**Java Exceptions: Data Transmission**

Coding Java Exception Handling Language Proficiency Medium Problem Solving

Question description

The task at hand involves the implementation of a secure communication channel for sending and

receiving encrypted messages, with a prerequisite of an encryption token for any user intending to

transmit messages. There exist two types of transmission operations:

Sending a new encrypted message.

Decrypting and reading an incoming message.

Before any transmission operation can take place, it must successfully pass through validation. Failure to

meet validation parameters results in error codes along with their corresponding messages.

USER\_NOT\_AUTHORIZED: Indicates an invalid operation, in the scenario where no encryption token is

provided for the transaction (either empty or null). The error message will read “User not authorized”.

INVALID\_MESSAGE\_LENGTH: This error code points to an attempt to send or read a zero or negativelength

message. The error message would read "Message length must be greater than zero".

INSUFFICIENT\_ENCRYPTION\_KEY: This error code flags an attempt to decrypt a longer message than

permitted by the available encryption keys (which may change when sending or reading messages).

The respective error message would read "Insufficient encryption key".

In this task, you are to establish the workflow of secure messaging by implementing the following three

classes in their entirety:

1. SecurityException class features:

Constructor: SecurityException(String errorStatus, String errorCode).

Method: String getErrorCode() to fetch the error code in case of an exception trigger.

2. SecureChannel class features:

Constructor: SecureChannel(String channelId, String userId).

Constructor: SecureChannel(String channelId, String userId, String userEncryptionToken).

Methods: String getChannelId() to fetch the channel ID, String getUserId() to fetch the user ID,

String getUserEncryptionToken() to fetch the encryption token, int

getChannelEncryptionKeyLength() to fetch the encryption key length, void

setChannelEncryptionKeyLength(int channelEncryption) to update the encryption key length.

Note: The channelEncryptionKeyLength instance variable should have an initial value of 0 and

will change with send and read operations.

3. SecureTransmission class features:

Methods: void sendMessage(SecureChannel secureChannel, int messageLength) to send a

message over a secure channel, this method throws "User not authorized" and "Message length

must be greater than zero" exception in case of an error, and if not the channel encryption key

length increases by messageLength.

void readMessage(SecureChannel secureChannel, int messageLength) to read a received

message from a secure channel. Both these methods should be capable of raising requisite

exceptions for any invalid transmission operations, and if no errors occur the channel encryption

key length decreases by messageLength.

The editor’s locked stub code first sets up and validates authorized and unauthorized secure channels,

following which a series of transmission operations are performed. These operations are:

channelId send messageLength: This transmission type is managed via the method:

SecureTransmission.sendMessage(secureChannel, messageLength).

channelId read messageLength: This transmission type is managed via the method:

SecureTransmission.readMessage(secureChannel, messageLength)

Upon completing the series of operations, the locked stub code prints out the channel ID, user ID, and

the available encryption key length for each secure channel.

Constraints

1 ≤ numberOfChannels ≤ 100

1 ≤ numberOfTransmissions ≤ 2 × 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains the value of numberOfChannels describing the total number of channels.

The next numberOfChannels lines contain the required information of the channel.

The next line contains the value of numberOfTransmissions describing the total number of

transmissions.

The next numberOfTransmissions lines contains one of the above-mentioned two transmissions.

SAMPLE CASE 0

Sample Input For Custom Testing

3

21

Julia bff834a2c11ceb782f98e428686ca3c4ea

2 Samantha

10

1 read 20

1 send 10

1 send 0

1 read 30

2 send 500

1 send -5

1 send 100

1 read -20

1 read 10

1 send 720

Sample Output

INSUFFICIENT\_ENCRYPTION\_KEY: Insufficient encryption key.

Message successfully sent.

INVALID\_MESSAGE\_LENGTH: Message length must be greater than zero.

INSUFFICIENT\_ENCRYPTION\_KEY: Insufficient encryption key.

USER\_NOT\_AUTHORIZED: User not authorized.

INVALID\_MESSAGE\_LENGTH: Message length must be greater than zero.

Message successfully sent.

INVALID\_MESSAGE\_LENGTH: Message length must be greater than zero.

Message successfully read.

Message successfully sent.

1 Julia 820

2 Samantha 0

Explanation

1 read 20: Fails because channelEncryptionKeyLength is 0, less than 20. Throws

INSUFFICIENT\_ENCRYPTION\_KEY .

1 send 10: Successfully sends the message. channelEncryptionKeyLength becomes 10.

1 send 0: Fails because the message length is 0. Throws INVALID\_MESSAGE\_LENGTH .

1 read 30: Fails because channelEncryptionKeyLength is 10, less than 30. Throws

INSUFFICIENT\_ENCRYPTION\_KEY .

2 send 500: Fails because Samantha has no encryption token. Throws USER\_NOT\_AUTHORIZED .

1 send -5: Fails because the message length is negative. Throws INVALID\_MESSAGE\_LENGTH .

1 send 100: Successfully sends the message. channelEncryptionKeyLength becomes 110.

1 read -20: Fails because the message length is negative. Throws INVALID\_MESSAGE\_LENGTH .

1 read 10: Successfully reads the message. channelEncryptionKeyLength becomes 100.

1 send 720: Successfully sends the message. channelEncryptionKeyLength becomes 820.

Interviewer guidelines

SOLUTION

: Java 17

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Scanner;

