**Coding Test**

**Duration: 30 mins**

Name: \_Omkar Padmale\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Position: Java Developer\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Result: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
This test is designed to test your coding fluency and problem-solving skills. Solutions should focus on solving the given problem, producing clean, working code, and producing solutions which optimize time and space complexity. You can use any programming language and you will not be penalized for minor syntax errors.  
  
Problem 1: Given a string, write a function which returns a boolean value indicating if it is palindrome or not. A string is said to be a palindrome if the reverse of the string is the same as the string. For example, **malayalam** is a palindrome, **geek** is not.  
  
Language you are using: \_Java\_\_\_\_\_\_\_  
Runtime Complexity of your solution: \_\_\_\_\_\_O(n)\_\_\_\_  
Space Complexity of your solution: \_\_\_\_O(1)\_\_\_\_\_\_\_\_  
public class PalindromeOrNot {

public static void main(String [] args){

String str="malayalam";

int fromStart=0;

int fromEnd= str.length()-1;

Boolean IsPalindrome = true;

while(fromStart<=fromEnd){

if(str.charAt(fromStart)!=str.charAt(fromEnd)){

IsPalindrome=false;

break;

}

fromStart++;

fromEnd--;

}

if(IsPalindrome)

{

System.***out***.println("given String is Palindrome");

}else

System.***out***.println("not Palindrome");

}

}

Problem 2: Write a function which takes in a 2D List/Array of transactions and returns a list of transaction IDs which are fraudulent. Any transaction greater than or equal to 10000 is considered fraudulent. Any transaction from the same credit card in a different city within 30 minutes is considered fraudulent.  
  
Input: A 2D List/Array of transactions with each transaction record having a transaction ID (integer), credit card ID (integer, transaction amount (double), city (string), and time in minutes (integer). You can assume all transactions happen on the same day.  
  
Example:  
Input: [ [1, 1000, 500.00, “Vadodara”, 0], [2, 1000, 500.00, “Mumbai”, 5], [3, 1001, 500.00, “Mumbai”, 10], [4, 1001, 10000.00, “Mumbai”, 10]]  
Output: [2, 4]  
  
Transactions 2 and 4 should be considered fraudulent. Transaction 2 occurred within 30 min of transaction 1 with the same credit card ID (1000) and in a different city. Transaction 4 has an amount is greater than or equal to 10000.  
  
Language you are using: \_\_\_java\_\_\_\_\_

Runtime Complexity of your solution: O(n2)O(n^2)O(n2), where nnn is the number of transactions in the list\_\_\_\_\_\_\_\_\_\_

Space Complexity of your solution: \_\_O(n), where nnn is the number of transactions in the list\_\_\_\_\_\_\_\_\_\_  
Write your code below (continue on back if needed):

//I have created 2 separate classes that is TransactionaDetails and Main.

public class TransactionDetails {

public TransactionDetails() {

super();

}

public TransactionDetails(int id, int credit\_card\_Id, double transaction\_Amount, String city, int time) {

super();

Id = id;

this.credit\_card\_Id = credit\_card\_Id;

this.transaction\_Amount = transaction\_Amount;

this.city = city;

this.time = time;

}

int Id;

int credit\_card\_Id;

double transaction\_Amount;

String city;

int time;

public int getId() {

return Id;

}

public void setId(int id) {

Id = id;

}

public int getCredit\_card\_Id() {

return credit\_card\_Id;

}

public void setCredit\_card\_Id(int credit\_card\_Id) {

this.credit\_card\_Id = credit\_card\_Id;

}

public double getTransaction\_Amount() {

return transaction\_Amount;

}

public void setTransaction\_Amount(double transaction\_Amount) {

this.transaction\_Amount = transaction\_Amount;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public int getTime() {

return time;

}

public void setTime(int time) {

this.time = time;

}

}

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

public class Main {

public static void main(String[] args) {

List<TransactionDetails> list=new ArrayList<>();

list.add(new TransactionDetails(1, 1000, 500.00, "Vadodara", 0));

list.add(new TransactionDetails(2, 1000, 500.00, "Mumbai", 5));

list.add(new TransactionDetails(3, 1001, 500.00, "Mumbai", 10));

list.add(new TransactionDetails(4, 1001, 10000.00, "Mumbai", 10));

