**Pocketmath - Java Software Developer Questions**

Here is a set of questions that we would like to have you complete, there are a total of 4 questions. We are looking for simplicity of the solution, time-space complexity, code style, functional correctness. Please be sure to explain the time-space complexity for the questions where appropriate. You can leave comments, we will read them!

For all of these questions, you can use any programming language(s), as long as they are part of the job description.

Estimated total time: 2 to 3 hours.

**1.**

Take an array of integers and partition it so that all the even integers in the array precede all the odd integers in the array. Your solution must take linear time in the size of the array and operate in-place with only a constant amount of extra space.

 sample input  : 2,4,7,6,1,3,5,4

 sample output : 2,4,6,4,7,1,3,5

Solution:-

Kl;

**public class** EvenOddSegregation {  
 **public static void** main(String args[]) {  
 **int** array[] = **new int**[]{2, 4, 7, 6, 1, 3, 5, 4};  
 **boolean** loopRequired;  
 **do** {  
 loopRequired = **false**;  
 **for** (**int** i = 0; i < array.**length** - 1; i++) {  
  
 **if** (array[i] % 2 != 0 && array[i + 1] % 2 == 0) {  
  
 **int** temp = array[i];  
 array[i] = array[i + 1];  
 array[i + 1] = temp;  
 loopRequired = **true**;  
 }  
 }  
 } **while** (loopRequired);  
 **for** (**int** k = 0; k < array.**length**; k++) {  
 System.***out***.println(array[k]);  
 }  
 }  
}

**2.**

Suppose we have the following code:

 class Foo {

   Foo() {...}

   void first() {...}

   void second() {...}

   void third() {...}

 }

The same instance of Foo will be passed to three different threads. Thread A will call first(), Thread B will call second(), and Thread C will call third(). Design a mechanism to ensure that first() is called before second() and second() is called before third().

**Solution:-**

One Soltuion:-

We can use SemaPhore to achieve the sequence of the Threads. This can easily be replaced with the wait() and the notifyAll() mechanism in java..

For example:-

Semaphore s\_a(0);

Semaphore s\_a(0);

A{

S\_a.release(1);

}

B {

S\_a.acquire(1);

S\_b.release(1);

}

C{

S\_b.acquire(1);

}

**3.**

Implement a MyQueue class which implements a queue using two stacks.

Solution:-

**package** com.test;  
  
**import** java.util.Stack;  
  
*/\*\*  
 \* <p>  
 \* Created with IntelliJ IDEA. <br/>  
 \* User: Ankur jain <br/>  
 \* Date: 25-Dec-16 <br/>  
 \* Time: 09:35 AM <br/>  
 \*/***public class** MyQueue<E> {  
 Stack **s1**;  
 Stack **s2**;  
  
 **public** MyQueue() {  
 **s1** = **new** Stack();  
 **s2** = **new** Stack();  
 }  
  
 **public boolean** isEmpty() {  
 **if** (**s2**.isEmpty()) {  
 **while** (!**s1**.isEmpty()) {  
 **s2**.push(**s1**.pop());  
 }  
 }  
 **return s2**.isEmpty();  
 }  
  
 **public void** enQueue(Object data) {  
 **s1**.push(data);  
 }  
  
 **public** Object deQueue() {  
 **if** (!**s2**.isEmpty()) {  
 **return s2**.pop();  
 } **else** {  
 **while** (!**s1**.isEmpty()) {  
 **s2**.push(**s1**.pop());  
 }  
 **return s2**.pop();  
 }  
 }  
}

**4.**

To discover what the question is, you will need to call our REST endpoint through the GET method. You must call with a custom header named **x-api-key**, along with the value of the API key for authorization.

URL of endpoint: https://fvjkpkflnc.execute-api.us-east-1.amazonaws.com

GET /prod/question

The API key is: gaqcRZE4bd58gSAJH3XsLYBo1EvwIQo88IfYL1L5

Solution:-

We are using the Java 8 Streams API for filtering over the collection as well as two POJo classes Transaction and Trader to fetch the Response from the API as a placeholder. Jackson-databind to convert the json response to the respective java objects.

