needs to perform to organize the array.

**Example:**

*Items = [4,10,10,6,2]*

The array is not organized since, for example, *items[2]* mod 2 = *items[3]* mod 2.

One way to organize the array is shown using 1-based indexing:

* Choose the index and divide it by 2; the new array is [4,5,10,6,2].
* Choose the index and divide it by 2; the new array is [4,5,10,3,2].
* [4,5,10,3,2] is an organized array so return the number of operations, 2.

**Answer**: Question\_4\_The\_Organized\_Shop

| */\*\*  \* Does not pass all test cases  \*/*  **public** **static** **int** **getMinimumOperations**(List<Integer> items) {  **int**[] itemsArr = items.stream().mapToInt(Integer::intValue).toArray();  **int** n = itemsArr.length;  **int** operations = 0;  **for** (**int** i = 0; i < n - 1; i++) {  **while** ((itemsArr[i] % 2) == (itemsArr[i + 1] % 2)) {  itemsArr[i + 1] = (**int**) Math.floor(itemsArr[i + 1] / 2.0);  operations++;  }  }   **return** operations;  } |
| --- |

### System and Strings

Alex and Chris are learning to infiltrate a secure system. Starting with a string of code, Alex will remove any substring with an odd number of vowels. Chris then removes any substring from the remaining code with an even number of vowels. This learning continues until one of them is unable to make a move. Given an array of n strings of code, determine the winner of each round and report the results as either “Alex” or “Chris” accordingly. Assume both act optimally.

Note: The vowels considered are ‘a’, ‘e’, ‘i’, ‘o’, and ‘u’

**Example**:

Given n = 3 and s = [“ljis”, “lhr”, “gms”]

* In the 1st round, the whole string “ljis” contains 1 vowel (odd) so Alex chooses the whole string. Now the string is “” (empty), and Chris cannot make a move, hence Alex wins the round
* In the 2nd round the string “lhr” does not contain any vowel hence Alex cannot make a move. Therefore Chris wins the round
* In the 3rd round, the string “gms” does not contain any vowel, so again Chris wins the round

Hence the answer is [“Alex”, “Chris”, “Chris”]

**Answer**: Question\_5\_System\_And\_Strings

| **public** **static** List<String> **getWho**(List<String> s) {  List<String> result = **new** ArrayList<>();  String vowels = "aeiou";   **for** (String str : s) {  **int**[] vowelByIndex = **new** **int**[str.length() + 1];  vowelByIndex[0] = 0;  **int** vowelCount = 0;  **int** counter = 1;  **int** highestOddCounterIndex = -1;  **for** (**char** c : str.toCharArray()) {  **if** ((vowels.indexOf(c) != -1)) {  vowelCount++;  *// If odd mark the index*  **if** (vowelCount % 2 == 1) highestOddCounterIndex = counter;  vowelByIndex[counter++] = vowelCount;  } **else** {  vowelByIndex[counter++] = vowelByIndex[counter - 1];  }  }   *// Odd*  **if** (vowelCount % 2 == 1) {  result.add("Alex");  } **else** {  *// Remove highest odd counter substring*  **if** (highestOddCounterIndex == -1) {  *// No odd value found*  result.add("Chris");  } **else** {  **int** res = vowelByIndex[vowelByIndex.length - 1] - vowelByIndex[highestOddCounterIndex];  **if** (res % 2 == 0) {  *// If even Chris can remove entire string*  result.add("Chris");  } **else** {  *// If odd then has no move*  result.add("Alex");  }  }  }  }  **return** result;  } |
| --- |

### Minimum Total Cost

Given an array of integers, the goal is to make all the elements in the array have equal values by applying some number of operations.

The rules of the operations are:

* To apply an operation, one needs to choose a prefix of the array and an integer x (x can be negative)
* In this operation, add x to each element inside this prefix
* The cost of this operation is |x| (Absolute value of x)

For example, if the array is [1,4,2,1] and the prefix of length 2 and x = -3 are chosen, the array would now become [-2,1,2,1] and the cost of this operation would be |x| = |-3| = 3.

The total cost is the sum of costs of each operation applied. Find the minimum total cost of making all the elements in the array have equal value.