**/\***

**\* Implement sendMessage and readMessage methods of SecureTransmission class.**

**\*/**

**class SecurityException extends Exception {**

**private String message;**

**private String errorCode;**

**static final long serialVersionUID = 1L;**

**public SecurityException(String message, String errorCode) {**

**super();**

**this.message = message;**

**this.errorCode = errorCode;**

**}**

**public String getErrorCode() {**

**return errorCode;**

**}**

**public String getMessage() {**

**return message;**

**}**

**}**

**class SecureChannel {**

**private String channelId;**

**private String userId;**

**private String userEncryptionToken;**

**private int channelEncryptionKeyLength;**

**public SecureChannel(String channelId, String userId) {**

**this.channelId = channelId;**

**this.userId = userId;**

**}**

**public SecureChannel(String channelId, String userId, String userEncryptionToken) {**

**this.channelId = channelId;**

**this.userId = userId;**

**this.userEncryptionToken = userEncryptionToken;**

**}**

**public String getChannelId() {**

**return channelId;**

**}**

**public String getUserId() {**

**return userId;**

**}**

**public String getUserEncryptionToken() {**

**return userEncryptionToken;**

**}**

**public int getChannelEncryptionKeyLength() {**

**return channelEncryptionKeyLength;**

**}**

**public void setChannelEncryptionKeyLength(int channelEncryptionKeyLength) {**

**this.channelEncryptionKeyLength = channelEncryptionKeyLength;**

**}**

**}**

**class SecureTransmission {**

**public static final String USER\_NOT\_AUTHORIZED = "USER\_NOT\_AUTHORIZED";**

**public static final String INVALID\_MESSAGE\_LENGTH = "INVALID\_MESSAGE\_LENGTH";**

**public static final String INSUFFICIENT\_ENCRYPTION\_KEY = "INSUFFICIENT\_ENCRYPTION\_KEY";**

**public void sendMessage(SecureChannel secureChannel, int messageLength) throws SecurityException {**

**if (secureChannel.getUserEncryptionToken() == null || secureChannel.getUserEncryptionToken().isEmpty()) {**

**throw new SecurityException("User not authorized", USER\_NOT\_AUTHORIZED);**

**}**

**if (messageLength <= 0) {**

**throw new SecurityException("Message length must be greater than zero", INVALID\_MESSAGE\_LENGTH);**

**}**

**secureChannel.setChannelEncryptionKeyLength(secureChannel.getChannelEncryptionKeyLength() + messageLength);**

**}**

**public void readMessage(SecureChannel secureChannel, int messageLength) throws SecurityException {**

**if (secureChannel.getUserEncryptionToken() == null || secureChannel.getUserEncryptionToken().isEmpty()) {**

**throw new SecurityException("User not authorized", USER\_NOT\_AUTHORIZED);**

**}**

**if (messageLength <= 0) {**

**throw new SecurityException("Message length must be greater than zero", INVALID\_MESSAGE\_LENGTH);**

**}**

**if (secureChannel.getChannelEncryptionKeyLength() < messageLength) {**

**throw new SecurityException("Insufficient encryption key", INSUFFICIENT\_ENCRYPTION\_KEY);**

**}**

**secureChannel.setChannelEncryptionKeyLength(secureChannel.getChannelEncryptionKeyLength() - messageLength);**

**}**

**}**

public class Solution {

private static final Scanner INPUT\_READER = new Scanner(System.in);

private static final SecureTransmission SECURE\_TRANSMISSION = new SecureTransmission();

private static final Map<String, SecureChannel> SECURE\_CHANNELS = new HashMap<>();

public static void main(String[] args) {

int numberOfChannels = Integer.parseInt(INPUT\_READER.nextLine());

while (numberOfChannels-- > 0) {

String[] channel = INPUT\_READER.nextLine().split(" ");

SecureChannel secureChannel;

if (channel.length == 2) {

secureChannel = new SecureChannel(channel[0], channel[1]);

} else {

secureChannel = new SecureChannel(channel[0], channel[1], channel[2]);

}

SECURE\_CHANNELS.put(channel[0], secureChannel);

}

int numberOfTransmissions = Integer.parseInt(INPUT\_READER.nextLine());

while (numberOfTransmissions-- > 0) {

String[] transmission = INPUT\_READER.nextLine().split(" ");

SecureChannel secureChannel = SECURE\_CHANNELS.get(transmission[0]);

if (transmission[1].equals("send")) {

try {

SECURE\_TRANSMISSION.sendMessage(secureChannel, Integer.parseInt(transmission[2]));

System.out.println("Message successfully sent.");

} catch (SecurityException ex) {

System.out.println(ex.getErrorCode() + ": " + ex.getMessage() + ".");

}

} else {

try {

SECURE\_TRANSMISSION.readMessage(secureChannel, Integer.parseInt(transmission[2]));

System.out.println("Message successfully read.");

} catch (SecurityException ex) {

System.out.println(ex.getErrorCode() + ": " + ex.getMessage() + ".");

}

}

}

System.out.println();

List<String> channelIds = new ArrayList<>();

channelIds.addAll(SECURE\_CHANNELS.keySet());

Collections.sort(channelIds);

for (String channelId : channelIds) {

SecureChannel secureChannel = SECURE\_CHANNELS.get(channelId);

System.out.println(secureChannel.getChannelId() + " " + secureChannel.getUserId() + " " + secureChannel.getChannelEncryptionKeyLength());

}

}

}

**Problem Solving: Trading Platform**

Coding Easy Hashing Arrays Strings

Question description

A quantitative trading firm seeks to create a tool for querying the net profit/loss of the firm at any given

time. The tool processes a list of events, each of which can be classified into one of four categories:

1. BUY stock quantity: Signifies the purchase of <quantity> shares of stock <stock> at the market price.

2. SELL stock quantity: Indicates the sale of <quantity> shares of stock <stock> at the market price.

3. CHANGE stock price: Signifies a change in the market price of <stock> by <price> amount, which can

be either positive or negative.

4. QUERY: Represents a query for the net profit/loss from the start of trading to the present.

The tool should return a list of numbers corresponding to each QUERY event.

Example

For instance, given the list of events [“BUY googl 20”, “BUY aapl 50”, “CHANGE googl 6”, “QUERY”, “SELL

aapl 10”, “CHANGE aapl -2”, “QUERY”].

Events Portfolio Profit so far

BUY googl 20 googl 20 0

BUY aapl 50 googl 20

aapl 50

0

CHANGE googl 6 googl 20

aapl 50

120

QUERY 120

SELL aapl 10 googl 20

aapl 40

120

CHANGE aapl -2 googl 20

aapl 40

40

QUERY googl 20

aapl 40

40

Hence the answer should be [120, 40].

Function Description

Complete the function getNetProfit in the editor below.

getNetProfit has the following parameter:

string events[n]: the events to process

Returns

int[]: the answers to the "QUERY" events

Constraints

1 ≤ n ≤ 10

| events[i] | ≤ 21

For query, SELL <stock> <quantity>, it is guaranteed that there are enough shares owned.

1 ≤ quantity < 10

The absolute value of a change in the price of any stock at any event will not exceed 10 .

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of elements in events.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains a string that represents events[i].

SAMPLE CASE 0

Sample Input For Custom Testing

5

3

3

STDIN FUNCTION

----- --------

2 → n = 5

BUY hackr 2 → events = [“BUY hackr 2", “QUERY”]

QUERY

Sample Output

0

Explanation

The firm purchased 2 stocks of hackr, then have a "QUERY" event. Since there is no change in stock

price, there is 0 profit so far.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

6 → n = 5

BUY stock2 2 → events = [“BUY stock2 2", “BUY stock1 4”, "CHANGE stock2 -8", "SELL stock1 2",

"BUY stock3 3", "QUERY"]

BUY stock1 4

CHANGE stock2 -8

SELL stock1 2

BUY stock3 3

QUERY

Sample Output

-16

Explanation

The price of 2 shares of stock2 decreased by 8.

Interviewer guidelines

SOLUTION

Skills: e.g., Loops and counters, I/O, dynamic programming

Optimal Solution:

Iterate through the events and maintain a map of stock quantity. In the event of buy and sell, add and

subtract the respective quantities and in the event of change, update the net profit earned based on

the quantity of stock present in the map.

def getNetProfit(events):

stock\_quantity = {} # dictionary to keep track of the quantity of each stock held

stock\_price = {} # dictionary to keep track of the price of each stock

net\_profit = 0 # variable to keep track of the net profit

result = [] # list to store the results of each query

for event in events:

event\_list = event.split()

event\_type = event\_list[0]

stock\_name = event\_list[1] if len(event\_list) > 1 else None

if event\_type == "BUY":

quantity = int(event\_list[2])

stock\_quantity[stock\_name] = stock\_quantity.get(stock\_name, 0) + quantity

elif event\_type == "SELL":

quantity = int(event\_list[2])

stock\_quantity[stock\_name] -= quantity

elif event\_type == "CHANGE":

price\_change = int(event\_list[2])

stock\_price[stock\_name] = stock\_price.get(stock\_name, 0) + price\_change

net\_profit += price\_change \* (stock\_quantity.get(stock\_name, 0))

elif event\_type == "QUERY":

result.append(net\_profit)

return result

COMPLEXITY ANALYSIS

Time Complexity - O(n)

The code iterates through each event in the input list once in O(n) time.