Java Application:-

TraderTransactionApplication.java

**package** com.pocketmath.test;  
  
  
**import** com.fasterxml.jackson.core.type.TypeReference;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.net.HttpURLConnection;  
**import** java.net.MalformedURLException;  
**import** java.net.URL;  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
  
*/\*\*  
 \* <p>  
 \* Created with IntelliJ IDEA. <br/>  
 \* User: Ankur jain <br/>  
 \* Date: 25-Dec-16 <br/>  
 \* Time: 10:15 AM <br/>  
 \*/***public class** TraderTransactionApplication {  
 **public static void** main(String args[]) {  
 **try** {  
 URL url = **new** URL(**"https://fvjkpkflnc.execute-api.us-east-1.amazonaws.com/prod/question"**);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod(**"GET"**);  
 connection.setRequestProperty(**"Accept"**, **"application/json"**);  
 connection.setRequestProperty(**"x-api-key"**, **"gaqcRZE4bd58gSAJH3XsLYBo1EvwIQo88IfYL1L5"**);  
 **if** (connection.getResponseCode() != 200) {  
 **throw new** RuntimeException(**"Failed: Http Error Code:"** + connection.getResponseCode());  
 }  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));  
 String output;  
 System.***out***.println(**"The Question to be solved is: "**);  
 **while** ((output = br.readLine()) != **null**) {  
 System.***out***.println(output);  
 }  
 List<Trader> traders = *getAllTraders*();  
 List<Transaction> transactions = *getAllTransactions*();  
 *tradersFromSingaporeSortedByName*(traders);  
 *transactionWithTheMaxValue*(transactions);  
 *transactionsInYear2016SortedByName*(transactions);  
 *averageOfTransactionsFromTradersLivingInBeijing*(traders, transactions);  
 connection.disconnect();  
 } **catch** (MalformedURLException exception) {  
 exception.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private static void** tradersFromSingaporeSortedByName(List<Trader> traders) {  
 List<Trader> tradersAfterFilteration = traders.stream().filter(p -> p.getCity().equals(**"Singapore"**)).collect(Collectors.*toList*());  
 Collections.*sort*(tradersAfterFilteration, (p1, p2) -> p1.getName().compareTo(p2.getName()));  
 System.***out***.println(**"Traders From Singapore Sorted By name"**);  
 **for** (Trader trader : tradersAfterFilteration) {  
 System.***out***.println(trader.getCity() + **" "** + trader.getId() + **" "** + trader.getName());  
 }  
 }  
  
 **private static void** transactionWithTheMaxValue(List<Transaction> transactions) {  
 List<Double> transactionValues = **new** ArrayList<Double>();  
 **for** (Transaction t : transactions) {  
 transactionValues.add(Double.*valueOf*(t.getValue()));  
 }  
 DoubleSummaryStatistics stats = transactionValues.stream().mapToDouble((x) -> x).summaryStatistics();  
 System.***out***.println(**"Transaction with the Highest Value is :"** + stats.getMax());  
 List<Transaction> transactionWithMaxValue = transactions.stream().filter(t -> Double.*valueOf*(t.getValue()) == stats.getMax()).collect(Collectors.*toList*());  
 **for** (Transaction t : transactionWithMaxValue) {  
 System.***out***.println(t.getTimestamp() + **" "** + t.getTraderId() + **" "** + t.getValue());  
 }  
 }  
  
 **private static void** transactionsInYear2016SortedByName(List<Transaction> transactions) {  
 **for** (Transaction t : transactions) {  
 Calendar cal = Calendar.*getInstance*();  
 cal.setTimeInMillis(Long.*valueOf*(t.getTimestamp()) \* 1000);  
 **int** year = cal.get(Calendar.***YEAR***);  
 t.setYear(year);  
 }  
 List<Transaction> transactionFilterOnYear = transactions.stream().filter(item -> item.getYear() == 2016).collect(Collectors.*toList*());  
 Collections.*sort*(transactionFilterOnYear, (t1, t2) -> Double.*valueOf*(t2.getValue()).compareTo(Double.*valueOf*(t1.getValue())));  
 System.***out***.println(**"Transaction in the year 2016 sorted by value are: "**);  
 **for** (Transaction t : transactionFilterOnYear) {  
 System.***out***.println(t.getTimestamp() + **" "** + t.getYear() + **" "** + t.getValue() + **" "** + t.getTraderId());  
 }  
 }  
  
 **private static void** averageOfTransactionsFromTradersLivingInBeijing(List<Trader> traders, List<Transaction> transactions) {  
 List<Trader> tradersInBeijing = traders.stream().filter(trader -> trader.getCity().equals(**"Beijing"**)).collect(Collectors.*toList*());  
 **double** sum = 0d, count = 0;  
 **for** (Transaction t : transactions) {  
 **for** (Trader trader : tradersInBeijing) {  
 **if** (trader.getId().equals(t.getTraderId())) {  
 sum = sum + Double.*valueOf*(t.getValue());  
 count++;  
 }  
 }  
 }  
 System.***out***.println(**"Average Of All Transactions whose traders are in beijing are:"** + sum / count);  
 }  
  