Note: It is guaranteed that there always exists a series of operations by which all elements in any array can be equal. These operations can be applied any number of times.

**Example**:

Consider n = 4 and arr = [1,2,1,5].

The array can be made equal using the following three operations:

* Choose the prefix of length 2 and x = -1. Hence the array now becomes [0,1,1,5]. The cost of this operation is 1.
* Choose the prefix of length 3 and x = 4. Hence the array now becomes [4,5,5,5]. The cost of this operation is 4.
* Choose the prefix of length 1 and x = 1. Hence the array now becomes [5,5,5,5] and the cost of this operation is 1.

Thus the total cost = 1 + 4 + 1 = 6, which is the minimum possible.

**Answer**: Question\_6\_Minimum\_Total\_Cost

| */\*\*  \* Only 6 out of 15 test cases pass. Times out  \*/*  **public** **static** **int** **findMinimumCost**(List<Integer> arr) {  **int** n = arr.size();  **int** totalCost = 0;   **for** (**int** i = n - 1; i > 0; i--) {  **if** (!arr.get(i).equals(arr.get(i-1))) {  **int** diff = arr.get(i) - arr.get(i - 1);  totalCost += Math.abs(diff);   **for** (**int** j = 0; j < i; j++) {  arr.set(j, arr.get(j) + diff);  }  }  }   **return** totalCost;  } |
| --- |

### Substring Removal

Given a string, seq, that consists of the characters ‘A’ and ‘B’ only, in one move, delete either an “AB”, or “BB” substring and concatenate the remaining substrings.

Find the minimum possible length of the remaining string after performing any number of moves.

Note: A substring is a contiguous subsequence of a string.

**Example**:

Seq = “BABBA”

Using 0-based indexing, the following moves are optimal.

* Delete the substring “AB” starting at index 1: “B**AB**BA” → “BBA”
* Delete the substring “BB” starting at index 0: “**BB**A” → “A”

There are no more moves, so the minimum possible length of the remaining string is 1.

**Answer**: Question\_7\_Substring\_Removal

| **public** **static** **int** **getMinLength**(String seq) {  StringBuilder sb = **new** StringBuilder(seq);  **boolean** found = **true**;  **while**(found) {  found = **false**;  **for** (**int** i = 0; i < sb.length() - 1; i++) {  *// Delete either AB or BB from string*  **if** ((sb.charAt(i) == 'A' && sb.charAt(i + 1) == 'B') || (sb.charAt(i) == 'B' && sb.charAt(i + 1) == 'B')) {  sb.delete(i, i + 2);  found = **true**;  **break**;  }  }  }  **return** sb.length();  } |
| --- |

## Super Day Questions

### Behavioral Interview

This interview does not have a specific format where they ask questions. So far it has been a discussion type of interview.

Behavioral Interview taken on August, 15, 2024 for Dylan Patel - 12 year profile

1. When you are presenting an idea, describe (very generic - do not need to share details of the design) an instance where someone looked at your design and they had another option. You felt that option did not fit into your scenario, what are some of the soft skills you used to persuade that person where you defended your design versus going with theirs. In addition, how do you defend your ideas? (what are some key aspects you use to defend your idea - e.g. solve a problem now by investing time or delegate it and keep having to spend time trying to fix it whenever it breaks)
2. How do you keep up on your skills?
3. It sounds like on your team sometimes you have to go and influence others to a new approach, how would you go about this? What type of artifacts/preparations do you do?
   1. I try to make a small PoC project where it implements the new approach I am trying to influence others with. I present this hopefully in a sandbox environment to show the benefits of whatever it is I am trying to influence others on.
4. As far as your interaction with Senior leaders, is there a time where you had to give a status report on where you are in a particular phase of a project or initiative, at a high level can you give a description of some of the information that you would include in that report?
5. You're interested in working at Chase, what kind of position would you be looking for?

### Coding Interview

This interview consisted of two parts. For part one we are told we are pretending a coworker sent us a pull request and we need to review it and add comments to it. Given the code, we have to provide recommendations and improvements.

