**Java: Calculator Classes**

Question description

Before computers were common, accountants used a mechanical adding machine for their calculations.

It did just that: it added signed numbers. If they wanted to multiply, they had to add that many times. For

example, to multiply 3 by 3, they pressed 3 <add> 3 <add> 3 <add> <total> and got 3 3 3 T9 on their

printout, resetting the sum to 0 at <total>. Luckily there was a subtotal so they could print out a value

and carry it forward.

Alex wants to build two calculators as described below:

The Adder calculator that returns the sum of two integers.

The Multiplier calculator that returns the product of two integers via addition.

In this challenge, help Alex build the calculators by writing the complete implementations of the

following two classes:

1. The Adder class should implement the method int add(int a, int b) to return the sum of two integers,

a and b. It should also print Adding integers: a b each time it is called.

2. The Multiplier class should implement the method int multiply(int a, int b, Calculator calculator) to

return the result of a x b by repeated addition using the Adderclass.

The locked stub code in the editor consists of the following:

An abstract class Calculator that contains an abstract method, int add(int a, int b).

A solution class that tests the implementation of the Adder and the Multiplier classes by

creating an object of the Adder class.

reading the inputs and passes them along with the Adder class object in the method int

multiply(int a, int b, Calculator calculator) of the Multiplier class.

Adds headers and footers

Testing Addition, Sum =

Testing Multiplication, Product =

Constraints

1 ≤ a, b ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains the first integer, a.

The next line contains the second integer, b.

SAMPLE CASE 0

Sample Input 0

3

74

Sample Output 0

Testing Addition

Adding integers: 7 4

Sum = 11

Testing Multiplication

Adding integers: 7 7

Adding integers: 14 7

Adding integers: 21 7

Product = 28

Explanation 0

Multiplying 7 by 4 is the same as adding 7 to itself 4 times.

7 + 7 = 14.

14 + 7 = 21.

21 + 7 = 28.

: Java 17

import java.util.Scanner;

**abstract class Calculator {**

**abstract int add(int a, int b);**

**}**

**/\***

**\* Write the implementations of Adder and Multiplier classes.**

**\*/**

**class Adder extends Calculator {**

**@Override**

**int add(int a, int b) {**

**System.out.println("Adding integers: " + a + " " + b);**

**return a + b;**

**}**

**}**

**class Multiplier {**

**int multiply(int a, int b, Calculator cal) {**

**int p = a;**

**for (int i = 0; i < b - 1; i++) {**

**p = cal.add(p, a);**

**}**

**return p;**

**}**

**}**

public class Solution {

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

private static final Calculator CALCULATOR = new Adder();

private static void testAddition(int a, int b) {

System.out.println("Sum = " + CALCULATOR.add(a, b));

}

private static void testMultiplication(int a, int b) {

System.out.println("Product = " + new Multiplier().multiply(a, b, CALCULATOR));

}

public static void main(String[] args) {

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

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

car inheritance

System.out.println("Testing Addition");

testAddition(a, b);

System.out.println("\nTesting Multiplication");

testMultiplication(a, b);

}

}

**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);**

**}**

**}**

**}**

**}**

**}**

**}**

**Car Inheritance**

Question description

Build on an abstract class and initialize an instance of each class with a variable. The program will then

test the implementation by retrieving the stored data.

The locked code in the editor does the following:

1. Declares an abstract class named Car with the implementations for getIsSedan() and getSeats()

methods and an abstract method named getMileage() .

2. Creates WagonR, HondaCity, or InnovaCrysta object based on input (0 for WagonR, 1 for HondaCity

and 2 for InnovaCrysta).

3. Calls the getIsSedan() , getSeats() , and getMileage() methods on the object.



The details for each car are provided below:

1. WagonR is not a sedan and has 4 seats.

2. HondaCity is a sedan and has 4 seats.

3. InnovaCrysta is not a sedan and has 6 seats.

12 of 16

Function Description

Complete the code in the editor below to implement the following:

1. Create classes named WagonR , HondaCity , and InnovaCrysta that all inherit from the Car class.

2. Each class must have a constructor that receives one integer argument representing the mileage of

the car.

3. Each class must implement a getMileage() method which returns a string in the form of ' <mileage>

kmpl' where <mileage> is the value provided to the constructor.

Constraints

0 ≤ type of car ≤ 2

5 ≤ mileage ≤ 30

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer that describes the type of car to instantiate.

The second line contains an integer, the mileage of the car.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN Function

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

0 → type of car to instantiate = 0 (WagonR)

22 → mileage = 22

Sample Output

A WagonR is not Sedan, is 4-seater, and has a mileage of around 22 kmpl.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN Function

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

1 → type of car to instantiate = 1 (HondaCity)

12 → mileage = 12

13 of 16

Sample Output

A HondaCity is Sedan, is 4-seater, and has a mileage of around 12 kmpl.

import java.io.\*;

abstract class Car {

protected boolean isSedan;

protected String seats;

public Car(boolean isSedan, String seats) {

this.isSedan = isSedan;

this.seats = seats;

}

public boolean getIsSedan() {

return this.isSedan;

}

public String getSeats() {

return this.seats;

}

abstract public String getMileage();

public void printCar(String name) {

System.out.println(

"A " + name + " is " + (this.getIsSedan() ? "" : "not ")

+ "Sedan, is " + this.getSeats() + "-seater, and has a mileage of around "

+ this.getMileage() + ".");

}

}

**// Write your code here.**

**/\*\***

**\* WagonR class**

**\*\*/**

**class WagonR extends Car {**

**int mileage;**

**WagonR(int mileage) {CA**

**super(false, "4");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return this.mileage + " kmpl";**

**}**

**}**

**/\*\***

**\* HondaCity class**

**\*\*/**

**class HondaCity extends Car {**

**int mileage;**

**HondaCity(int mileage) {**

**super(true, "4");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return mileage + " kmpl";**

**}**

**}**

**/\*\***

**\* InnovaCrysta class**

**\*\*/**

**class InnovaCrysta extends Car {**

**int mileage;**

**InnovaCrysta(int mileage) {**

**super(false, "6");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return mileage + " kmpl";**

**}**

**}**

public class Solution {

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

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

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

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

if (carType == 0) {

Car wagonR = new WagonR(carMileage);

wagonR.printCar("WagonR");

}

if (carType == 1) {

Car hondaCity = new HondaCity(carMileage);

hondaCity.printCar("HondaCity");

}

if (carType == 2) {

Car innovaCrysta = new InnovaCrysta(carMileage);

innovaCrysta.printCar("InnovaCrysta");

}

}

**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: Movie Library**

Question description

A movie library application is being developed. This app has methods to maintain movie listings, with

functionality to add movies, remove movies, and search for movies.