 **private static** List<Trader> getAllTraders() {  
 List<Trader> traders = **new** ArrayList<Trader>();  
 **try** {  
 URL url = **new** URL(**"https://fvjkpkflnc.execute-api.us-east-1.amazonaws.com/prod/traders"**);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod(**"GET"**);  
 StringBuilder str = **new** StringBuilder();  
 connection.setRequestProperty(**"Accept"**, **"application/json"**);  
 connection.setRequestProperty(**"x-api-key"**, **"gaqcRZE4bd58gSAJH3XsLYBo1EvwIQo88IfYL1L5"**);  
 **if** (connection.getResponseCode() != 200) {  
 **throw new** RuntimeException(**"Failed: Http Error Code:"** + connection.getResponseCode());  
 }  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));  
 String output;  
 **while** ((output = br.readLine()) != **null**) {  
 str.append(output);  
 }  
 ObjectMapper mapper = **new** ObjectMapper();  
 traders = mapper.readValue(str.toString(), **new** TypeReference<List<Trader>>() {  
 });  
 connection.disconnect();  
 } **catch** (MalformedURLException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** traders;  
 }  
  
 **private static** List<Transaction> getAllTransactions() {  
 List<Transaction> transactions = **new** ArrayList<Transaction>();  
 **try** {  
 URL url = **new** URL(**"https://fvjkpkflnc.execute-api.us-east-1.amazonaws.com/prod/transactions"**);  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod(**"GET"**);  
 connection.setRequestProperty(**"Accept"**, **"application/json"**);  
 connection.setRequestProperty(**"x-api-key"**, **"gaqcRZE4bd58gSAJH3XsLYBo1EvwIQo88IfYL1L5"**);  
 **if** (connection.getResponseCode() != 200) {  
 **throw new** RuntimeException(**"Failed: Http Error Code:"** + connection.getResponseCode());  
 }  
 BufferedReader br = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));  
 String output;  
 StringBuilder response = **new** StringBuilder();  
 **while** ((output = br.readLine()) != **null**) {  
 response.append(output);  
 }  
 ObjectMapper mapper = **new** ObjectMapper();  
 transactions = mapper.readValue(response.toString(), **new** TypeReference<List<Transaction>>() {  
 });  
 connection.disconnect();  
 } **catch** (MalformedURLException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** transactions;  
 }  
}

Trader.java

**package** com.pocketmath.test;  
  
*/\*\*  
 \* <p>  
 \* Created with IntelliJ IDEA. <br/>  
 \* User: Ankur jain<br/>  
 \* Date: 25-Dec-16 <br/>  
 \* Time: 10:47 AM <br/>  
 \*/***public class** Trader {  
 **private** String **city**;  
 **private** String **name**;  
 **private** String **id**;  
  
 **public** String getId() {  
 **return this**.**id**;  
 }  
  
 **public void** setId(String id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getCity() {  
 **return this**.**city**;  
 }  
  
 **public void** setCity(String city) {  
 **this**.**city** = city;  
 }  
  
 **public** String getName() {  
 **return this**.**name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
  
}

Transaction.java

**package** com.pocketmath.test;  
  
*/\*\*  
 \* <p>  
 \* Created with IntelliJ IDEA. <br/>  
 \* User: Ankur jain <br/>  
 \* Date: 25-Dec-16 <br/>  
 \* Time: 11:10 AM <br/>  
 \*/***public class** Transaction {  
 String **timestamp**;  
 String **traderId**;  
 String **value**;  
 **int year**;  
  
  
 **public int** getYear() {  
 **return this**.**year**;  
 }  
  
 **public void** setYear(**int** year) {  
 **this**.**year** = year;  
 }  
  
  
 **public** String getTimestamp() {  
 **return this**.**timestamp**;  
 }  
  
 **public void** setTimestamp(String timestamp) {  
 **this**.**timestamp** = timestamp;  
 }  
  
 **public** String getTraderId() {  
 **return this**.**traderId**;  
 }  
  
 **public void** setTraderId(String traderId) {  
 **this**.**traderId** = traderId;  
 }  
  
 **public** String getValue() {  
 **return this**.**value**;  
 }  
  
 **public void** setValue(String value) {  
 **this**.**value** = value;  
 }  
}

pom.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.pocketmath.test</**groupId**>  
 <**artifactId**>com.pocketmath.test</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
  
 <**dependencies**>  
 *<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->* <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.8.5</**version**>  
 </**dependency**>  
  
 </**dependencies**>  
</**project**>

ReadMe.md

**Language/Framework  
==================  
  
-** Java  
  
**Problem statement  
=================**  
  
question - Hello potential Pocketeer**!** At PocketMath we look for software engineers who can build simple and elegant solutions, so we've designed these challenges to see what you can come up with. We'd like for you to build an application using one of Java/Golang/Rust/Scala/Clojure with any framework(s) and/or toolchain(s) of your choice. The application will need to invoke our REST endpoints for data and answer the following queries. Every run of the application should invoke the endpoints for data, i.e. don't persist the data for repeated runs, the answers for each challenge can simply be printed on the console or written to a file. Please provide your source code and a brief README.md on how to run your application to our recruiter. We are not interested in the actual answer to these queries, we're much more interested in how you've solved them**!** These challenges are all of the same domain: traders executing transactions. Traders have a name, a city, and a unique trader ID. Transactions have a UTC 0 UNIX epoch millisecond timestamp, a value (representing the monetary value transacted in USD), and a trader ID (identifying the trader who made the transaction). There are REST endpoints which you can invoke GET to obtain the data. The endpoints require the same API key for authorization. The REST endpoints are 'GET /prod/traders' (for all traders) and 'GET /prod/transactions' (for all transactions).  
  
challenges -   
1: Find all traders from Singapore and sort them by name.  
  