#### Analysis & Improvement

##### Code Snippet

| **import** java.net.HttpURLConnection; **import** java.net.URL; **import** java.util.HashMap; **import** java.util.LinkedList; **import** java.util.Map; **import** javax.xml.parsers.DocumentBuilderFactory; **import** org.w3c.dom.Document;  **public** **class** **SendCustomerRequest** {  **public** **static** Map<String, String> API\_SERVERS = **new** HashMap<>() {  {  put("LOCAL", "http://127.0.0.1");  put("DEV", "http://cust-api-dev.internal.mycorp.com");  put("PROD", "http://cust-api.internal.mycorp.com");  }  };  **static** **private** LinkedList<SendCustomerRequest> ar = **new** LinkedList<>();   **static** **private** String pass;  **static** **protected** **volatile** **int** issued = 0;  **private** String pd\_ky;  **private** String cs\_ky;  **private** **int** q;  **private** String expedited = "false";  **private** **long** created;  **private** String rq\_ky;  **private** String is\_v;   SendCustomerRequest(String pd\_ky, **int** q, String cs\_ky) {  **this**(pd\_ky, q, cs\_ky, "false");  }   */\*  \* Used to register a new customer request  \*  \* pd\_ky Example ABC1234  \* q E \* cs\_ky xample 10 Example DE040500A  \*/*  SendCustomerRequest(String pd\_ky, **int** q, String cs\_ky, String expedited) {  **this**.pd\_ky = pd\_ky;  **this**.q = q;  **this**.cs\_ky = cs\_ky;  **this**.expedited = expedited;  created = System.currentTimeMillis();  *// create a unique ref for the request*  String k = pd\_ky + "-" + cs\_ky + "-" + issued + "-" + q;  rq\_ky = Integer.toHexString(k.hashCode());  is\_v = "true";  issued = issued + 1;   ar.add(**this**);  **if** (is\_invalid()) {  *// TODO Call in background thread to save time in calling code*  System.err.println("Customer request failed validation: " + rq\_ky);  is\_v = "false";  }  }   */\*  \* Check the request is valid against API  \*/*   **private** **boolean** **is\_invalid**() {  System.out.println("Checking request " + rq\_ky);  ApiResponse cresp = call\_url("/api/customer/check?code=" + cs\_ky);  ApiResponse presp = call\_url("/api/product/lookup?code=" + pd\_ky);  **return** cresp.statusCode != 400 & presp.statusCode != 400;  }   */\*  \* Total value of high value requests received so far  \*/*  **private** **static** **double** **totalValue**(**int** minValue, **boolean** urgentOnly) {  **double** result = 0;  **for** (SendCustomerRequest req : ar) {  ApiResponse resp = call\_url("/api/product/lookup?code=" + req.pd\_ky);  **double** price;  **if** (resp.statusCode != 500)  price = Double.parseDouble(resp.body.getElementById("price").getNodeValue());  **else**  ;  price = 100; *// the product call times out quite often so use a default*  **double** value = price \* req.q;  **if** (value >= minValue && !urgentOnly || req.expedited =="true")  result = result + value;  }  **return** result;  }   */\*\*  \* @param urlSuffix  \* @return  \*/*  **private** **static** ApiResponse **call\_url**(String urlSuffix) {  String env = System.getenv("API\_ENV");  **if** (env == **null**)  env = "LOCAL";  String user = System.getenv("API\_USR");  **if** (user == **null**)  user = "CX00001";  pass = System.getenv("API\_PAS");  **if** (pass == **null**)  pass = "secret123";  String apiBase = API\_SERVERS.get(env);  URL url;  **try** {  url = **new** URL(apiBase + urlSuffix);  HttpURLConnection con = (HttpURLConnection) url.openConnection();  con.setRequestMethod("GET");  con.setRequestProperty("Accept", "application/xml");  **int** status\_code = con.getResponseCode();  DocumentBuilderFactory f = DocumentBuilderFactory.newDefaultInstance();  Document doc = f.newDocumentBuilder().parse(con.getInputStream());  **return** **new** ApiResponse(status\_code, doc);  } **catch** (Exception e) {  *// TODO Auto-generated catch block*  e.printStackTrace();  }  **return** **new** ApiResponse(500, **null**);  }   **public** **static** **void** **main**(String args[]) {  SendCustomerRequest test\_req = **new** SendCustomerRequest("TEST\_PROD\_1", 55, "TEST\_CUST\_1", "true");  **assert** test\_req.cs\_ky == "TEST\_CUST\_1";  **assert** test\_req.pd\_ky == "TEST\_PROD\_1";  **assert** test\_req.rq\_ky == "7e48de13";  **assert** SendCustomerRequest.issued == 1;  **assert** **totalValue**(1000, **true**) == 5500;  System.out.println("TESTING PASSED - PUSH ME TO PROD !!!!!!");   } }  **class** **ApiResponse** {  **int** statusCode;  Document body;   ApiResponse(**int** statusCode, Document body) {  statusCode = statusCode;  body = body;  } } |
| --- |