Create a new class called Film and implement the IFilm interface.

Inside the Film class, define the following properties:

title (String): the title of the film

director (String): the director of the film

year (int): the year the film was released

Create another class called FilmLibrary and implement the IFilmLibrary interface.

Inside the FilmLibrary class, declare a private field called films of type List<IFilm> to store the films.

Add the following methods to the FilmLibrary class:

addFilm(IFilm film): adds a film to the film library. It takes an IFilm object as a parameter and

adds it to the films list

removeFilm(String title): removes a film from the film library based on its title if it is in the films

getFilms(): returns a list of all films in the film library

searchFilms(String query): searches for films in the film library based on a query string. It returns

a list of films whose title or director contains the query

getTotalFilmCount(): returns the total number of films in the film library

Example

There are 2 Film objects, with Title, Director, and Year.

HarryPotter DavidYates 2007

TheLordOfTheRings PeterJackson 2001

Add them to the list and determine the word to search from the movie list.

DavidYates

Finally, select the movie to be removed from the movie list.

TheLordOfTheRings

Output:

Total Film Count: 2

Search Results for DavidYates:

HarryPotter (DavidYates, 2007)

Removed Film: TheLordOfTheRings (PeterJackson, 2001)

All Films:

HarryPotter (DavidYates, 2007)

INPUT FORMAT FOR CUSTOM TESTING

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

Each of the next n lines contains the film information separated by space (Title Director Year.).

The next line contains a string, the Director to search.

The next line contains a string, the Title to delete.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN Function

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

4 number of films n = 4

Film-1 Director-2 2004 first film Title = 'Film-1', Director = 'Director-2', 'Year' = 2004

Film-2 Director-1 2018 second film...

Film-3 Director-1 2001

Film-4 Director-3 2017

Director-1 Title or Director to search for = 'Director-1'

Film-1 Title to delete = 'Film-1'

Sample Output

Total Film Count: 4

Search Results for Director-1:

Film-2 (Director-1, 2018)

Film-3 (Director-1, 2001)

Removed Film: Film-1 (Director-2, 2004)

All Films:

Film-2 (Director-1, 2018)

Film-3 (Director-1, 2001)

Film-4 (Director-3, 2017)

Explanation

There are 4 films to add. The code stub reads the data, makes the method calls, and generates results.

SAMPLE CASE 2

Sample Input For Custom Testing

4 Film-1

Director-4

2014

Film-2 Director-3 2016

Film-3 Director-4 2012

Film-4 Director-3 2003

Director-3

Film-4

Sample Output

Total Film Count: 4

Search Results for Director-3:

Film-2 (Director-3, 2016)

Film-4 (Director-3, 2003)

Removed Film: Film-4 (Director-3, 2003)

All Films:

Film-1 (Director-4, 2014)

Film-2 (Director-3, 2016)

Film-3 (Director-4, 2012)

Explanation

There are 4 films to add. The code stub reads the data, makes the method calls, and generates results.

: Java 17

import java.io.\*;

import java.util.\*;

interface IFilm {

void setTitle(String title);

String getTitle();

void setDirector(String director);

String getDirector();

void setYear(int year);

int getYear();

}

interface IFilmLibrary {

void addFilm(IFilm film);

void removeFilm(String title);

List<IFilm> getFilms();

List<IFilm> searchFilms(String query);

int getTotalFilmCount();

}

//Write The Code Here

class Film implements IFilm {

private String title;

private String director;

private int year;

@Override

public String toString(){

return title + "(" + director + "," + year + ")";

}

@Override

public void setTitle(String title){

this.title = title;

}

@Override

public void setDirector(String director) {

this.director = director;

}

@Override

public void setYear(int year) {

this.year = year;

}

@Override

public String getTitle() {

return title;

}

@Override

public String getDirector() {

return director;

}

@Override

public int getYear() {

return year;

}

}

class FilmLibrary implements IFilmLibrary {

private List<IFilm> films;

public FilmLibrary() {

this.films = new ArrayList<>();

}

@Override

public void addFilm(IFilm film) {

films.add(film);

}

@Override

public void removeFilm(String title) {

films.removeIf(film -> film.getTitle().equalsIgnoreCase(title));

}

@Override

public List<IFilm> searchFilms(String query) {

List<IFilm> result = new ArrayList<>();

for (IFilm film : films) {

if (film.getTitle().contains(query) || film.getDirector().contains(query)) {

result.add(film);

}

}

return result;

}

@Override

public List<IFilm> getFilms() {

return films;

}

@Override

public int getTotalFilmCount() {

return (int) films.stream().count();

}

}

public class Main {

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

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

PrintWriter out = new PrintWriter(System.out);

IFilmLibrary filmLibrary = new FilmLibrary();

List<IFilm> films = new ArrayList<>();

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

for (int i = 0; i < fCount; i++) {

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

IFilm e = new Film();

e.setTitle(a[0]);

e.setDirector(a[1]);

e.setYear(Integer.parseInt(a[2]));

filmLibrary.addFilm(e);

films.add(e);

}

int totalFilmCount = filmLibrary.getTotalFilmCount();

out.println("Total Film Count: " + totalFilmCount);

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

String query = b[0];

List<IFilm> searchResults = filmLibrary.searchFilms(query);

out.println("Search Results for " + query + ":");

for (IFilm film : searchResults) {

out.println(film.getTitle() + " (" + film.getDirector() + ", " + film.getYear() + ")");

}

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

String title = c[0];

IFilm randomFilm = null;

for (IFilm film : films) {

if (film.getTitle().equals(title)) {

randomFilm = film;

break;

}

}

if (randomFilm != null) {

filmLibrary.removeFilm(randomFilm.getTitle());

out.println("Removed Film: " + randomFilm.getTitle() + " (" + randomFilm.getDirector() + ", " + randomFilm.getYear() + ")");

}

List<IFilm> allFilms = filmLibrary.getFilms();

out.println("All Films:");

for (IFilm film : allFilms) {

out.println(film.getTitle() + " (" + film.getDirector() + ", " + film.getYear() + ")");

}

out.flush();

out.close();

}

}

**Java: Grocery Receipt**

Coding Abstract Class Easy OOPS Java

Question description

A software development team for an e-commerce company is working on billing software. They are

given some product prices, followed by discounts and the number of items purchased by a user.