Space Complexity - O(n)

The solution uses two dictionaries to store the current quantity and price of each stock. Assuming the

names of the stock are not large, the space complexity will be O(n).

Java 17

class Result {

/\*

\* Complete the 'getNetProfit' function below.

\*

\* The function is expected to return a LONG\_INTEGER\_ARRAY.

\* The function accepts STRING\_ARRAY events as parameter.

\*/

public static List<Long> getNetProfit(List<String> events) {

**// Write your code here**

**Map<String, Long> companyTrack = new HashMap<>();**

**List<Long> out = new ArrayList<>();**

**long profit = 0L;**

**for (String s1 : events) {**

**String command = "";**

**long value = 0L;**

**String[] arr = s1.split(" ");**

**command = arr[0];**

**if (!command.equals("QUERY")) {**

**value = Long.parseLong(arr[2]);**

**} else {**

**out.add(profit);**

**}**

**if (command.equals("BUY")) {**

**companyTrack.put(arr[1], companyTrack.getOrDefault(arr[1], 0L) + value);**

**}**

**if (command.equals("CHANGE")) {**

**profit += value \* companyTrack.get(arr[1]);**

**}**

**if (command.equals("SELL")) {**

**companyTrack.put(arr[1], companyTrack.getOrDefault(arr[1], 0L) - value);**

**}**

**}**

**return out;**

**}**

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int eventsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> events = IntStream.range(0, eventsCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

List<Long> result = Result.getNetProfit(events);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**Java Basics: Bank Implementation**

Coding Java Interfaces OOPS Medium

Question description

Given an interface termed Bank, list the following functions:

void assignLoans(int[] loans);

void averageLoan();

void maxLoan();

void minLoan();

Create 2 classes, called PersonalLoanDept and BusinessLoanDept implementing the Bank interface. The

specifications are given below.

1. The class PersonalLoanDept should include an integer-type array termed loanAmounts. This class

should further comprise the subsequent methods:

11 of 19

PersonalLoanDept(int clients): An empty array loanAmounts of clients length is initialized in this class,

where clients is the count of loan recipients. The initial loan amount assigned is zero.

void assignLoans(int[] loans): The loans array is linked to loanAmounts. If the lengths of the two

arrays differ, as many values as possible are assigned, and then stop allocating more and print

"Loans for clients processed".

void averageLoan(): This displays the loan average in the pattern "Average loan amount for clients is

{averageLoan}". The average computation should consider any zero value present in

loanAmountsn and should be rounded to 2 decimal places.

void maxLoan(): This displays "Maximum loan amount amongst clients is {maximumLoan}" reflecting

the largest loan.

void minLoan(): This displays "Minimum loan amount amongst clients is {minimumLoan}" reflecting

the smallest loan given.

2. The BusinessLoanDept class necessitates an int[] as a variable denoted loanAmounts and should

implement subsequent methods:

BusinessLoanDept(int businesses): This initializes an empty array loanAmounts of length

businesses which signifies the number of business recipients.

void assignLoans(int[] loans): This assigns loans array to loanAmounts. The system discontinues

further allocation if the lengths of arrays do not match post the assignment of possible values and

prints "Loans for businesses processed".

void averageLoan(): This prints "Average loan amount for businesses is {averageLoan}". Any residual

zero values in loanAmounts are included in the average calculation.

void maxLoan(): This prints "Maximum loan amongst businesses is {maximumLoan}" reflecting the

highest loan.

void minLoan(): This prints "Minimum loan amongst businesses is {minimumLoan}" reflecting the

lowest loan value.

Ensure the usage of inheritance and encapsulation to prevent redundant code. Implementation of the

PersonalLoanDept and BusinessLoanDept classes is checked using the interface Bank provided in the

locked code stub.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains two space-separated integers, n, and m, the count of personal and business

loan applicants respectively.

The second line contains space-separated integers that represent the loans of clients.

12 of 19

The third line contains space-separated integers that represent the loans of businesses.

SAMPLE CASE 0

Sample Input For Custom Testing

4 4

2348 929 1284 5543

3117 5196 3352 7068

Sample Output

Loans for clients processed

Loans for businesses processed

Average loan amount for clients is 2526.00

Maximum loan amount amongst clients is 5543

Minimum loan amount amongst clients is 929

Average loan amount for businesses is 4683.25

Maximum loan amongst businesses is 7068

Minimum loan amongst businesses is 3117

SAMPLE CASE 1

Sample Input For Custom Testing

5 3

1500 3000 4500

1000 2000

Sample Output

Loans for clients processed

Loans for businesses processed

Average loan amount for clients is 1800.00

Maximum loan amount amongst clients is 4500

Minimum loan amount amongst clients is 0

Average loan amount for businesses is 1000.00

Maximum loan amongst businesses is 2000

Minimum loan amongst businesses is 0

SAMPLE CASE 2

Sample Input For Custom Testing

13 of 19

4 4

1000 2000 0 3000

4000 0 5000 6000

Sample Output

Loans for clients processed

Loans for businesses processed

Average loan amount for clients is 3000.00

Maximum loan amount amongst clients is 4000

Minimum loan amount amongst clients is 2000

Average loan amount for businesses is 2625.00

Maximum loan amongst businesses is 4500

Minimum loan amongst businesses is 0

SAMPLE CASE 3

Sample Input For Custom Testing

5 3

1500 3000 4500

1000 2000

Sample Output

Loans for clients processed

Loans for businesses processed

Average loan amount for clients is 1500.00

Maximum loan amount amongst clients is 3000

Minimum loan amount amongst clients is 0

Average loan amount for businesses is 3750.00

Maximum loan amongst businesses is 6000

Minimum loan amongst businesses is 0

Interviewer guidelines

SOLUTION

Java 17

interface Bank {

void assignLoans(int[] loans);

void averageLoan();

void maxLoan();

void minLoan();