##### Improvements

###### 1. Static Initialization of API\_SERVERS

* **Problem:** The API\_SERVERS map is statically initialized with an anonymous inner class, which can be simplified.
* **Improvement:** Use a static initializer block or a regular static initialization to avoid the creation of an anonymous class.

Old:

| **public** **static** Map<String, String> API\_SERVERS = **new** HashMap<>() {  {  put("LOCAL", "http://127.0.0.1");  put("DEV", "http://cust-api-dev.internal.mycorp.com");  put("PROD", "http://cust-api.internal.mycorp.com");  }  }; |
| --- |

New:

| **public** **static** **final** Map<String, String> API\_SERVERS = Map.of(  "LOCAL", "http://127.0.0.1",  "DEV", "http://cust-api-dev.internal.mycorp.com",  "PROD", "http://cust-api.internal.mycorp.com" ); |
| --- |

Bonus:

Ultimately these urls should be passed in as environment variables through a YAML file or specified on the deployment. These values may change in the future so if its hardcoded you will have to make changes in every location in the codebase and if this is used in 100s of locations it will be a nightmare to manage. So if they are defined in a centralized location and used in different deployments, you only have to make a single change.

###### 2. Thread Safety and Volatility

* **Problem:** The issued variable is marked as volatile, which ensures visibility across threads, but incrementing it (issued = issued + 1) is not an atomic operation and could lead to race conditions in a multi-threaded environment.
* **Improvement:** Use AtomicInteger for thread-safe increments.

Old:

| **static** **protected** **volatile** **int** issued = 0; |
| --- |

New:

| **static** **final** AtomicInteger issued = **new** AtomicInteger(0); |
| --- |

###### 3. LinkedList Usage

* **Problem:** A LinkedList (ar) is used to store instances of SendCustomerRequest, but no operations specific to a linked list (like frequent insertions/deletions at ends) are being performed. An ArrayList might be more appropriate.
* **Improvement:** Consider using ArrayList for faster random access.

Old:

| **static** **private** LinkedList<SendCustomerRequest> ar = **new** LinkedList<>(); |
| --- |

New:

| **static** **private** List<SendCustomerRequest> ar = **new** ArrayList<>(); |
| --- |

###### 4. Error Handling in call\_url

* **Problem:** The call\_url method has a try-catch block that catches general Exception. This can mask specific exceptions and make debugging difficult.
* **Improvement:** Catch more specific exceptions like IOException, ParserConfigurationException, and SAXException.
* **Improvement:** Consider retry logic or timeout handling for network requests.

###### 5. Inconsistent Error Handling

* **Problem:** The totalValue method silently handles a 500 status code by defaulting the price to 100, which could lead to misleading results.
* **Improvement:** Log a warning when using the default price, or handle the timeout/error more gracefully.

Old:

| **if** (resp.statusCode != 500)  price = Double.parseDouble(resp.body.getElementById("price").getNodeValue());  **else**  ;  price = 100; *// the product call times out quite often so use a default* |
| --- |

New:

| **if** (resp.statusCode == 500) {  System.err.println("Warning: Using default price due to 500 status code for product lookup.");  price = 100; } **else** {  price = Double.parseDouble(resp.body.getElementById("price").getNodeValue()); } |
| --- |

Bonus:

Setting a default price is not the best solution even if we are displaying a message for this action. The product could be a different price altogether so we will be passing incorrect information to the user.