Given a list of unique items with their prices, a list of unique items with their discounts, and a list of

items purchased by the user, generate the final invoice. The format for each item is item, price,

total\_price (total\_price is calculated as sum\_of\_quantity \* price\_of\_item, and if a discount exists, it's

subtracted from it). There might be duplicates in the list of purchases.

This invoice list should be sorted in ascending order of product name.

GroceryReceipt class:

Create the 'GroceryReceipt' class that extends GroceryReceiptBase

These properties will be passed.

Prices - the list of items and their prices

Discounts - the list of discounts for each item

Example

It is given that Bananas, Apples, and Oranges are priced at 10, 20, and 5, respectively. The discount on an

Orange is 10%. The customer shopping list is as follows.

item quantity

Banana 5

Orange 2

Orange 1

The final invoice is:

item price total\_price

Banana 10 50

Orange 5 13.5

Normally, an Orange costs 5 per unit, but there is a 10% discount. For 3 units of Orange, the gross price

is 15, less 10% is 13.5.

Function Description

Create GroceryReceipt class by extending the GroceryReceiptBase abstract class. Implement a Calculate

function to create invoices. The function should return the item, quantity, and the total price after the

discount for every item in the grocery receipt.

INPUT FORMAT FOR CUSTOM TESTING

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

Each of the next n lines contains two space-separated values, (fruit (String), price per unit (Double)).

The next line contains an integer m, the number of discount items.

Each of the next m lines contains two space-separated values, (fruit (String), percentage discount

(Integer)).

The next line contains an integer k, the number of items purchased.

Each of the next k lines contains two space-separated values, (fruit (String), units purchased (Integer)).

SAMPLE CASE 0

Sample Input

STDIN FUNCTION

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

3 → n = 3

Apple 34 → [fruit,price per unit] = [["Apple", 34],

Banana 14 ["Banana", 14],

Orange 4 ["Orange", 4]]

1 → m = 1

Orange 10 → [fruit,percentage discount] = [["Orange", 10]]

2 → k = 2

Apple 2 → [fruit, units] = [["Apple", 2], ["Apple", 5]]

Apple 5

Sample Output

Apple 34.0 238.0

Explanation

The shopping list is 2 units Apple and 5 units Apple. 7 units of Apple cost 7 \* 34 = 238 and there is no

discount.

SAMPLE CASE 1

Sample Input

STDIN FUNCTION

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

3 → n = 3

Apple 31 → [fruit,price per unit] = [["Apple", 31],

Banana 39 ["Banana", 39],

Orange 47 ["Orange", 47]]

3 → m = 3

Apple 40 → [fruit,percentage discount] = [["Apple", 40],

Banana 40 ["Banana", 40],

Orange 50 ["Orange", 50]]

2 → k = 2

Banana 4 → [fruit, units] = [["Banana", 4], ["Apple", 3]]

Apple 3

Sample Output

Apple 31.0 55.8

Banana 39.0 93.6

Explanation

3 units of Apples and 4 units of Bananas are purchased. There is a 40% discount on both items.

Apple: 3 \* 31 = 93, less 40% is 55.8

Banana: 4 \* 39 = 156, less 40% is 93.6

class Grocery {

String fruit;

double price, total;

Grocery(String fruit, double price, double total) {

this.fruit = fruit;

this.price = price;

this.total = total;

}

}

class Node {

String fruit;

int count;

Node(String fruit, int count) {

this.fruit = fruit;

this.count = count;

}

}

abstract class GroceryReceiptBase {

private Map<String, Double> prices;

private Map<String, Integer> discounts;

public GroceryReceiptBase(Map<String, Double> prices, Map<String, Integer> discounts) {

this.prices = prices;

this.discounts = discounts;

}

public abstract List<Grocery> Calculate(List<Node> shoppingList);

public Map<String, Double> getPrices() {

return prices;

}

public Map<String, Integer> getDiscounts() {

return discounts;

}

}

// Create the 'GroceryReceipt' class that extends GroceryReceiptBase above.

**class GroceryReceipt extends GroceryReceiptBase {**

**public GroceryReceipt(Map<String, Double> prices, Map<String, Integer> discounts) {**

**super(prices, discounts);**

**}**

**@Override**

**public List<Grocery> Calculate(List<Node> shoppingList) {**

**Map<String, Integer> list = new TreeMap<>();**

**for (Node s : shoppingList) {**

**String fruit = s.fruit;**

**int sum = s.count;**

**if (!list.containsKey(fruit)) {**

**list.put(fruit, sum);**

**} else {**

**list.put(fruit, list.get(fruit) + sum);**

**}**

**}**

**List<Grocery> res = new ArrayList<>();**

**for (Map.Entry<String, Integer> entry : list.entrySet()) {**

**String fruit = entry.getKey();**

**int sum = entry.getValue();**

**double price = getPrices().get(fruit);**

**double total = 0;**

**if (getDiscounts().containsKey(fruit)) {**

**total = sum \* price \* (1 - (double) getDiscounts().get(fruit) / 100);**

**} else {**

**total = sum \* price;**

**}**

**res.add(new Grocery(fruit, price, total));**

**}**

**return res;**

**}**

**}**

class Solution {

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

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

PrintWriter writer = new PrintWriter(System.out);

List<Node> boughtItems = new ArrayList<>();

Map<String, Double> prices = new HashMap<>();

Map<String, Integer> discounts = new HashMap<>();

int n = Integer.parseInt(reader.readLine().trim());

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

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

prices.put(a[0], Double.parseDouble(a[1]));

}

int m = Integer.parseInt(reader.readLine().trim());

for (int i = 0; i < m; i++) {

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

discounts.put(a[0], Integer.parseInt(a[1]));

}

int b = Integer.parseInt(reader.readLine().trim());

for (int i = 0; i < b; i++) {

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

boughtItems.add(new Node(a[0], Integer.parseInt(a[1])));

}

GroceryReceipt g = new GroceryReceipt(prices, discounts);

List<Grocery> result = g.Calculate(boughtItems);

for (Grocery x : result) {

writer.printf("%s %.1f %.1f\n", x.fruit, x.price, x.total);

}

writer.flush();

writer.close();

}

**Java Basics: Car Building**

Coding Easy Inheritance Java OOPS

Question description

Implement the following classes:

1. Class Car containing the following properties:

\* public int no\_of\_tires = 4 (Default value)

\* public String bodyType = "Plastic"

\* public void method reverseGear() => Prints ("Reverse Gear is Applied...")

