**Client Interview Preparation**

You are on the verge of completing your training and being on-boarded to your first project. Before on-boarding, you are required to take up a client interview. Your academy wants to ensure that you clear the interview with flying colors. In a quick brainstorming, they thought every project would have an end user (Customer and Address). So they decided to frame questions based on these two classes.

**Customer Comparison**

Create a class named as **Customer**, which contains following private variables/ attributes,

|  |  |
| --- | --- |
| **Member Field name** | **Type** |
| \_id | long |
| \_name | string |
| \_gender | char (M/F) |
| \_email | string |
| \_contactNumber | string |
| \_createdOn | DateTime (time in 24 hrs clock) dd/MM/yyyy HH:mm:ss |

Mark all the attributes as private  
Create / Generate appropriate Properties  
Create a class named **Program** to test the above class.  
Add a default constructor and a parameterized constructor to take in all attributes.  
When the “Customer” object is printed, it should display the following details: (override ToString method)

* Customer id: \_id
* Customer: \_name
* Customer contact details: \_contactNumber, \_email

Two members are considered same if they have same \_email and \_contactNumber. Implement the logic in the appropriate function. (override Equals method)

**Input format:**  
By default, there are only two customers. More details in Sample IO.  
  
**Output format:**  
If both the customers are same then print "**Customer 1 is same as Customer 2**" without quotes.  
Otherwise, print "**Customer 1 and Customer 2 are different**" without quotes.  
Refer Sample Input and Outputs.

**Sample Input and Output 1:**

Customer1 :

id:

**45**

name:

**John**

Gender:

**M**

email:

**john@a.com**

contact number:

**+997-4854-7485965123**

createdOn:

**12/10/2016 09:30:00**

Customer2 :

id:

**12**

name:

**Marc**

Gender:

**M**

email:

**marc@a.com**

contact number:

**+997-4854-7485965123**

createdOn:

**11/10/2016 10:00:00**

Customer id: 45

Customer: John

Customer contact details: +997-4854-7485965123, john@a.com

Customer id: 12

Customer: Marc

Customer contact details: +997-4854-7485965123, marc@a.com

Customer 1 and Customer 2 are different

**Sample Input and Output 2:**

Customer1 :

id:

**12**

name:

**James William**

Gender:

**M**

email:

**james@a.com**

contact number:

**+88-7489-9958748512**

createdOn:

**15/02/2017 10:00:00**

Customer2 :

id:

**78**

name:

**James Camron**

Gender:

**M**

email:

**james@a.com**

contact number:

**+88-7489-9958748512**

createdOn:

**03/03/2017 16:30:00**

Customer id: 12

Customer: James William

Customer contact details: +88-7489-9958748512, james@a.com

Customer id: 78

Customer: James Camron

Customer contact details: +88-7489-9958748512, james@a.com

Customer 1 is same as Customer 2

**Validate customer details**

There are always typical human entry errors that need to be validated so that the data saved in the system are valid and can be used for later processing. Simple Rules that need to be taken care:

Create a Class named as **Program**, which contains the following methods,

|  |  |
| --- | --- |
| **No** | **Method Name** |
| 1 | public static void ParseName(string name) |
| 2 | public static void IsValidEmail(string email) |
| 3 | public static void PlayContactNumber(string phone) |
| 4 | public  static void UserLifeTime(string time) |

**ParseName ( ):**  
The customer name contains first name and last name separated by any “special character”. Given a name, display the first and last name.  
Examples of Special character like: **' , '** , **'@'** , **'#' , . . . . . (NOTE: There can be any special character).**  
  
**IsValidEmail ( ):**  
Check if the email address entered is valid. A Valid email address would end with 3 domains (com, org, net) and would contain a “@”.  
  
**PlayContactNumber ( ):**  
Every contact number would be in the format (ISD Code-STD Code-Number). The size of the fields would be (3 digits – 4 digits – 10 digits). Given a contact number, find the sum of last 10 digits until it reaches to a single digit. Print the digit.  
  