}

**class PersonalLoanDept implements Bank {**

**int[] loanAmounts;**

**public PersonalLoanDept(int clients) {**

**loanAmounts = new int[clients];**

**}**

**public void assignLoans(int[] loans) {**

**for (int i = 0; i < loanAmounts.length; i++) {**

**if (i < loans.length) {**

**loanAmounts[i] = loans[i];**

**}**

**}**

**System.out.println("Loans for clients processed");**

**}**

**public void averageLoan() {**

**double average = Arrays.stream(loanAmounts).average().orElse(0);**

**System.out.printf("Average loan amount for clients is %.2f", average);**

**System.out.println();**

**}**

**public void maxLoan() {**

**int max = Arrays.stream(loanAmounts).max().orElse(0);**

**System.out.println("Maximum loan amount amongst clients is " + max);**

**}**

**public void minLoan() {**

**int min = Arrays.stream(loanAmounts).min().orElse(0);**

**System.out.println("Minimum loan amount amongst clients is " + min);**

**}**

**}**

**class BusinessLoanDept implements Bank {**

**int[] loanAmounts;**

**public BusinessLoanDept(int business) {**

**loanAmounts = new int[business];**

**}**

**public void assignLoans(int[] loans) {**

**for (int i = 0; i < loanAmounts.length; i++) {**

**if (i < loans.length) {**

**loanAmounts[i] = loans[i];**

**}**

**}**

**System.out.println("Loans for businesses processed");**

**}**

**public void averageLoan() {**

**double average = Arrays.stream(loanAmounts).average().orElse(0);**

**System.out.printf("Average loan amount for businesses is %.2f", average);**

**System.out.println();**

**}**

**public void maxLoan() {**

**int max = Arrays.stream(loanAmounts).max().orElse(0);**

**System.out.println("Maximum loan amongst businesses is " + max);**

**}**

**public void minLoan() {**

**int min = Arrays.stream(loanAmounts).min().orElse(0);**

**System.out.println("Minimum loan amongst businesses is " + min);**

**}**

**}s**

public class Solution {

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

Scanner sc = new Scanner(System.in);

String[] count = sc.nextLine().split(" ");

PersonalLoanDept p = new PersonalLoanDept(Integer.parseInt(count[0]));

BusinessLoanDept b = new BusinessLoanDept(Integer.parseInt(count[1]));

count = sc.nextLine().split(" ");

int[] loansClients = new int[count.length];

for (int i = 0; i < count.length; i++) {

loansClients[i] = Integer.parseInt(count[i]);

}

p.assignLoans(loansClients);

count = sc.nextLine().split(" ");

int[] loansBusinesses = new int[count.length];

for (int i = 0; i < count.length; i++) {

loansBusinesses[i] = Integer.parseInt(count[i]);

}

b.assignLoans(loansBusinesses);

p.averageLoan();

p.maxLoan();

p.minLoan();

b.averageLoan();

b.maxLoan();

b.minLoan();

}

}

**Java: Exchange Rate**

Coding Overloading OOPS Java Polymorphism Easy

Question description

In the scope of Risk Management in global finance, you are tasked with implementing a class named

exchangeRate to optimize investments. This class should include three methods with the same label but

different arguments.

11 of 14

Note that method overloading allows a class to have multiple methods with the same name but different

argument lists.

Using method overloading, complete the implementation of the class exchangeRate with 3 methods.

1. public String rate(double localCurrency, String foreignCurrency) needs to process arguments like

rate(1, "Euro") and should return "1Euro".

2. public String rate(double localCurrency, double foreignCurrency) must return the sum (as exchange

rate approximation) when given inputs like rate(1.1,2.2). The resulting sum (like 3.3) should be

rounded off to two decimal places.

Rounding Rules: Use the BigDecimal class with RoundingMode.HALF\_UP for rounding. Follow

these rounding rules:

3.765 => 3.77

3.7649 => 3.76

3.778 => 3.79

3. public String rate(String localCurrency, String foreignCurrency), should handle inputs like

rate("dollars,","yen") and return the concatenated string: "dollarsyen".

When code is submitted, the provided Solution class will test the add methods with different arguments.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of inputs.

Each of the next n lines contains 2 space-separated values.

SAMPLE CASE 0

Sample Input For Custom Testing

24

1

2 2

Sample Output

5.0

4.0

SAMPLE CASE 1

Sample Input For Custom Testing

31

Euro

dollars yen

5 7

Sample Output

1Euro

dollarsyen

12.0

Interviewer guidelines

SOLUTION

:Java 7

**class exchangeRate {**

**public String rate(double localCurrency, String foreignCurrency) {**

**return String.valueOf((int) Math.round(localCurrency)) + foreignCurrency;**

**}**

**public String rate(String localCurrency, String foreignCurrency) {**

**return localCurrency + foreignCurrency;**

**}**

**public String rate(double localCurrency, double foreignCurrency) {**

**double sum = localCurrency + foreignCurrency;**

**BigDecimal bd = new BigDecimal(Double.toString(sum));**

**bd = bd.setScale(2, RoundingMode.HALF\_UP);**

**String result = bd.toString();**

**if (result.endsWith(".00")) {**

**result = result.substring(0, result.length() - 1);**

**}**

**return result;**

**}**

**}**

**public class Solution {**

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

**/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/**

**Scanner sc = new Scanner(System.in);**

**String input = sc.nextLine();**

**int n = Integer.parseInt(input);**

**for (int i = 0; i < n; i++) {**

**String[] entries = sc.nextLine().split(" ");**

**Boolean isFirstEntryDouble = isFloat(entries[0]);**

**Boolean isSecondEntryDouble = isFloat(entries[1]);**

**exchangeRate currencyExchange = new exchangeRate();**

**if (isFirstEntryDouble && isSecondEntryDouble) {**

**System.out.println(currencyExchange.rate(Double.parseDouble(entries[0]), Double.parseDouble(entries[1])));**

**} else {**

**System.out.println(currencyExchange.rate(entries[0], entries[1]));**

**}**

**}**

**}**

**private static final Pattern DOUBLE\_PATTERN = Pattern.compile(**

**"[\\x00-\\x20]\*[+-]?(NaN|Infinity|((((\\p{Digit}+)(\\.)?((\\p{Digit}+)?)" +**

**"([eE][+-]?(\\p{Digit}+))?)|(\\.((\\p{Digit}+))([eE][+-]?(\\p{Digit}+))?)|" +**

**"(((0[xX](\\p{XDigit}+)(\\.)?)|(0[xX](\\p{XDigit}+)?(\\.)" +**

**"(\\p{XDigit}+)))[pP][+-]?(\\p{Digit}+)))[fFdD]?))[\\x00-\\x20]\*");**

**public static boolean isFloat(String s) {**

**return DOUBLE\_PATTERN.matcher(s).matches();**

**}**

**}**

**Java Exceptions: Securing Messages**

Coding Easy Java Exception Handling Language Proficiency Static Methods Strings

Question description

Create an Encrypter class to handle input validation and encryption. It should have the method String

getEncryptedName(String name) that performs a simple string manipulation.

The getEncryptedName method must perform the following tasks:

Validate the name by calling the validate(name) method provided in the Validator class

If the name validation returns true:

Reverse the entire string and convert upper-case letters to lower-case

Return the modified string

If the name validation returns false:

Throw an exception of IllegalArgumentException class with the message 'Try again with valid

name'

The locked stub code in the editor provides the complete implementation of the Validator class and

validates the implementation of the Encrypter class.

Constraints

The name contains no more than 100 characters.

INPUT FORMAT FOR CUSTOM TESTING

There is a single line that contains the string name.

SAMPLE CASE 0

Sample Input

STDIN Function

----- -----

Kate Winslet → name = 'Kate Winslet'

Sample Output

telsniw etak

Explanation

The name validation is successful, so the string is reversed and converted to lower case.

SAMPLE CASE 1

Sample Input

STDIN Function

----- -----

Kate Wins?let → name = 'Kate Wins?let'

Sample Output

java.lang.IllegalArgumentException: Try again with valid name

Explanation

Invalid characters are found in the name, so the required exception is thrown with the message 'Try

again with valid name'.