\* public void method switchOnHeadlights() => Prints ("Headlights turned on...")

2. Class BMW, which extends class car and has the following properties:

\* public String modelName = "X3"

\* public void topSpeed() => Prints ("TopSpeed of BMW is 200 kmph")

Once submitted, a hidden Solution class will check the implementation by calling appropriate methods.

SAMPLE CASE 0

Sample Input

reverseGear

switchOnHeadlights

topSpeed

Sample Output

Reverse Gear is Applied...

Headlights turned on...

TopSpeed of BMW is 200 kmph

Explanation

Methods are called in the following order:

reverseGear()

switchOnHeadlights()

topSpeed()

Interviewer guidelines

11 of 12

: Java 17

//Implement the Code here

**class Car {**

**public int no\_of\_tires = 4;**

**public String bodyType = "Plastic";**

**public void reverseGear() {**

**System.out.println("Reverse Gear is Applied...");**

**}**

**public void switchOnHeadlights() {**

**System.out.println("Headlights turned on...");**

**}**

**}**

**class BMW extends Car {**

**public String modelName = "X3";**

**public void topSpeed() {**

**System.out.println("TopSpeed of BMW is 200 kmph");**

**}**

**}**

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

for (int i = 0; i < 3; i++) {

String name = sc.nextLine();

BMW b = new BMW();

if (name.equals("reverseGear")) {

b.reverseGear();

}

if (name.equals("switchOnHeadlights")) {

b.switchOnHeadlights();

}

if (name.equals("topSpeed")) {

b.topSpeed();

}

}

}

}

**Java Basics: Library Structure**

Question description

Implement the Library structure using the following class.

1. Library class with the following properties

\* Attributes

\* private integer attribute number\_of\_books

\* private string attribute name (library name)

\* private Map<String, Integer> bookGenres, map of genres to book count

\* Public getter and setter methods

----- Getter----

\* int getNumber\_of\_books() method

\* string getName() method

\* Map<String, Integer> getBookGenres() method

----- Setter ------

\* void setNumber\_of\_books(int number\_of\_books) method

\* void setName(string name) method

\* void setBookGenres(Map<String, Integer> bookGenres) method

10 of 15

Note: The code stub receives input and passes it to the methods.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, that denotes the number of library instances to be initiated.

Each of the n instances of libraries contains lines in the following format:

NAME

NUMBER\_OF\_BOOKS

BOOK\_GENRES\_COUNT

Each of the subsequent BOOK\_GENRES\_COUNT lines contains

BOOK\_GENRE COUNT

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN Function

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

1 → library instances n = 1

National Library → NAME = 'National Library'

20 → NUMBER\_OF\_BOOKS = 20

2 → BOOKS\_GENRES\_COUNT = 2

Drama 5 → BOOK\_GENRE = 'Drama' books in genre = 2

Fiction 15 → BOOK\_GENRE = 'Fiction', books in genre = 15

Sample Output

Library with name : National Library is started

It contains 20 books

It contains books with following genres :

Drama 5

Fiction 15

Explanation

The sample output shows an implementation of the Library class.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN Function

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

11 of 15

2 → library instances n = 2

National Library → NAME = 'National Library'

20 → NUMBER\_OF\_BOOKS = 20

2 → BOOKS\_GENRES\_COUNT = 2

Drama 5 → BOOK\_GENRE = 'Drama' books in genre = 5

Fiction 15 → BOOK\_GENRE = 'Fiction', books in genre = 15

Delhi Library → NAME = 'Delhi Library'

5 → NUMBER\_OF\_BOOKS = 5

1 → BOOKS\_GENRES\_COUNT = 1

Mythology 5 → BOOK\_GENRE = 'Mythology' books in genre = 5

Sample Output

Library with name : National Library is started

It contains 20 books

It contains books with following genres :

Drama 5

Fiction 15

Library with name : Delhi Library is started

It contains 5 books

It contains books with following genres :

Mythology 5

Explanation

The sample output shows an implementation of the Library class.

Interviewer guidelines

: Java 17

class Library {

private int number\_of\_books;

private String name;

private Map<String, Integer> bookGenres;

public void setNumber\_of\_books(int number\_of\_books) {

this.number\_of\_books = number\_of\_books;

}

public int getNumber\_of\_books() {

return number\_of\_books;

}

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setBookGenres(Map<String, Integer> bookGenres) {

this.bookGenres = bookGenres;

}

public Map<String, Integer> getBookGenres() {

return bookGenres;

}

}

class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String sub = sc.nextLine();

int n = Integer.parseInt(sub);

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

String name = sc.nextLine();

Library library = new Library();

library.setName(name);

System.out.println("Library with name : " + library.getName() + " is started");

Integer numberOfBooks = Integer.parseInt(sc.nextLine());

library.setNumber\_of\_books(numberOfBooks);

System.out.println("It contains " + library.getNumber\_of\_books() + " books");

Integer bookGenresCount = Integer.parseInt(sc.nextLine());

Map<String, Integer> bookGenresMap = new LinkedHashMap<>();

System.out.println("It contains books with following genres :");

for (int j = 0; j < bookGenresCount; j++) {

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

bookGenresMap.put(bookGenres[0], Integer.parseInt(bookGenres[1]));

}

library.setBookGenres(bookGenresMap);

Map<String, Integer> temp = library.getBookGenres();

for (Map.Entry<String, Integer> t : temp.entrySet()) {

System.out.println(t.getKey() + " " + t.getValue());

}

}

}

}

**JavaBasics: Points and Lines**

Question description

Given two fully implemented classes, Point and Line, read through the given code to understand their

implementations. Implement the class LineList which implements the interface ListOfLines in order to

perform some operations on a list of lines. Interface ListOfLines consists of:

Attribute `Vector<Line> list\_of\_lines` which stores a list of lines.

Constructor with parameter Vector<Line> to initialize the list\_of\_lines attribute.