2: Find the transaction with the highest value.  
  
3: "Find all transactions in the year 2016 and sort them by value (high to small)."  
  
4: "Find the average of transactions' values from the traders living in Beijing."

Response:-

The Question to be solved is:

{

"question": "Hello potential Pocketeer! At PocketMath we look for software engineers who can build simple and elegant solutions, so we've designed these challenges to see what you can come up with. We'd like for you to build an application using one of Java/Golang/Rust/Scala/Clojure with any framework(s) and/or toolchain(s) of your choice. The application will need to invoke our REST endpoints for data and answer the following queries. Every run of the application should invoke the endpoints for data, i.e. don't persist the data for repeated runs, the answers for each challenge can simply be printed on the console or written to a file. Please provide your source code and a brief README.md on how to run your application to our recruiter. We are not interested in the actual answer to these queries, we're much more interested in how you've solved them! These challenges are all of the same domain: traders executing transactions. Traders have a name, a city, and a unique trader ID. Transactions have a UTC 0 UNIX epoch second timestamp, a value (representing the monetary value transacted in USD), and a trader ID (identifying the trader who made the transaction). There are REST endpoints which you can invoke GET to obtain the data. The endpoints require the same API key for authorization. The REST endpoints are 'GET /prod/traders' (for all traders) and 'GET /prod/transactions' (for all transactions).",

"challenges": [

"Find all traders from Singapore and sort them by name.",

"Find the transaction with the highest value.",

"Find all transactions in the year 2016 and sort them by value (high to small).",

"Find the average of transactions' values from the traders living in Beijing."

]

}

Traders From Singapore Sorted By name

Singapore 75f87fa90004d185432caa1dfb208bbf Brittany

Singapore d6460d863cc7403c4d48eb8682d87784 Donald

Singapore 40687c8206d15373954d8b27c6724f62 Jack

Singapore 195fbb57ffe7449796d23466085ce6d8 May

Singapore dc2054afd537ddc98afd9347136494ac Tim

Transaction with the Highest Value is :0.9817253648270564

1369920350 f9dc77cece7fa16f6edd2d1d64853e4b 0.9817253648270564

Transaction in the year 2016 sorted by value are:

1464390091 0.9510087764973526 e2ef48860012614fc88d1b20a72e71e3

1469552504 0.9057669559853739 ba0e0cde1bf72c28d435c89a66afc61a

1457086632 0.861510525858726 40687c8206d15373954d8b27c6724f62

1469278218 0.8400671857631128 75f87fa90004d185432caa1dfb208bbf

1468240825 0.8159875545712488 2b80f09163f60ce1774b438e605eb1f9

1457310093 0.7722524970561755 f9dc77cece7fa16f6edd2d1d64853e4b

1459899942 0.6285759964411405 dc2054afd537ddc98afd9347136494ac

1465665137 0.5879074361309028 d6460d863cc7403c4d48eb8682d87784

1460627664 0.5671923672878423 578ad8e10dc4edb52ff2bd4ec9bc93a3

1451646635 0.4706412755534929 4be90ed9c41356ec34247e49aec714a9

1464832873 0.3991428735818352 9406e3c325bfc9873426e5eda4ba6e18

1468703436 0.34941675325735955 40687c8206d15373954d8b27c6724f62

1467233082 0.343196528196014 195fbb57ffe7449796d23466085ce6d8

1462743254 0.2267158447915525 9406e3c325bfc9873426e5eda4ba6e18

1469124037 0.2241815198927285 12ace8436c64ceb907536640b58788f0

1468042826 0.1603709149402308 578ad8e10dc4edb52ff2bd4ec9bc93a3

1453178033 0.14698269030024114 dc2054afd537ddc98afd9347136494ac

1464575692 0.07647547998772064 17b847bbabf88e3e4512bd3cda49e5cc

1452576168 0.06359176383227838 75f87fa90004d185432caa1dfb208bbf

1462451251 0.04844135558348883 e2ef48860012614fc88d1b20a72e71e3

1455464497 0.024371020734830884 75f87fa90004d185432caa1dfb208bbf

1463385006 0.01324683368496038 195fbb57ffe7449796d23466085ce6d8

Average Of All Transactions whose traders are in beijing are:0.40219304265178935