Set<Integer> fraudulentTransactionIds = new HashSet<>();

for (TransactionDetails transaction : list) {

if (transaction.getTransaction\_Amount() >= 10000) {

fraudulentTransactionIds.add(transaction.getId());

}

}

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

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

TransactionDetails t1 = list.get(i);

TransactionDetails t2 = list.get(j);

if (t1.getCredit\_card\_Id() == t2.getCredit\_card\_Id() &&

!t1.getCity().equals(t2.getCity()) &&

Math.*abs*(t1.getTime() - t2.getTime()) <= 30) {

fraudulentTransactionIds.add(t1.getId());

fraudulentTransactionIds.add(t2.getId());

}

}

}

for (Integer id : fraudulentTransactionIds) {

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

}

}

}

Problem 3: A railway system is keeping track of customer travel times between different stations. They are using this data to calculate the average time it takes to travel from one station to another.

Implement a class called **UndergroundSystem** with the following functions:

**void checkIn(int id, string stationName, int t)**A customer with a card ID equal to id, checks in at the station stationName at time t. A customer can only be checked into one place at a time.

**void checkOut(int id, string stationName, int t)**A customer with a card ID equal to id, checks out from the station stationName at time t.

**double getAverageTime(string startStation, string endStation)**Returns the average time it takes to travel from startStation to endStation.

The average time is computed from all the previous traveling times from startStation to endStation that happened directly, meaning a check in at startStation followed by a check out from endStation. The time it takes to travel from startStation to endStation may be different from the time it takes to travel from endStation to startStation. There will be at least one customer that has traveled from startStation to endStation before getAverageTime is called. You may assume all calls to the checkIn and checkOut methods are consistent. If a customer checks in at time t1 then checks out at time t2, then t1 < t2. All events happen in chronological order.  
  
Language you are using: \_\_\_java\_\_\_\_\_  
Runtime Complexity of your Solution: \_\_\_\_\_\_\_\_\_\_  
Space Complexity of your Solution: \_\_\_\_\_\_\_\_\_\_\_\_  
**Write your code below (continue on back if needed):**

**//class trip**

public class Trip {

private String startStation;

private int startTime;

public Trip(String startStation, int startTime) {

this.startStation = startStation;

this.startTime = startTime;

}

public String getStartStation() {

return startStation;

}

public int getStartTime() {

return startTime;

}

}

//class journy

public class Journey {

private int totalTime;

private int tripCount;

public Journey() {

this.totalTime = 0;

this.tripCount = 0;

}

public void addTrip(int time) {

this.totalTime += time;

this.tripCount++;

}

public double getAverageTime() {

return (double) totalTime / tripCount;

}

}

//class underGround

import java.util.HashMap;

import java.util.Map;

public class UndergroundSystem {

private Map<Integer, Trip> checkInMap;

private Map<String, Journey> journeyMap;

public UndergroundSystem() {

checkInMap = new HashMap<>();

journeyMap = new HashMap<>();

}

public void checkIn(int id, String stationName, int t) {

checkInMap.put(id, new Trip(stationName, t));

}

public void checkOut(int id, String stationName, int t) {

Trip trip = checkInMap.get(id);

if (trip != null) {

String key = trip.getStartStation() + "->" + stationName;

int travelTime = t - trip.getStartTime();

journeyMap.putIfAbsent(key, new Journey());

journeyMap.get(key).addTrip(travelTime);

checkInMap.remove(id);

}

}

public double getAverageTime(String startStation, String endStation) {

String key = startStation + "->" + endStation;

Journey journey = journeyMap.get(key);

if (journey != null) {

return journey.getAverageTime();

}

return -1.0; // Should never happen based on problem constraints

}

public static void main(String[] args) {

UndergroundSystem undergroundSystem = new UndergroundSystem();

undergroundSystem.checkIn(1, "Mumbai", 3);

undergroundSystem.checkIn(2, "Pune", 8);

undergroundSystem.checkOut(1, "Sangli", 15);

undergroundSystem.checkOut(2, "Cambridge", 22);

undergroundSystem.checkIn(3, "Mumbai", 10);

undergroundSystem.checkOut(3, "Sangli", 20);

System.out.println(undergroundSystem.getAverageTime("Mumbai", "Sangli"));

undergroundSystem.checkIn(4, "Mumbai", 24);

undergroundSystem.checkOut(4, "Sangli", 38);

System.out.println(undergroundSystem.getAverageTime("Mumbai", "Sangli"));

}

}