Java 17

**/\***

**\* Create the Encrypter class here.**

**\*/**

**class Encrypter {**

**public static String getEncryptedName(String name) {**

**Validator v = new Validator();**

**boolean ans = v.validate(name);**

**StringBuilder sb = new StringBuilder();**

**if (ans) {**

**String lname = name.toLowerCase();**

**sb.append(lname);**

**sb.reverse();**

**} else {**

**throw new IllegalArgumentException("Try again with valid name");**

**}**

**return sb.toString();**

**}**

**}**

**class Validator {**

**public boolean validate(String name) {**

**for (int i = 0; i < name.length(); i++) {**

**char ch = name.charAt(i);**

**if (ch != ' ' && !(Character.isLowerCase(ch) || Character.isUpperCase(ch))) {**

**return false;**

**}**

**}**

**return true;**

**}**

**}**

public class Solution {

private static final Scanner INPUT\_READER = new Scanner(System.in);

public static void main(String[] args) {

String name = INPUT\_READER.nextLine();

try {

System.out.println(Encrypter.getEncryptedName(name));

} catch (Exception e) {

System.out.println(e);

}

}

}

**Java Basics: Nutrition Chain**

Question description

Nutrition in food can be broken down into proteins, fats, and carbohydrates. Implement the following

classes about food and nutrition to complete this challenge:

1. abstract class Food with the following properties:

double proteins

double fats

double carbs

double tastyScore

void getMacroNutrients [Abstract Method]

2. class Egg which extends class Food and has the following properties:

Constructor to initialize the attributes (proteins, fats, and carbs) in the same order.

int tastyScore = 7

String type = "non-vegetarian"

void getMacroNutrients => prints("An egg has [this.proteins] gms of protein, [this.fats] gms of fats

and [this.carbs] gms of carbohydrates.")

3. class Bread which extends class Food and has the following properties:

Constructor to initialize the attributes (proteins, fats, and carbs) in the same order.

int tastyScore = 8

String type = "vegetarian"

void getMacroNutrients => prints(" A slice of bread has [this.proteins] gms of protein, [this.fats] gms

of fats and [this.carbs] gms of carbohydrates.")

Note: The code stub handles input.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of food items.

Every food item takes input in the next 4 lines where the first line is the name of the food and the next

three lines are method calls (getType, getTaste, and getMacros) in random order.

SAMPLE CASE 0

Sample Input

1

Bread

getType

getMacros

getTaste

Sample Output

Bread is vegetarian

A slice of bread has 4.0 gms of protein, 1.1 gms of fats and 13.8 gms of carbohydrates.

Taste: 8

SAMPLE CASE 1

Sample Input

1 Egg

getMacros

getTaste

getType

Sample Output

An egg has 6.3 gms of protein, 5.3 gms of fats and 0.6 gms of carbohydrates.

Taste: 7

Egg is non-vegetarian

Java 7

**abstract class Food {**

**protected double proteins;**

**protected double fats;**

**protected double carbs;**

**protected double tastyScore;**

**abstract void getMacroNutrients();**

**}**

**class Egg extends Food {**

**public int tastyScore = 7;**

**public String type = "non-vegetarian";**

**public Egg(double proteins, double fats, double carbs) {**

**this.proteins = proteins;**

**this.fats = fats;**

**this.carbs = carbs;**

**}**

**@Override**

**void getMacroNutrients() {**

**System.out.println("An egg has " + this.proteins + " gms of protein, "**

**+ this.fats + " gms of fats and " + this.carbs + " gms of carbohydrates.");**

**}**

**public int getTastyScore() {**

**return tastyScore;**

**}**

**public String getType() {**

**return type;**

**}**

**}**

**class Bread extends Food {**

**public int tastyScore = 8;**

**public String type = "vegetarian";**

**public Bread(double proteins, double fats, double carbs) {**

**this.proteins = proteins;**

**this.fats = fats;**

**this.carbs = carbs;**

**}**

**@Override**

**void getMacroNutrients() {**

**System.out.println("A slice of bread has " + this.proteins + " gms of protein, "**

**+ this.fats + " gms of fats and " + this.carbs + " gms of carbohydrates.");**

**}**

**public int getTastyScore() {**

**return tastyScore;**

**}**

**public String getType() {**

**return type;**

**}**

**}**

**public class Solution {**

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

**Scanner sc = new Scanner(System.in);**

**int cnt = Integer.parseInt(sc.nextLine());**

**for (int i = 0; i < cnt; i++) {**

**String name = sc.nextLine();**

**if (name.equals("Bread")) {**

**Bread breadObj = new Bread(4, 1.1, 13.8);**

**for (int j = 0; j < 3; j++) {**

**String command = sc.nextLine();**

**if (command.equals("getMacros")) {**

**breadObj.getMacroNutrients();**

**} else if (command.equals("getTaste")) {**

**System.out.println("Taste: " + breadObj.tastyScore);**

**} else if (command.equals("getType")) {**

**System.out.println("Bread is " + breadObj.type);**

**}**

**}**

**} else if (name.equals("Egg")) {**

**Egg eggObj = new Egg(6.3, 5.3, 0.6);**

**for (int j = 0; j < 3; j++) {**

**String command = sc.nextLine();**

**if (command.equals("getMacros")) {**

**eggObj.getMacroNutrients();**

**} else if (command.equals("getTaste")) {**

**System.out.println("Taste: " + eggObj.tastyScore);**

**} else if (command.equals("getType")) {**

**System.out.println("Egg is " + eggObj.type);**

**}**

**}**

**}**

**}**

**}**

**}**

**Problem Solving.:Same Substring**

Coding Easy Strings Two Pointers

Question description

Two strings, s, and t, each of length n, that contain lowercase English characters are given as well as an

integer K.

The cost to change the i character in s from s to t is the absolute difference of the ASCII value of

characters, i.e., abs( s - t ).

Find the maximum length of a substring of s that can be changed to the corresponding substring of t

with a total cost less than or equal to K. If there is no such substring, return 0.

Example

It is given that s = "adpgki", t = "cdmxki", K = 6.

Change s from 'a' to 'c' with cost = abs('a' - 'c') = 2. String s is now "cdpgki" and K = 6 - 2 = 4.

Change s from 'p' to 'm' with cost = abs('p' - 'm') = 3. String s is "cdmgki" and K = 4 - 3 = 1.

The only character left to change is 'g' to 'x', which costs more than K.

The longest substring in s that is equal to the corresponding substring in t is s[0, 2] = t[0, 2].

Hence, the answer is 3.

Function Description

Complete the function sameSubstring in the editor below.

sameSubstring has the following parameters:

string s: the string to alter

th

i i

i i

0

2

string t: the string to match

int K: the maximum sum of costs

Returns

int: the maximum length of a substring that can be obtained

Constraints

1 ≤ n ≤ 2 x 10

0 ≤ K ≤ 10

Strings s and t contain lowercase English letters only.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains string s.

The second line contains the string t.