Method: `Line getLineWithMaxLength()` which returns the first longest line from the list of lines.

Method: `Vector<Line> getLinesStartingFromPoint(Point p)` which returns a list of all the lines

starting from the given point p.

Note: The code stub reads input and calls methods.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, denoting the number of lines. Next, n lines are the coordinates for

each line in the format [x[1] y[1] x[2] y[2]], where (x[1], y[1]) is the starting point and (x[2], y[2]) is the

endpoint of the line. The last line is the starting point for comparison. Format: [x y]

SAMPLE CASE 0

Sample Input

STDIN Function

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

4 number of lines n = 4

3 8 13 14 (x1, y1, x2, y2)[0] = 3, 8, 13, 14

8 4 3 12 (x1, y1, x2, y2)[1] = 8, 4, 3, 12

-7 -4 17 6 (x1, y1, x2, y2)[2] = -7, -4, 17, 6

7 3 1 2 (x1, y1, x2, y2)[3] = 7, 3, 1, 2

8 4 starting point (x, y) = 8, 4

Sample Output

Longest Line --> Start: (-7.0, -4.0) End: (17.0, 6.0)

Length: 26.00

All the Lines starting from point: (8.0, 4.0)

Start: (8.0, 4.0) End: (3.0, 12.0)

Explanation

4 is the total number of lines and (8, 4) is the point to find all the lines starting from this point.

SAMPLE CASE 1

Sample Input

STDIN Function

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

5 number of lines n = 5

23 15 -4 8 (x1, y1, x2, y2)[0] = 23, 15, -4, 8

7 3 2 1 (x1, y1, x2, y2)[1] = 7, 3, 2, 1

-5 4 13 8 (x1, y1, x2, y2)[2] = -5, 4, 13, 8

7 3 19 -12 (x1, y1, x2, y2)[3] = 7, 3, 19, -12

3 8 1 0 (x1, y1, x2, y2)[4] = 3, 8, 1, 0

7 3 starting point (x, y) = 7, 3

Sample Output

Longest Line --> Start: (23.0, 15.0) End: (-4.0, 8.0)

Length: 27.89

All the Lines starting from point: (7.0, 3.0)

Start: (7.0, 3.0) End: (2.0, 1.0)

Start: (7.0, 3.0) End: (19.0, -12.0)

Explanation

5 is the total number of lines and (7, 3) is the point to find all the lines starting from this point.

: Java 17

import java.util.\*;

class Point {

private double x;

private double y;

Point(double x, double y) {

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public void setX(double x) {

this.x = x;

}

public void setY(double y) {

this.y = y;

}

public static double distance(Point p1, Point p2) {

double distance = Math.sqrt(

Math.pow(p2.getY() - p1.getY(), 2) +

Math.pow(p2.getX() - p1.getX(), 2)

);

return distance;

}

public String toString() {

return "(" + this.x + ", " + this.y + ")";

}

}

class Line {

private Point start = new Point(0, 0);

private Point end = new Point(0, 0);

Line(Point start, Point end) {

this.start = start;

this.end = end;

}

public Point getStart() {

return start;

}

public Point getEnd() {

return end;

}

public void setStart(Point start) {

this.start = start;

}

public void setEnd(Point end) {

this.end = end;

}

public double getLineLength() {

return Point.distance(start, end);

}

public String toString() {

return "Start: " + this.start + " End: " + this.end;

}

}

interface ListOfLines {

Vector<Line> getLinesFromStartingPoint(Point p);

Line getLineWithMaxLength();

Vector<Line> list\_of\_lines = null;

}

//Implement The Code

class LineList implements ListOfLines {

Vector<Line> lines;

public LineList(Vector<Line> lines) {

this.lines = lines;

}

public Vector<Line> getLinesFromStartingPoint(Point p) {

Vector<Line> res = new Vector<>();

for (int i = 0; i < lines.size(); i++) {

Line curlin = lines.elementAt(i);

if (test(curlin.getStart(), p))

res.add(curlin);

}

return res;

}

private boolean test(Point a, Point b) {

return a.getX() == b.getX() && a.getY() == b.getY();

}

public Line getLineWithMaxLength() {

Line maxl = lines.elementAt(0);

for (int i = 1; i < lines.size(); i++) {

if (lines.elementAt(i).getLineLength() > maxl.getLineLength())

maxl = lines.elementAt(i);

}

return maxl;

}

}

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] input;

String sub = sc.nextLine();

int nLines = Integer.parseInt(sub);

Vector<Line> lines = new Vector<Line>();

for (int i = 0; i < nLines; i++) {

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

double x1 = Double.parseDouble(input[0]);

double y1 = Double.parseDouble(input[1]);

double x2 = Double.parseDouble(input[2]);

double y2 = Double.parseDouble(input[3]);

Point p1 = new Point(x1, y1);

Point p2 = new Point(x2, y2);

Line line = new Line(p1, p2);

lines.add(line);

}

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

double point\_x1 = Double.parseDouble(input[0]);

double point\_y1 = Double.parseDouble(input[1]);

Point startingPoint = new Point(point\_x1, point\_y1);

LineList myList = new LineList(lines);

Line maxLine = myList.getLineWithMaxLength();

System.out.println("Longest Line --> " + maxLine);

double length = maxLine.getLineLength();

DecimalFormat df = new DecimalFormat("#.00");

System.out.println("Length: " + df.format(length));

Vector<Line> starting\_lines = myList.getLinesFromStartingPoint(startingPoint);

System.out.println("All the Lines starting from point: " + startingPoint);

Iterator<Line> value = starting\_lines.iterator();

while (value.hasNext()) {

System.out.println(value.next());

}

}

}

**JavaBasics: Edge**

Question description

Design a class Edge to represent an edge of a directed graph. The class should override the following

functions such that any two edge objects can be compared by value.

Implement class Edge:

public static Edge createEdge(int fromNode, int toNode): a factory method to create an instance of

the class.

public boolean equals(Object object): an overridden function to check if the current and passed

objects' attribute values are the same.

public int compareTo(Edge edge): an overridden function

public int hashCode(): an overridden function to compute hash code for an object to support storing

in hash tables.

public String toString(): an overridden function to support printing objects in the format "From node:

<this.fromNode> To node: <this.toNode>"

The code stub handles reading input and passes it to appropriate functions. There are numberOfQueries

edges. Each of the next numberOfQueries lines contains two space-separated integers, fromNode, and

toNode, the ends of an edge.