**UserLifeTime ( ):**  
Given a “CreatedOn” Date, print number of minutes before which the user was created. Assume the current date to be ‘28-07-2017 09:00’.  
Date format : "**dd-MM-yyyy HH:mm**";  
  
**Example for LifeTime calculation:**  
If the createdOn date is "28-07-2017 09:43", then the life time is 43 minutes.

**Sample Input and Output:**

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**1**

Enter name:

**John%Abraham\*Lincoln**

John Abraham Lincoln

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**1**

Enter name:

**Marc;Farnando**

Marc Farnando

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**2**

Enter E-mail id:

**john.com**

Email is invalid

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**2**

Enter E-mail id:

**john@.com**

Email is invalid

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**2**

Enter E-mail id:

**john@a.com**

Email id is valid

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**3**

Enter contact number:

**844-7485-44784578459**

Contact number invalid

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**3**

Enter contact number:

**58-847-8547123654**

Contact number invalid

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**3**

Enter contact number:

**847-7845-9557898865**

Sum of contact number: 7

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**4**

Enter Created on date(dd-MM-yyyy HH:mm):

**29-07-2017 10:30**

Life time is: 1530 minutes

Menu

1. Parse Name

2. Valid Email

3. Play Contact Number

4. User Lifetime

5. Exit

**5**

**Overloading - Payment**

With all the software systems that are being built, one of the widely used utilities in the financial sector is payments through various channels. The widely used channels are OnlineBanking, CreditCard and Wallet. The banks generate revenue by charging a small margin as part of the usage. Create a class called PaymentUtil. Overload a method “makePayment” as given below in the specification and calculate the total payment amount.

Create a class named as **PaymentUtil**, which contains following Overload methods,

|  |  |  |
| --- | --- | --- |
| **No** | **Method Name** | **Method Description** |
| 1 | public double MakePayment(Dictionary<string,float> bankTax,string bankName,double amount) | This method calculates the total amount (amount + service tax amount). Here the tax percent gets varied depending on the bank. Fetch the corresponding service tax from a map collection by bank name. Map<String bankName, Float serviceTax> This map is prefilled. Only banks present in the map would be part of input. |
| 2 | public double MakePayment(double amount) | This method calculates the total amount (include service tax and value added tax(VAT). First, calculate the service tax amount and then calculate the VAT amount. [ VAT % is applied on total amount+service tax ] |
| 3 | public double MakePayment(double amount, float discountPercent) | This method calculates the total amount, which is a discount from the amount with parameterized discount percentage. |

Create the class named as **Program**, which contains the Main method. Input and output operations will be done in the main method.  
It calls various methods in PaymentUtil class to perform the task.  
  
The following data will be placed inside the **Main** class

        onlineBankingDict.Add("ICICI", 4.2f);

        onlineBankingDict.Add("IBRD", 3f);

        onlineBankingDict.Add("IFC", 4.9f);

        onlineBankingDict.Add("HSBC", 3.9f);

That data will provide in template code.

**Problem Specification:**  
Overload a method “MakePayment” as given in the specification and calculate the total payment amount.  
The bank name is case insensitive.  
The service tax for credit card is 5.2%.  
The VAT for the creadit card is 2.3%.  
The discount  percentege for Wallet is 20.2%.  
  
**Input and Output format:**  
Refer Sample Input and Output.  
  
**Sample Input and Output 1:**

1. Online banking

2. Credit card

3. Wallet

Enter the choice:

**1**

Enter the user name:

**ICICI74484**

Enter the password:

**Mu7485**

Enter the amount:

**12560**

Enter the bank name:

**icici**  
Username : ICICI74484  
Password : Mu7485  
Amount : 12560  
Bank : icici

Total amount(Inclusive of Service Tax): 13087.52  
  
**Sample Input and Output 2:**

1. Online banking

2. Credit card

3. Wallet

Enter the choice:

**2**

Enter the account number:

**8458 9665 7485 2256**

Enter the pin:

**8451**

Enter the amount:

**16300**  
AccountNumber : 8458 9665 7485 2256  
Pin : 8451  
Amount : 16300

Total amount(Inclusive of Service Tax and VAT): 17541.99  
  
**Sample Input and Output 3:**

1. Online banking

2. Credit card

3. Wallet

Enter the choice:

**3**

Enter the user name:

**HSBC7457**

Enter the password:

**Yu67488**

Enter the amount:

**28500**  
AccountNumber : HSBC7457  
Password : Yu67488  
Amount : 28500

Discounted amount: 22743.00

**Find customer by id and state**  

Create a class named as **Address**, which contains following private member variables,

|  |  |
| --- | --- |
| **Member Field name** | **Type** |
| \_street | string |
| \_city | string |
| \_state | string |
| \_country | string |
| \_zipCode | int |

Create appropriate Properties.

Add a default constructor and a parameterized constructor to take in all attributes.

Create a class named as **Customer**, which contains following private member variables,

|  |  |
| --- | --- |
| **Member Field name** | **Type** |
| \_id | long |
| \_name | string |
| \_gender | char (M/F) |
| \_email | string |
| \_contactNumber | string |
| \_createdOn | DateTime (time in 24 hrs clock) dd/MM/yyyy HH:mm:ss |
| \_address | Address |

Create appropriate Properties.

Add a default constructor and a parameterized constructor to take in all attributes.

Implement the following methods in the Customer class

|  |  |  |
| --- | --- | --- |
| **No** | **Method Name** | **Method Description** |
| 1 | public Customer FindCustomerById(List<Customer> customerList, int id) | In this method, that takes up an id and returns a matching customer object. If the object is not found then return null. |
| 2 | public List<Customer> FindCustomerListByState(List<Customer> customerList, string state) | In this method, given a state as a parameter, return the list of customers who belong to the state. |

The customer details will be populated (provided in the template code).  
Refer Sample Input and Output.  
Use the following format for specified output.  
**"{0,-15}{1,-20}{2,-15}{3,-15}{4}", "Name", "Email", "City", "Country", "Zipcode"**  
  
**Problem Specification:**  
If the customer with the id is not found then print "**No Customer with that id**" without quotes.  
If none of the customer belongs to the state then print "**No customer belongs that state**" without quotes.  
  
**Sample Input and Output 1:**

Menu

1. Find customer by id

2. Find customer by states

Enter the choice:

**1**

Enter the Id to find customer:

**5**

Customer Name: Tedmond

Email: tedmond@a.com

City: Tacoma

State: Washington

Country: USA

Zip code: 98412  
  
**Sample Input and Output 2:**

Menu

1. Find customer by id

2. Find customer by states

Enter the choice:

**2**

Enter the state:

**Texas**

Name                Email                                 City                Country        Zipcode

Aekerman       aekerman@a.com      Plano             USA               75025

Madeleine       madeleine@a.com      Lubbock      USA               79404

Raymond        raymond@a.com         Texas City    USA              77591

**Batch Processing**   

Usually, As part of batch processing jobs, a CSV or TXT file is read, relevant objects are created and database is populated with CSV contents.  
Create the list of customer objects with the CSV content provided in the sample IO.

Create a class named as **Customer**, which contains following private variable/ attributes,   

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Member Field name** | **Type** | | \_id | long | | \_name | string | | \_gender | char (M/F) | | \_email | string | | \_contactNumber | string | | \_createdOn | DateTime (time in 24 hrs clock) mm/dd/yyyy HH:mm:ss | |  |

Mark all the attributes as private

Include properties

Add a default constructor and a parameterized constructor to take in all attributes.