The last line contains an integer, K.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

uaccd → s = "uaccd"

gbbeg → t = "gbbeg"

4 → K = 4

Sample Output

3

Explanation

index in s current new new s cost K

1 a b ubccd 1 3

2 c b ubbcd 1 2

3 c e ubbed 2 0

5

6

Substring s[1, 3] is equal to substring t[1, 3].

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

hffk → s = "hffk"

larb → t = "larb"

3 → K = 3

Sample Output

0

Explanation

The cost to change any s to t is greater than K = 3. No characters can be changed, and none match.

Interviewer guidelines

SOLUTION

Skills: Loops, Algorithms, Problem Solving

This problem can be solved using two pointers.

Keep the left pointer fixed and increase the right pointer until the current cost is less than equal to K.

If the cost becomes greater than K, then increase the left pointer until the current cost is greater than

K.

Update the answer with the maximum of answer and (right pointer - left pointer + 1).

Java 17

class Result {

/\*

\* Complete the 'sameSubstring' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. STRING s

\* 2. STRING t

\* 3. INTEGER K

\*/

**public static int sameSubstring(String s, String t, int K) {**

**//Write the code heres**

**int left = 0;**

**int max = 0;**

**int current = 0;**

**for (int r = 0; r < s.length(); r++) {**

**current += Math.abs(s.charAt(r) - t.charAt(r));**

**while (current > K) {**

**current -= Math.abs(s.charAt(left) - t.charAt(left));**

**left++;**

**}**

**max = Math.max(max, r - left + 1);**

**}**

**return max;**

**}**

**}**

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String s = bufferedReader.readLine();

String t = bufferedReader.readLine();

int K = Integer.parseInt(bufferedReader.readLine().trim());

int result = Result.sameSubstring(s, t, K);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

**Java Basics: Zoo Management**

Coding Inheritance Interfaces Easy Java

Question description

A product development team has been asked to create a system that manages the animals in the zoo.

Create a new class called Animal and implement the IAnimal interface.

Inside the Animal class, define the following properties:

id(int): the unique Id of the animal

species(string): the species of the animal

name(string): the name of the animal

age(int): the age of the animal

Create a new class called Zoo and implement the IZoo interface.

Add a private field animals of type List<IAnimal> to store the animals in the zoo.

Implement the following methods in the Zoo class:

addAnimal(IAnimal animal): adds the animal to the list of animals

removeAnimal(int id): removes the animal with the specified ID from the list of animals

countAnimals(): returns the number of animals in the list

getAnimalsBySpecies(String species): returns a list of animals in the list that match

getAnimalsByAge(): returns a list of map entries, where each map entry contains an age as key

and a list of animals with that age as value

Example

There are 2 animal objects with Id, Species, Name, and Age.

1 Mammals Tiger 6

2 Mammals Whale 8

After creating the list of animals, count the animals.

There are 2 animals in the zoo.

List the animals according to their species.

Mammals:

Tiger (6 years old)

Whale (8 years old)

Group and list the animals by age.

6 year(s) old:

- Tiger (Mammals)

8 year(s) old:

- Whale (Mammals)

Remove one animal from among them and calculate the total number of animals again.

There is now 1 animal in the zoo.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer n, the number of animals.

Each of the next n lines contains the animal information separated by space (Id Species Name Age).

The next line contains a string, the species to filter.

The next line contains the integer m, which is the animal Id to remove.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN Function

----- --------

3 number of animals n = 3

1 Spec-2 Animal-0 17 1st animal: Id = 1, Species = "Spec-2" Name = "Animal-0" Age = 17

2 Spec-2 Animal-1 11

3 Spec-3 Animal-2 9

Spec-2 species to filter is "Spec-2"

2 remove animal with Id = 2

Sample Output

There are 3 animals in the zoo

Spec-2:

Animal-0 (17 years old)

Animal-1 (11 years old)

Animals by age:

17 year(s) old:

- Animal-0 (Spec-2)

11 year(s) old:

- Animal-1 (Spec-2)

9 year(s) old:

- Animal-2 (Spec-3)

There are now 2 animals in the zoo

Explanation

There are 3 animals to add to the list. Filter the list by "Spec-2". Group animals by age. Remove the

animal with Id = 2, and determine how many animals are left.

SAMPLE CASE 1

Sample Input For Custom Testing

41

Spec-2 Animal-0 18

2 Spec-1 Animal-1 2

3 Spec-3 Animal-2 10

4 Spec-2 Animal-3 12

Spec-3

3

Sample Output

There are 4 animals in the zoo

Spec-3:

Animal-2 (10 years old)

Animals by age:

18 year(s) old:

- Animal-0 (Spec-2)

12 year(s) old:

- Animal-3 (Spec-2)

10 year(s) old:

- Animal-2 (Spec-3)

2 year(s) old:

- Animal-1 (Spec-1)

There are now 3 animals in the zoo

Explanation

There are 4 animals to add to the list. Filter the list by "Spec-3". Group animals by age. Remove the

animal with Id = 3, and determine how many animals are left.

Interviewer guidelines

Solution:

**class Animal implements IAnimal {**

**private int id;**

**private String species;**

**private String name;**

**private int age;**

**@Override**

**public void setId(int id) {**

**this.id = id;**

**}**

**@Override**

**14 of 20**

**public int getId() {**

**return id;**

**}**

**@Override**

**public void setSpecies(String species) {**

**this.species = species;**

**}**

**@Override**

**public String getSpecies() {**

**return species;**

**}**

**@Override**

**public void setName(String name) {**

**this.name = name;**

**}**

**@Override**

**public String getName() {**

**return name;**

**}**

**@Override**

**public void setAge(int age) {**

**this.age = age;**

**}**

**@Override**

**public int getAge() {**

**return age;**

**}**

**}**

**class Zoo implements IZoo {**

**private List<IAnimal> animals = new ArrayList<>();**

**@Override**

**public void addAnimal(IAnimal animal) {**

**animals.add(animal);**

**}**

**@Override**

**public void removeAnimal(int id) {**

**Iterator<IAnimal> iterator = animals.iterator();**

**15 of 20**

**while (iterator.hasNext()) {**

**IAnimal animal = iterator.next();**

**if (animal.getId() == id) {**

**iterator.remove();**

**}**

**}**

**}**

**@Override**

**public int countAnimals() {**

**return animals.size();**

**}**

**@Override**

**public List<IAnimal> getAnimalsBySpecies(String species) {**

**List<IAnimal> specAnimals = new ArrayList<>();**

**for (IAnimal animal : animals) {**

**if (animal.getSpecies().equals(species)) {**

**specAnimals.add(animal);**

**}**

**}**

**return specAnimals;**

**}**

**@Override**

**public List<Map.Entry<Integer, List<IAnimal>>> getAnimalsByAge() {**

**Map<Integer, List<IAnimal>> ageGroups = new HashMap<>();**

**for (IAnimal animal : animals) {**

**int age = animal.getAge();**

**if (!ageGroups.containsKey(age)) {**

**ageGroups.put(age, new ArrayList<IAnimal>());**

**}**

**ageGroups.get(age).add(animal);**

**}**

**List<Map.Entry<Integer, List<IAnimal>>> result = new ArrayList<>(ageGroups.entrySet());**

**Collections.sort(result, new Comparator<Map.Entry<Integer, List<IAnimal>>>() {**

**@Override**

**public int compare(Map.Entry<Integer, List<IAnimal>> o1, Map.Entry<Integer, List<IAnimal>>**

**o2) {**

**return o2.getKey().compareTo(o1.getKey());**

**}**

**});**

**return result;**

**}**

16 of 20

}

: Java 15

interface IAnimal {

void setId(int id);

int getId();

void setSpecies(String species);