###### 6. String Comparison

* **Problem:** String comparison using == instead of equals can lead to unexpected behavior due to reference comparison.
* **Improvement:** Use equals for string comparison.

Old:

| **if** (value >= minValue && !urgentOnly || req.expedited =="true") |
| --- |

New:

| **if** (value >= minValue && !urgentOnly || req.expedited.equals("true")) |
| --- |

###### 7. Redundant Initialization

* **Problem:** The pass variable is being set from environment variables and has a fallback value, but it’s marked as static, meaning it will retain the last value assigned across all method calls.
* **Improvement:** Consider initializing pass as a local variable within the call\_url method to avoid potential issues with shared state.

Old:

| **if** (pass == **null**)  pass = "secret123"; |
| --- |

New:

| **private** **static** ApiResponse **call\_url**(String urlSuffix) {  String pass = System.getenv("API\_PAS");  **if** (pass == **null**) pass = "secret123";  *//...* } |
| --- |

Bonus:

But ultimately this is a hardcoded secret in source code which is really bad practice and imposes security risk. This value should essentially be stored in a vault and read from it during runtime.

###### 8. Inefficient Looping

* **Problem:** The totalValue method calls call\_url repeatedly within a loop, potentially leading to multiple API calls, which is inefficient.
* **Improvement:** Batch process the requests if possible, or cache the results of API calls to avoid redundant requests.

###### 9. ApiResponse Class

* **Problem:** The ApiResponse constructor assigns values to its parameters (statusCode and body) incorrectly using statusCode = statusCode; which is ineffective.
* **Improvement:** Use this.statusCode = statusCode; to correctly assign parameter values to instance variables.

Old:

| **class** **ApiResponse** {  **int** statusCode;  Document body;   ApiResponse(**int** statusCode, Document body) {  statusCode = statusCode;  body = body;  } } |
| --- |

New:

| **class** **ApiResponse** {  **int** statusCode;  Document body;   ApiResponse(**int** statusCode, Document body) {  **this**.statusCode = statusCode;  **this**.body = body;  } } |
| --- |

Bonus:

In addition to this, we should probably give private access modifiers to prevent values from getting the default specifier and prevent code from directly changing values. We can also include more meaningful information like a message, specification of what HTTP verb was used, a timestamp on when the response was sent.

###### 10. Method Documentation

* **Improvement:** Add documentation for methods like call\_url and totalValue that explains their purpose, parameters, and return values.

###### 11. Consider Using a Logger

* **Improvement:** Replace System.out.println and System.err.println with a logging framework like SLF4J or Log4J, which provides more control over logging levels and output.

###### 12. Space and Time Complexity Analysis

* **Space Complexity:** The space complexity is dominated by the storage of the SendCustomerRequest objects in the ar list, which is O(n), where n is the number of requests.
* **Time Complexity:**
  + is\_invalid(): Each API call in call\_url is O(1) assuming the network latency is constant.
  + totalValue(): The time complexity is O(n) where n is the number of requests in the list. Each call to call\_url is O(1), making the method effectively O(n).

###### 13. Resource Management & Exception Handling

Resource Management:

**Problem:** The code opens a connection to an external resource (an HTTP URL connection) but doesn’t explicitly close it. This can lead to resource leaks, especially if the connection is left open indefinitely.

**Improvement:** Always close resources like connections, streams, or files in a finally block or, more preferably in modern Java, use the **try-with-resources** statement, which automatically closes resources.

Exception Handling:

* **Problem:** The catch block merely calls e.printStackTrace(), which prints the exception stack trace to System.err. This practice is problematic because:
  + It doesn’t provide meaningful error handling.
  + The exception details are not logged, stored, or propagated, so they’re essentially lost after being printed.
* **Improvement:** Properly log the exception using a logging framework and potentially rethrow the exception or handle it meaningfully depending on the application's needs.