In the Customer class, implement the following methods.

|  |  |  |
| --- | --- | --- |
| **No** | **Method Name** | **Method Description** |
| 1 | public static List<Customer> PopulateCustomers(List<string> csvList) | In this method, given parameter is the list of customer details in a string format where each data is separated by a comma. Parse the string and create a customer arrayList. |
| 2 | public static List<Customer> FindCustomerNameFromList(List<Customer> customers, string subString) | In this method, given part of customer name, search the customer list based on name and return the customer list with matching names. |

**Input format:**  
The first input consists of an integer that corresponds  to the number of customer  n.  
The next n line of the input consists of a string that corresponds to the customer details, which is separated  by a comma.  
Input sequence:  
**id, name, gender, email, contactNumber, createdOn.**  
The last input is the substring that used to search the specified customers.  
  
**Output format:**  
Refer Sample Input and Output.  
  
**Sample Input and Output:**

Enter the number of customer:   
**5**   
Enter the customer 1 detail:   
**12,John Smith,M,johnsmith@a.com,+85-7489-8596478596,12/12/2016 12:30:00**   
Enter the customer 2 detail:   
**15,Tedmond,M,tedmond@a.com,+45-9857-5266987485,14/01/2017 04:30:00**   
Enter the customer 3 detail:   
**11,Dalton,M,dalton@a.com,+48-8967-7485947558,12/02/2017 20:00:00**   
Enter the customer 4 detail:   
**5,Raymond,M,raymond@a.com,+88-8745-8554712569,28/01/2017 10:30:00**   
Enter the customer 5 detail:   
**9,Ruford,M,ruford@a.com,+88-4859-7714589633,01/04/2017 17:45:00**   
Id   Name           GenderEmail                    Contact no          Created on   
12   John Smith     M    johnsmith@a.com          +85-7489-8596478596 12/12/2016 12:30:00 PM   
15   Tedmond        M    tedmond@a.com            +45-9857-5266987485 1/14/2017 4:30:00 AM   
11   Dalton         M    dalton@a.com             +48-8967-7485947558 2/12/2017 8:00:00 PM   
5    Raymond        M    raymond@a.com            +88-8745-8554712569 1/28/2017 10:30:00 AM   
9    Ruford         M    ruford@a.com             +88-4859-7714589633 4/1/2017 5:45:00 PM   
Enter the substring to search from customer list:   
**mon**   
Id   Name           GenderEmail                    Contact no          Created on   
15   Tedmond        M    tedmond@a.com            +45-9857-5266987485 1/14/2017 4:30:00 AM   
5    Raymond        M    raymond@a.com            +88-8745-8554712569 1/28/2017 10:30:00 AM

**Customer & Dictionary**

Create a class named as **Address**, which contains following private member variables,

|  |  |
| --- | --- |
| **Member Field name** | **Type** |
| \_street | string |
| \_city | string |
| \_state | string |
| \_country | string |
| \_zipCode | int |

Create appropriate Properties.

Add a default constructor and a parameterized constructor to take in all attributes.

Create a class named as **Customer**, which contains following private member variables,

|  |  |
| --- | --- |
| **Member Field name** | **Type** |
| \_id | long |
| \_name | string |
| \_gender | char (M/F) |
| \_email | string |
| \_contactNumber | string |
| \_createdOn | DateTime (time in 24 hrs clock) dd/MM/yyyy HH:mm |
| \_address | Address |

Create appropriate Properties.

Add a default constructor and a parameterized constructor to take in all attributes.

Implement the following static methods in the Customer class

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Method Name** |  | **Method Description** |
| 1 | public static Dictionary<string, int> convertCsvToDictionary(List<string> csvDetails) |  | This method returns a Dictionary<key, value>. The key would be the state name and count would be the number of customers in the corresponding state. |
| 2 | public static List<Customer> GetCustomerListFromDictionary(Dictionary<string, int> customerMap) |  | This method returns the customer list, which is sorted (Ascending) based on both state and name. State – sorted based on string ascending order. Customer – sorted based on name ascending order. |

**Input Format:**

The first input consists of an integer that corresponds to the number of customers n.

The next n input consists of a string that corresponds to the customer details, which is separated by the comma (,).