String getSpecies();

void setName(String name);

String getName();

void setAge(int age);

int getAge();

}

interface IZoo {

void addAnimal(IAnimal animal);

void removeAnimal(int id);

int countAnimals();

List<IAnimal> getAnimalsBySpecies(String species);

List<Map.Entry<Integer, List<IAnimal>>> getAnimalsByAge();

}

class Animal implements IAnimal {

private int id;

private String species;

private String name;

private int age;

@Override

public void setId(int id) {

this.id = id;

}

public int getId() {

return id;

}

@Override

public void setSpecies(String species) {

this.species = species;

}

public String getSpecies() {

return species;

}

@Override

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

@Override

public void setAge(int age) {

this.age = age;

}

public int getAge() {

return age;

}

}

class Zoo implements IZoo {

private List<IAnimal> animals = new ArrayList<>();

@Override

public void addAnimal(IAnimal animal) {

animals.add(animal);

}

@Override

public void removeAnimal(int id) {

Iterator<IAnimal> iterator = animals.iterator();

while (iterator.hasNext()) {

IAnimal animal = iterator.next();

if (animal.getId() == id) {

iterator.remove();

}

}

}

@Override

public int countAnimals() {

return animals.size();

}

@Override

public List<IAnimal> getAnimalsBySpecies(String species) {

List<IAnimal> specAnimals = new ArrayList<>();

for (IAnimal animal : animals) {

if (animal.getSpecies().equals(species)) {

specAnimals.add(animal);

}

}

return specAnimals;

}

@Override

public List<Map.Entry<Integer, List<IAnimal>>> getAnimalsByAge() {

Map<Integer, List<IAnimal>> ageGroups = new HashMap<>();

for (IAnimal animal : animals) {

int age = animal.getAge();

ageGroups.putIfAbsent(age, new ArrayList<>());

ageGroups.get(age).add(animal);

}

List<Map.Entry<Integer, List<IAnimal>>> result = new ArrayList<>(ageGroups.entrySet());

result.sort((o1, o2) -> o2.getKey().compareTo(o1.getKey()));

return result;

}

}

public class Solution {

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

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

PrintWriter out = new PrintWriter(System.out);

IZoo zoo = new Zoo();

int aCount = Integer.parseInt(br.readLine().trim());

for (int i = 1; i <= aCount; i++) {

String[] a = br.readLine().trim().split(" ");

IAnimal e = new Animal();

e.setId(Integer.parseInt(a[0]));

e.setSpecies(a[1]);

e.setName(a[2]);

e.setAge(Integer.parseInt(a[3]));

zoo.addAnimal(e);

}

out.println("There are " + zoo.countAnimals() + " animals in the zoo");

String[] b = br.readLine().trim().split(" ");

String species = b[0];

List<IAnimal> specAnimals = zoo.getAnimalsBySpecies(species);

out.println(species + ":");

for (IAnimal sp : specAnimals) {

out.println("" + sp.getName() + " (" + sp.getAge() + " years old)");

}

List<Map.Entry<Integer, List<IAnimal>>> animalsByAge = zoo.getAnimalsByAge();

out.println("Animals by age:");

for (Map.Entry<Integer, List<IAnimal>> groups : animalsByAge) {

out.println("" + groups.getKey() + " year(s) old:");

for (IAnimal animal : groups.getValue()) {

out.println("- " + animal.getName() + " (" + animal.getSpecies() + ")");

}

}

String[] c = br.readLine().trim().split(" ");

int id = Integer.parseInt(c[0]);

zoo.removeAnimal(id);

out.println("There are now " + zoo.countAnimals() + " animals in the zoo");

out.flush();

out.close();

}

}

**Problem Solving:Array Reduction 3**

Question description

Given an array arr of n integers, a sequence of n-1 operations must be performed on the array.

In each operation,

Remove the minimum and maximum elements from the current array and add their sum back to the

array.

The cost of an operation, cost = ceil((minimum\_element + maximum\_element) / (maximum\_elementminimum\_

element + 1)) .

Find the total cost to reduce the array to a single element.

Example

Given arr = [2, 3, 4, 5, 7].

The possible sequence of operations are:

1. Choose 2 and 7, the cost = ceil[(2 + 7) / (7 - 2 + 1)] = ceil[9 / 6] = 2. Remove 2 and 7, append 9, arr = [3,

4, 5, 9], total\_cost = 2.

2. Choose 3 and 9, the cost = ceil[(3 + 9) / (9 - 3 + 1)] = ceil[12 / 7] = 2, arr = [4, 5, 12], total\_cost = 2 + 2 =

4.

3. Choose 4 and 12, the cost = ceil[(4 + 12) / (12 - 4 + 1)] = ceil[16 / 9] = 2, arr = [5, 16], total\_cost = 4 + 2 =

6.

4. Choose 5 and 16, the cost = ceil[(5 + 16) / (16 - 5 + 1)] = ceil[21 / 12] = 2, arr = [21], total\_cost = 6 + 2 =

8.

4 of 14

Return the total cost, 8.

Function Description

Complete the function findTotalCost in the editor below.

findTotalCost has the following parameter:

int arr[n]: the array to be reduced

Returns

int: the total cost to reduce the array to 1 element

Constraints

2 ≤ n ≤ 2 . 10

1 ≤ arr[i] ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of elements in arr.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains an integer describing arr[i].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

6 → arr[] size n = 6

3 → arr = [3, 5, 2, 1, 9, 6]

52196

Sample Output

10

Explanation

The possible sequence of operations are:

5

6

5 of 14

1. Choose 1 and 9, the cost = ceil[(1 + 9) / (9 - 1 + 1)] = ceil[10/ 9] = 2. Remove 1 and 9, append 10, arr

= [3, 5, 2, 6, 10], total\_cost = 2.

2. Choose 2 and 10, the cost = ceil[(2 + 10) / (10 - 2 + 1)] = ceil[12 / 9] = 2, arr = [3, 5, 6, 12], total\_cost =

2 + 2 = 4.

3. Choose 3 and 12, the cost = ceil[(3 + 12) / (12 - 3 + 1)] = ceil[15 / 10] = 2, arr = [5, 6, 15], total\_cost =

4 + 2 = 6.

4. Choose 5 and 15, the cost = ceil[(5 + 15) / (15 - 5 + 1)] = ceil[20 / 11] = 2, arr = [6, 20], total\_cost = 6 +

2 = 8.