Solution below is for both points

Old:

| URL url;  **try** {  url = **new** URL(apiBase + urlSuffix);  HttpURLConnection con = (HttpURLConnection) url.openConnection();  con.setRequestMethod("GET");  con.setRequestProperty("Accept", "application/xml");  **int** status\_code = con.getResponseCode();  DocumentBuilderFactory f = DocumentBuilderFactory.newDefaultInstance();  Document doc = f.newDocumentBuilder().parse(con.getInputStream());  **return** **new** ApiResponse(status\_code, doc);  } **catch** (Exception e) {  *// TODO Auto-generated catch block*  e.printStackTrace();  }  **return** **new** ApiResponse(500, **null**); |
| --- |

New:

| HttpURLConnection con = **null**;  **try** {  URL url = **new** URL(apiBase + urlSuffix);  con = (HttpURLConnection) url.openConnection();  con.setRequestMethod("GET");  con.setRequestProperty("Accept", "application/xml");    **int** statusCode = con.getResponseCode();  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  Document doc = factory.newDocumentBuilder().parse(con.getInputStream());   **return** **new** ApiResponse(statusCode, doc);  } **catch** (Exception e) {  *// Proper logging*  logger.severe("Failed to call URL: " + urlSuffix + " - Exception: " + e.getMessage());  **return** **new** ApiResponse(500, **null**);  } **finally** {  **if** (con != **null**) {  **try** {  con.disconnect();  } **catch** (Exception e) {  logger.warning("Failed to close the connection: " + e.getMessage());  }  }  } |
| --- |

###### 14. Other

* Variable names can be more meaning, we have a variable called *q* which is being passed in as a constructor parameter when creating SendCustomerRequest, it’s hard to tell what it’s use case is for
* We are mixing method naming conventions. For example the method is\_invalid using snake casing and we have another method called totalValue which uses camelcase, we should stick to one naming convention to be consistent.
* In our main method we are writing unit tests in production level code, this is not a good practice. Assertions are typically used as a debugging tool to verify assumptions made by the program. They are not meant for production code but rather to catch bugs during development.

#### Algorithm Question

The second part of the coding interview consists of a medium level leetcode problem.

##### Count Substrings Having Distinct Characters

Given a string, how many different substrings exist in it that have no repeating characters? Two substrings are considered different if they have a different start or end index.

**Example**

S = “abac”

The substrings that have no repeating characters in them are “a”, “b”, “a”, “c”, “ab”, “ba”, “ac”, and “bac”. Note that “aba” and “abac” do not qualify because the character “a” is repeated in them. Also note that two substrings “a” and “a”, both qualify because their start indices are different: s[0] and s[2]. There are 8 substrings that have no repeating characters.

Answer

| **public** **static** **int** **findSubstrings**(String s) {  **if** (s.isEmpty()) **return** 0;  **int** sizeOfString = s.length();   **int** result = 0;  **int**[] charFrequency = **new** **int**[26];   **int** left = 0, right = 0;   **while** (left < sizeOfString) {  *// Check all characters in substring from left to right are distinct*  **if** (right < sizeOfString && (charFrequency[s.charAt(right) - 'a'] == 0)) {  *// Increment count of right character*  charFrequency[s.charAt(right) - 'a']++;  *// Add every substring ending with right pointer and starting with any index between left/right to result*  result += (right - left) + 1;  right++;  } **else** {  *// Move right pointer to the left*  charFrequency[s.charAt(left) - 'a']--;  left++;  }  }   **return** result;  } } |
| --- |