Constraints

1 ≤ numberOfQueries ≤ 10

1 ≤ fromNode, toNode ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, numberOfQueries, the number of edges.

Each i of the next numberOfQueries lines contains 2 space-separated integers, fromNode, and toNode

respectively.

5

5

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN Function

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

4 numberOfQueries = 4

1 3 first query, fromNode = 1, toNode = 3

1 2

2 1

1 3

Sample Output

3 From node: 1

To node: 3

From node: 1 To node: 2

From node: 2 To node: 1

Explanation

After processing all the edges, there are 3 edges.

From node: 1 To node: 3

From node: 1 To node: 2

From node: 2 To node: 1

The edge (1, 3) is duplicated but does not appear twice in the output.

Interviewer guidelines

SOLUTION

: Java 7

import java.util.\*;

import java.io.\*;

//Write The Code here

**class Edge {**

**public final int fromNode;**

**public final int toNode;**

**private Edge(int fromNode, int toNode) {**

**this.fromNode = fromNode;**

**this.toNode = toNode;**

**}**

**@Override**

**public boolean equals(Object o) {**

**if (this == o) {**

**return true;**

**}**

**Edge edge = (Edge) o;**

**return this.fromNode == edge.fromNode && this.toNode == edge.toNode;**

**}**

**@Override**

**public int hashCode() {**

**return (int) (fromNode \* 100000L \* toNode) % 1000000007;**

**}**

**@Override**

**public String toString() {**

**return ("From node: " + this.fromNode + " To node: " + this.toNode);**

**}**

**public static Edge createEdge(int fromNode, int toNode) {**

**return new Edge(fromNode, toNode);**

**}**

**}**

public class Solution {

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

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

PrintWriter out = new PrintWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String[] arr;

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

StringBuilder sb = new StringBuilder();

HashSet<Edge> set = new HashSet<>();

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

arr = br.readLine().trim().split(" ");

int u = Integer.parseInt(arr[0]);

int v = Integer.parseInt(arr[1]);

Edge edge = Edge.createEdge(u, v);

if (set.contains(edge)) {

continue;

}

set.add(edge);

sb.append(edge.toString()).append("\n");

}

out.println(set.size());

out.println(sb);

out.flush();

out.close();

}

}

**Java: Data Encryption**

Question description

You are required to customize a class named DataEncryption, having the following attributes:

encryptionRatio, baseEncryptionValue, and a method titled encryptValue().

The default encryptValue() function works by calculating the encrypted value as follows:

encryptedValue = baseEncryptionValue + encryptionRatio

Add more functionality to the existing method encryptValue() so that it accepts a variable named

encryptionAdjustment of integer data type. The function should accordingly adjust the calculation of the

encrypted value as:

encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment

Additionally, you are asked to overload this method so that it accepts encryptionAdjustment as a string

type, converts it to an integer, and further uses it in the calculation of the encrypted value.

encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment

There are three overloaded versions of encryptValue() in the DataEncryption class:

1. Default Version: Takes no parameters and calculates the encrypted value as baseEncryptionValue +

encryptionRatio .

2. First Overloaded Version: Accepts an integer encryptionAdjustment and calculates the encrypted

value as (baseEncryptionValue + encryptionRatio) - encryptionAdjustment .

3. Second Overloaded Version: Accepts a string encryptionAdjustment , converts it to an integer, and

calculates the encrypted value as (baseEncryptionValue + encryptionRatio) - encryptionAdjustment .

Example

encryptionRatio = 50

baseEncryptionValue = 200

encryptionAdjustment = 14

11 of 15

Default, encryptedValue = 250

With the first method modification, encryptedValue = 236

With the second method modification, encryptedValue = 236

INPUT FORMAT FOR CUSTOM TESTING

A single line of input consists of space-separated integers: encryptionRatio, baseEncryptionValue,

and encryptionAdjustment.

SAMPLE CASE 0

Sample Input For Custom Testing

0 0 0

Sample Output

0 0 0

SAMPLE CASE 1

Sample Input For Custom Testing

10 12 4

Sample Output

22 18 18

Interviewer guidelines

: Java 8

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

class DataEncryption {

private int encryptionRatio;

private int baseEncryptionValue;

public DataEncryption(int encryptionRatio, int baseEncryptionValue) {

this.encryptionRatio = encryptionRatio;

this.baseEncryptionValue = baseEncryptionValue;

}

public int encryptValue() {

int encryptedValue = baseEncryptionValue + encryptionRatio;

return encryptedValue;

}

public int encryptValue(int encryptionAdjustment) {

int encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment;

return encryptedValue;

}

public int encryptValue(String encryptionAdjustment) {

int a = Integer.parseInt(encryptionAdjustment);

int encryptedValue = (baseEncryptionValue + encryptionRatio) - a;

return encryptedValue;

}

}

public class Solution {

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

// reader and writer

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

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

String[] values = bufferedReader.readLine().split(" ");

// check if int array

try {

Integer.parseInt(values[0]);

// System.out.println("Integer array");

Integer[] ia = new Integer[values.length];

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

ia[i] = new Integer(values[i]);

}

DataEncryption dataEncryption = new DataEncryption(ia[0], ia[1]);

bufferedWriter.write(

dataEncryption.encryptValue() + " "

+ dataEncryption.encryptValue(ia[2]) + " "

+ dataEncryption.encryptValue(String.valueOf(ia[2]))

);

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

} catch (NumberFormatException nfe) {

// then string array

// System.out.println("String array");

System.out.println();

}

}

}

**Java: Employee Profile**

Question description

Implement the following classes:

1. abstract class Employee with the following methods:

\* abstract void setSalary(int salary) method

\* abstract int getSalary() method

\* abstract void setGrade(String grade) method (grade of the employee in the organization)

\* abstract String getGrade() method

\* void label() method which prints "Employee's data:\n" (Concrete method, implementation is hidden

from end user)

2. class Engineer extending class Employee:

\* private attribute int salary

\* private attribute String grade

\* implement the setter and getter methods from the parent class to set and get attributes (salary and

grade)

3. class Manager extending class Employee:

\* private attribute int salary

\* private attribute String grade

\* implement the setter and getter methods from the parent class to set and get attributes (salary and

grade)

Note: The code stub handles input and calls the methods.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, that denotes the number of employees to be instantiated.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains 3 variables.