5. Choose 6 and 20, the cost = ceil[(6 + 20) / (20 - 6 + 1)] = ceil[26 / 15] = 2, arr = [26], total\_cost = 8 + 2

= 10.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

4 → arr[] size n = 4

8 → arr = [8, 8, 8, 8]

888

Sample Output

21

Explanation

The possible sequence of operations are:

1. Choose 8 and 8, cost = ceil[(8 + 8) / (8 - 8 + 1)] = ceil[16 / 1] = 16, arr = [8, 8, 16], total\_cost = 16.

2. Choose 8 and 16, cost = ceil[(8 + 16) / (16 - 8+ 1)] = ceil[24 / 9] = 3, arr = [8, 24], total\_cost = 16 + 3 =

19.

3. Choose 8 and 24, cost = ceil[(8 + 24) / (24 - 8 + 1)] = ceil[32 / 17] = 2, arr = [32], total\_cost = 19 + 2 =

21.

Interviewer guidelines

SOLUTION

6 of 14

Skills: Arrays, Sorting, Implementation

Optimal Solution:

In order to find the total cost, sort the array so the minimum element is found at the zeroth index and

the maximum is at the last index. One observation that can be useful to do this task optimally is after

the operation, we add minimum + maximum to the array, and the newly added element will be our

new maximum for the array after this operation. We don’t have to calculate the minimum and

maximum for each operation.

Time Complexity: O(n logn)

Space Complexity: O(n)

COMPLEXITY ANALYSIS

Time Complexity - O(n logn)

Sorting the array takes O(n logn) time. Then, each element except the last is iterated over and merged

with the maximum, which takes O(n) time. Hence, the total time complexity is O(n logn + n) which is of

the order O(n logn).

Space Complexity - O(n)

Since we are using an inbuilt sort function, hence space complexity is O(n).

: Java 17

class Result {

/\*

\* Complete the 'findTotalCost' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

public static int findTotalCost(List<Integer> arr) {

**//Write The Code Here**

**float cost = 0f;**

**Collections.sort(arr);**

**while (arr.size() > 1) {**

**float min = arr.get(0);**

**float max = arr.get(arr.size() - 1);**

**cost += Math.ceil((min + max) / (max - min + 1));**

**arr.remove(arr.size() - 1);**

**arr.remove(0);**

**arr.add((int) (min + max));**

**}**

**return (int) cost;**

**}**

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int arrCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = IntStream.range(0, arrCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int result = Result.findTotalCost(arr);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

**Problem Solving:Optimal String**

Coding Greedy Algorithms Easy Problem Solving

Question description

A string is to be constructed using only the characters 'A' and 'B'. Given four integers, countA, countB,

maxA, maxB, the constructed string is said to be optimal if:

There are at most countA 'A' characters, and countB 'B' characters.

Each substring of only 'A's contains at most maxA 'A' characters.

Each substring of only 'B's contains at most maxB 'B' characters.

HackerRank organized fun trivia for its employees where it asked for the maximum possible length of

an optimal string that can be constructed satisfying the criteria above. The goal is to find the maximum

possible length of an optimal string.

Note:

There can be multiple optimal strings with the same maximal length.

A substring of a string is a contiguous subsegment of the string.

Example:

Consider countA = 3, countB = 5, maxA = 1, maxB = 1.

The maximal length optimal string that can be constructed is 'BABABAB' whose length is 7. Thus, the

answer is 7.

4 of 15

Function Description

Complete the function getOptimalStringLength in the editor below.

getOptimalStringLength has the following parameters:

int countA: the maximum count of character 'A'

int countB: the maximum count of character 'B'

int maxA: the maximum substring length of character 'A'

int maxB: the maximum substring length of character 'B'

Returns:

long\_int: the maximum length of optimal string that can be constructed

Constraints

0 ≤ countA, countB, maxA, maxB ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, countA, denoting the maximum count of character 'A'

The next line contains an integer, countB, denoting the maximum count of character 'B'.

The next line contains an integer, maxA, denoting the maximum substring length of character 'A'.

The next line contains an integer, maxB, denoting the maximum substring length of character 'B'.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

2 → countA = 2

4 → countB = 4

2 → maxA = 2

1 → maxB = 1

Sample Output

5

Explanation

The maximal length optimal string is 'BABAB'.

SAMPLE CASE 1

Sample Input For Custom Testing

6

5 of 15

STDIN FUNCTION

----- --------

3 → countA = 3

3 → countB = 3

3 → maxA = 3

3 → maxB = 3

Sample Output

6

Explanation

The maximal length optimal string is 'AAABBB'.

Interviewer guidelines

HINT 1

What happens if maxA = 0 or countA = 0? What does the string look like?

Answer: All the characters in the string must be 'B'.

HINT 2

For a certain countA > maxA, how many Bs are required in order to use all the As? Can you think of a

mathematical relation?

Answer: If we divide countA As into groups of maxA length, we need 1 B to separate each group. For

example, countA = 9, maxA = 2. Divide the As as AABAABAABAABA and 4 Bs are required. There must

be at least ceiling(countA/maxA) - 1 Bs to use all the As.

SOLUTION

Skills: Implementation, Constructive Algorithms

Optimal Solution:

The problem can be broken down into several cases:

If either maxA = 0, or countA = 0, then the string will consist only of 'B's. So, the answer will

be min(maxB, countB). A similar argument also follows for the case when maxB = 0 or countB = 0.

If (countB + 1) \* maxA < countA, then there aren't enough 'B' characters to use all the 'A' characters. So,

place 'A's into groups of size maxA between each single occurrence of B. The maximum length in this

case would be countB + (countB + 1) \* maxA. A similar argument also follows for the case when (countA

+ 1) \* maxB < countB.

6 of 15

If both the above cases are not true, then it is always possible to use all 'A's and 'B's. So, the answer

is (countA + countB).

Time Complexity: O(1)

Space Complexity: O(1)

COMPLEXITY ANALYSIS

Time Complexity - O(1)

No iterations are used to compute the answer. Simple if-checks are performed, which take constant

time. Thus, the time complexity is O(1).

Space Complexity - O(1)

No extra space is needed. Thus, the space complexity is O(1).

: Java 17

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getOptimalStringLength' function below.

\*

\* The function is expected to return a LONG\_INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER countA

\* 2. INTEGER countB

\* 3. INTEGER maxA

\* 4. INTEGER maxB

\*/

public static long getOptimalStringLength(int countA, int countB, int maxA, int maxB) {

**//Write the Code Here**

**int tlength = 0;**

**int cA = 0;**

**int cB = 0;**

**while (countA > 0 || countB > 0) {**

**if ((cA < maxA && countA > countB) || cB == maxB) {**

**if (countA > 0) {**

**tlength++;**

**countA--;**

**cA++;**

**cB = 0;**

**} else {**

**break;**

**}**

**} else {**

**if (countB > 0) {**

**tlength++;**

**countB--;**

**cB++;**

**cA = 0;**

**} else {**

**break;**

**}**

**}**

**}**

**return tlength;**

**}**

}

public class Solution {

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

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int countA = Integer.parseInt(bufferedReader.readLine().trim());

int countB = Integer.parseInt(bufferedReader.readLine().trim());

int maxA = Integer.parseInt(bufferedReader.readLine().trim());

int maxB = Integer.parseInt(bufferedReader.readLine().trim());

long result = Result.getOptimalStringLength(countA, countB, maxA, maxB);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}