### Systems Design

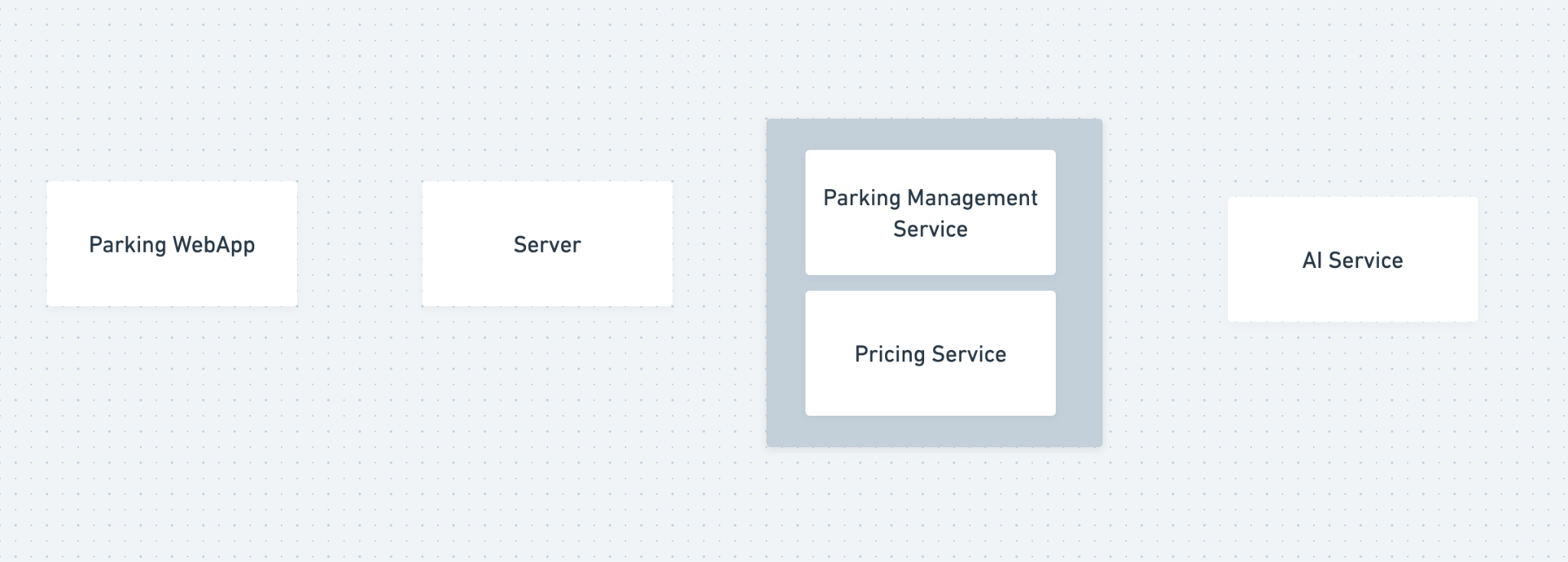
#### AI Parking System

The owner of a parking lot has come up with a system to change the face of parking using AI. They have written an AI service that automatically captures the vehicle number along with the timestamp when a car enters the lot.

There is one attendant that sits at the exit, when the vehicle is exiting the attendant should be able to get the price to charge the vehicle through the WebApp.

Design a scalable system design for a web app that uses this AI service to automate the whole parking experience flawlessly.

For this problem we are given a default system design which we need to connect the arrows and explain the flow of what happens when a customer is exiting. You will also be required to select a major cloud provider for this system design: AWS, GCP, Azure (pick AWS)



Questions to ask initially:

* Is there any customer facing portion in the application or this is only to be used by the parking attendant?
* Can we assume there is only one garage or can this scale across multiple garages in various locations/regions in the world?
* Are we storing the images captured by the AI service?

Some other questions which the interviewer will ask

1. What services would you use to deploy this application?
   1. We can use ECS Fargate in order to deploy these backend services for Parking Management and Pricing. We do not have to worry about underlying infrastructure and hardware management and fargate will choose the best option for us. We can also set auto scaling policies to scale up and down to handle traffic spikes
   2. For AI service we can use AWS Lambda. Since we are only capturing the image of the vehicle coming in, we can trigger the lambda service whenever we have an event where we detect a vehicle entering. The lambda function can perform processing on the image and extract any relevant data and pass that to the Parking Management Service and it can store the user’s number plate and timestamp into the database and lambda can store the image itself into AWS S3.
2. If we want to deploy this application across regions in the US (or internationally) how would that change your application structure?
   1. If we are deploying in multi regions we can use CI/CD like Jenkins and add deployment stages to deploy to the supported regions. We can configure Route 53 to use Global DNS Routing and we can use either latency based routing policy or geo-location routing policy in order to route traffic to the correct regions
   2. For the database we have 2 options, we can use regional databases or we can use a global database which every region will connect to. There is a tradeoff, with regional databases, data is stored only in the specific region, if one region ever goes down or the request happens to go to the opposite region for whatever reason, the database will not contain the data since it is a different region. This can influence the database choice, I would recommend using DynamoDB with Global tables, DynamoDB Global Tables automatically replicate data across regions, providing low-latency access and high availability.

Final Diagram Design