TYPE\_OF\_EMPLOYEE GRADE SALARY

SAMPLE CASE 0

Sample Input For Custom Testing

2 ENGINEER B

50000

MANAGER A 70000

Sample Output

Employee's data:

GRADE : B

SALARY : 50000

Employee's data:

GRADE : A

SALARY : 70000

SAMPLE CASE 1

Sample Input For Custom Testing

3 ENGINEER B

50000

MANAGER A 70000

MANAGER A 90000

Sample Output

Employee's data:

GRADE : B

SALARY : 50000

Employee's data:

GRADE : A

SALARY : 70000

Employee's data:

GRADE : A

SALARY : 90000

Interviewer guidelines

: Java 8

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

//Write The Code Here

**abstract class Employee {**

**private int salary;**

**private String grade;**

**public abstract void setSalary(int salary);**

**public abstract int getSalary();**

**public abstract void setGrade(String grade);**

**public abstract String getGrade();**

**public void label() {**

**System.out.println("Employee's data:");**

**}**

**}**

**class Engineer extends Employee {**

**private int salary;**

**private String grade;**

**public void setSalary(int salary) {**

**this.salary = salary;**

**}**

**public int getSalary() {**

**return salary;**

**}**

**public void setGrade(String grade) {**

**this.grade = grade;**

**}**

**public String getGrade() {**

**return grade;**

**}**

**}**

**class Manager extends Employee {**

**private int salary;**

**private String grade;**

**public void setSalary(int salary) {**

**this.salary = salary;**

**}**

**public int getSalary() {**

**return salary;**

**}**

**public void setGrade(String grade) {**

**this.grade = grade;**

**}**

**public String getGrade() {**

**return grade;**

**}**

**}**

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String sub = sc.nextLine();

int n = Integer.parseInt(sub);

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

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

if (input[0].equals("ENGINEER")) {

Engineer e = new Engineer();

e.setSalary(Integer.parseInt(input[2]));

e.setGrade(input[1]);

e.label();

System.out.println("GRADE : " + e.getGrade());

System.out.println("SALARY : " + e.getSalary());

}

if (input[0].equals("MANAGER")) {

Manager e = new Manager();

e.setSalary(Integer.parseInt(input[2]));

e.setGrade(input[1]);

e.label();

System.out.println("GRADE : " + e.getGrade());

System.out.println("SALARY : " + e.getSalary());

}

}

}

}

**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());

}

}

}

**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);**

**}**

**}**

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 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

4

1 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();

}

}

**String Comparison**

Coding Strings Easy Implementation

Question description

Given a pair of strings. The goal is to make them equal by doing one of the following operations at most

once:

Select two characters, char1 and char2, and replace all occurrences of char1 in the first string with

char2.

Select two characters, char1 and char2, and replace all occurrences of char1 in the second string with

char2.

For example, the string "xxyz" can be converted to any of the following strings by performing the

operation as mentioned above: "yyyz", "vvyz", "xxzz", "xxwz", etc. However, it cannot be converted to

the string "yzyz" (Since all the occurrences of a character can be replaced by a single new character).

Note that doing both operations simultaneously is not allowed. For example, given two strings "xz" and

"zx", we can not convert both the x's in the respective strings to 'z' to make them equal.

Given two arrays of n strings each, compare the strings present at the same indices in both the arrays

and find if they can be made equal or not. Return an array of n strings, where the i element would be

YES if the strings at index i can be equalized; otherwise, the answer is NO.

Example

Consider the array of strings to be:

firstStrings = ["aaa", "abbc", "zyz"]

secondStrings = ["bbb", "cccc", "zyx"]

th

4 of 21

Let us compare the strings at each index:

i = 0: firstStrings[0] = "aaa", secondStrings[0] = "bbb"

We can change all the occurrences of 'a' to 'b' in the first string in this case. Therefore, the answer is

"YES".

i = 1: firstStrings[1] = "abbc", secondStrings[1] = "cccc"

There is no way in which we can make these strings identical. Therefore, the answer is "NO".

i = 2: firstStrings[2] = "zyz", secondStrings[2] = "zyx"

We can replace the only occurrence of 'x' in the second string with 'z' to make them identical.

Therefore, the answer is "YES".

Hence, the answer corresponding to each pair of strings is ["YES," "NO," "YES"].

Function Description

Complete the function canBeEqualized in the editor below.

canBeEqualized has the following parameters:

firstStrings[firstStrings[0],...firstStrings[n-1]]: an array of strings representing the first string of each

pair

secondStrings[secondStrings[0],...secondStrings[n-1]]: an array of strings representing the second

string of each pair

Returns

string[]: an array of strings representing the answer to each pair of strings

Constraints

1 ≤ n ≤ 10

1 ≤ |firstStrings[i]|, |secondStrings[i]| ≤ 10

All the strings consist of lowercase English characters only.

INPUT FORMAT FOR CUSTOM TESTING

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

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

The next line contains an integer, n, denoting the number of elements in secondStrings.

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

SECOND STRINGSSAMPLE CASE 0

3

3

5 of 21

Sample Input For Custom Testing

STDIN FUNCTION

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

2 → firstStrings[] size n = 2

ac → firstStrings = ["ac", "pqqrr"]

pqqrr

2 → secondStrings[] size n = 2

ca → secondStrings = ["ca", "prrrr"]

prrrr

Sample Output

NO

YES

Explanation

We can check that there is no way to make them equal for the first pair of strings. Thus the answer

is "NO".

For the second pair of strings, we can convert all the occurrences of 'q' in the first string to 'r', thus

the answer is "YES".

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

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

2 → firstStrings[] size n = 2

a → firstStrings = ["a", "aaabbb"]

aaabbb

2 → secondStrings[] size n = 2

b → secondStrings = ["b", "abbbbb"]

abbbbb

Sample Output

YES

NO

Explanation

For the first pair of strings, we can convert the only occurrence of 'a' to 'b', thus the answer is "YES".

For the second pair of strings, it can be seen that it is impossible to make the two strings equal,

thus the answer is "NO".

6 of 21

Interviewer guidelines

SOLUTION

Topic Covered: loops and counters, arrays, strings

Optimal Solution: This problem can be solved by iterating over all n pairs of strings, and finding the

answers one by one. To verify whether two strings are similar, we need to make sure the following:

The length of both of these strings is the same.