Input sequence - id,name,gender,email,contactNumber,createdOn,street,city,state,country,zipCode.

**Output Format:**

Display the list in the following format

{0,-15}{1,-20}{2,-5}{3,-20}{4,-20}{5,-20}{6,-15}{7,-15}{8,-15}{9,-15}{10,-15}", "Id", "Name", "Gender", "Email", "Contact Number", "Created On", "Street", "City", "State", "Country", "Zipcode"

**Sample Input and Output:**

Enter the number of customer:

**10**

Enter the customer 1 details:

**1,John Smith,M,johnsmith@a.com,+89-7485-8578974885,15/01/2016 10:30,112th St,Utica,New York,USA,13455**

Enter the customer 2 details:

**2,Aekerman,M,aekerman@a.com,+99-7489-8857945569,14/02/2016 16:30,Avenue,Austin,Texas,USA,88596**

Enter the customer 3 details:

**3,Madeleine,F,madeleine@a.com,+88-7859-7748599989,25/01/2016 10:00,155th St,Plano,Texas,USA,56684**

Enter the customer 4 details:

**4,Edrick,M,edrick@a.com,+99-7482-4115233987,18/12/2016 07:30,111th St,Sitka,Alaska,USA,66584**

Enter the customer 5 details:

**5,Tedmond,M,tedmond@a.com,+77-8599-4225610074,05/01/2016 08:30,Parc St,Olympia,Washington,USA,85574**

Enter the customer 6 details:

**6,Nelson,M,nelson@a.com,+78-7488-4221258447,02/01/2017 10:30,5th St,Dayton,Ohio,USA,84587**

Enter the customer 7 details:

**7,Dalton,M,dalton@a.com,+78-8547-8555479512,15/01/2017 20:30,North St,Buffallo,New York,USA,25664**

Enter the customer 8 details:

**8,Raymond,M,raymond@a.com,+89-7484-8577458895,25/02/2017 10:30,15th St,Waco,Texas,USA,7858**

Enter the customer 9 details:

**9,Rosemary,F,rosemary@a.com,+88-4888-7485998741,20/02/2017 12:30,15th St,Tacoma,Washington,USA,87458**

Enter the customer 10 details:

**10,Ruford,M,ruford@a.com,+84-422-9887485995,22/03/2017 09:30,9th St,Butte,Montana,USA,22458**

State - No of customers

Alaska - 1

Montana - 1

New York - 2

Ohio - 1

Texas - 3

Washington - 2

Id  Name  Gender Email   Contact number  Created on  Street City State    Country Zipcode

4 Edrick M edrick@a.com  +99-7482-4115233987 18/12/2016 07:30 111th St Sitka       Alaska USA 66584

10 Ruford M ruford@a.com  +84-422-9887485995 22/03/2017 9:30  9th St Butt ,Montana USA 22458

7 Dalton M dalton@a.com  +78-8547-8555479512 15/01/2017 20:30 North St Buffallo New York USA 25664

1 John Smith M johnsmith@a.com +89-7485-8578974885 15/01/2016 10:30 112th St  Utica New York USA 13455

6 Nelson M nelson@a.com +78-7488-4221258447 02/01/2017 10:30 5th St Dayton Ohio USA 84587

2,Aekerman M aekerman@a.com +99-7489-8857945569 14/02/2016 16:30 Avenue Austin Texa ,USA 88596

3 Madeleine F madeleine@a.com +88-7859-7748599989 25/01/2016 10:00 155th St,Plano Texas USA 56684

8 Raymond M raymond@a.com  +89-7484-8577458895 25/02/2017 10:30 15th St Waco Texas USA 7858

9 Rosemary F rosemary@a.com  +88-4888-7485998741 20/02/2017 12:30 15th St Tacoma Washington USA 87458

5 Tedmond M tedmond@a.com  +77-8599-4225610074 05/01/2016 08:30 Parc St Olympia  Washington USA 85574