There is at most one pair of characters that mismatch in the corresponding indices.

For this pair of mismatch, we should ensure that at least one of the strings don't have this character to

be paired up with the same character.

COMPLEXITY ANALYSIS

Time Complexity - O(n\*len) where len represents the maximum length of firstString overall given

strings.

Space Complexity - O(1) as we're using constant space.

: 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 'canBeEqualized' function below.

\*

\* The function is expected to return a STRING\_ARRAY.

\* The function accepts following parameters:

\* 1. STRING\_ARRAY firstStrings

\* 2. STRING\_ARRAY secondStrings

\*/

public static List<String> canBeEqualized(List<String> firstStrings,

List<String> secondStrings) {

**// Write your code here**

**List<String> list = new ArrayList<>();**

**int flag = 0;**

**for (int i = 0; i < firstStrings.size(); i++) {**

**if (firstStrings.get(i).length() != secondStrings.get(i).length()) {**

**list.add("NO");**

**continue;**

**}**

**for (int j = 0; j < firstStrings.get(i).length(); j++) {**

**String temp = secondStrings.get(i).replaceAll(secondStrings.get(i).charAt(j) + "", firstStrings.get(i).charAt(j) + "");**

**if (temp.equals(firstStrings.get(i))) {**

**list.add("YES");**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**for (int j = 0; j < firstStrings.get(i).length(); j++) {**

**String temp = firstStrings.get(i).replaceAll(firstStrings.get(i).charAt(j) + "", secondStrings.get(i).charAt(j) + "");**

**if (temp.equals(secondStrings.get(i))) {**

**list.add("YES");**

**flag = 1;**

**break;**

**}**

**}**

**}**

**if (flag == 0)**

**list.add("NO");**

**flag = 0;**

**}**

**return list;**

**}**

}

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 firstStringsCount = Integer.parseInt(bufferedReader.readLine().trim());

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

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

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

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

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

List<String> result = Result.canBeEqualized(firstStrings, secondStrings);

bufferedWriter.write(

result.stream()

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**1. Similar Password**

Coding Easy Strings Greedy Algorithms Sorting

Question description

A password detection system for HackerRank accounts detects a password as similar if the number of

vowels is equal to the number of consonants in the password.

Passwords consist of lowercase English characters only, and vowels are ('a', 'e', 'i', 'o', 'u').

To check the strength of a password and how easily it can be hacked, some manipulations can be made to

the password. In one operation, any character of the string can either be incremented or decremented.

For example, 'f' can be incremented to 'g', or decremented to 'e'. Note that character 'a' cannot be

decremented and 'z' cannot be incremented.

Find the minimum number of operations in which the password can be made similar.

Example

Consider password = "hack". The 'h' can be changed to 'i' in one operation. The resultant string is "iack"

which has 2 vowels ('i', 'a') and 2 consonants ('c', 'k') and hence the string is similar. Return 1, the

minimum number of operations required.

Function Description

Complete the function countMinimumOperations in the editor below.

countMinimumOperations has the following parameter:

string password: the password

Rishi Khuntia

4 of 15

Returns

int: the minimum number of operations required to make the password similar

Constraints

2 ≤ |password| ≤ 3\*10

It is guaranteed that the length of the password is even.

The given string consists of lowercase Latin characters only.

INPUT FORMAT FOR CUSTOM TESTING

The first and only line contains the given string password.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

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

abcd → password = "abcd"

Sample Output

1

Explanation

In one operation, 'd' can be changed to 'e'. The resultant string is "abce" which has an equal number of

vowels and consonants.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

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

bigbangt → the given string, password = "bigbangt"

Sample Output

2

Explanation

5

In the first operation, the first occurrence of 'b' can be changed to 'a'. In the next operation, another

occurrence of 'b' can be changed to 'a'. The resultant string is "aigaangt" which has an equal number

of vowels and consonants.

Interviewer guidelines

SOLUTION

Skills: Sorting, Strings, Greedy Algorithms

Optimal Solution:

Consider the number of vowels to be v and the number of consonants to be c. Firstly, each vowel can

be changed to a consonant in a single operation. So, if v is greater than c, then the extra (v - c) / 2

vowels can be changed in one operation each. Hence, the answer in this case is (v - c) / 2.

Otherwise, for each consonant, find the minimum number of operations needed to convert it to a

vowel, and then choose the minimum (c - v) / 2 consonants, and sum their operations to obtain the

answer.

COMPLEXITY ANALYSIS

Time Complexity - O(n log n)

Traversing each character and finding the minimum operations for each character takes O(n) time.

There can be up to n consonants, so sorting the operations array takes O(n log n) time. Hence, time

complexity is O(n log n).

Space Complexity - O(n)

The operations array can have up to n entries, so space complexity is O(n).

: Java 8

import java.util.ArrayList;

import java.util.PriorityQueue;

class Result {

/\*

\* Complete the 'countMinimumOperations' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts STRING password as parameter.

\*/

public static int countMinimumOperations(String p) {

**int n = p.length();**

**int half = n / 2, v = 0, c = 0;**

**ArrayList<Character> arr = new ArrayList<Character>();**

**arr.add('a');**

**arr.add('e');**

**arr.add('i');**

**arr.add('o');**

**arr.add('u');**

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

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

**if (arr.contains(ch)) {**

**v++;**

**} else {**

**c++;**

**}**

**}**

**if (v > c) {**

**return v - half;**

**} else if (c > v) {**

**int diff = c - half;**

**PriorityQueue<Integer> pq = new PriorityQueue<Integer>();**

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

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

**int mini = Integer.MAX\_VALUE;**

**if (!arr.contains(ch)) {**

**mini = Math.min(mini, Math.abs(ch - 'a'));**

**mini = Math.min(mini, Math.abs(ch - 'e'));**

**mini = Math.min(mini, Math.abs(ch - 'i'));**

**mini = Math.min(mini, Math.abs(ch - 'o'));**

**mini = Math.min(mini, Math.abs(ch - 'u'));**

**}**

**pq.add(mini);**

**}**

**int ans = 0;**

**while (diff > 0) {**

**int top = pq.remove();**

**ans += top;**

**diff--;**

**}**

**return ans;**

**}**

**return 0;**

**}**

}

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 password = bufferedReader.readLine();

int result = Result.countMinimumOperations(password);

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

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}