Food aggregator - Requirement 1

You being an aspiring entrepreneur wish to start a Food aggregator app of your own. You have collected the details of menu and user details from hotels around your city. Now it's time to develop your app by creating domains and integrate collected details into it. You have chosen four domains for your app namely Item, Order, Purchase and User. Let's add features to your app one by one.

Requirement 1:

Let's start off by creating two **User** objects and check whether they are equal.

1. Create a **User** Class with the following properties:

Member Field Name	**Type**
_name	string
_email	string
_phoneNumber	string
_location	string

2. Mark all the properties as private
3. Create / Generate appropriate Getters & Setters for properties
4. Add a parameterized constructor to take in all properties in the given order:
   User (string _name, string _email, string _phoneNumber , string _location)
5. When the "User" object is printed, it should display the following details: **[Override the ToString method]**
   Print format:
   Name: "_name"
   Email: "_email"
   PhoneNumber: "_phoneNumber "
   Location: "_location"
6. Two Users are considered same if they have the same name, and email. Implement the logic in the appropriate function. (Case – Insensitive) **[Override the Equals method]**

   The input format consists of User details separated by comma in the below order,
   _name, _email, _phoneNumber , _location

The Input to your program would be details of two Users, you need to display their details as given in "5th point(refer above)" and compare the two Users and display if the Users are same or different.

Note: There is an empty line between display statements. Print the empty lines in main function.

Sample Input/Output:

```
Enter user 1 detail:
Oliver,oliver@gmail.com,7856124589,SanFrancisco
Enter user 2 detail:
```

```
Harry,harry@gmail.com,9856231478,NewYork

User 1
Name: Oliver
Email: oliver@gmail.com
PhoneNumber: 7856124589
Location: SanFrancisco

User 2
Name: Harry
Email: harry@gmail.com
PhoneNumber: 9856231478
Location: NewYork

User 1 and User 2 are different
```

Sample Input/Output 2:

```
Enter user 1 detail:
Harry,harry@gmail.com,9856231478,NewYork
Enter user 2 detail:
haRRy,harry@gmail.com,9856231478,NewYork

User 1
Name: Harry
Email: harry@gmail.com
PhoneNumber: 9856231478
Location: NewYork

User 2
Name: haRRy
Email: harry@gmail.com
PhoneNumber: 9856231478
Location: NewYork

User 1 is same as User 2
```

Food Aggregator - Requirement 2

Requirement 2:

In this requirement, you need to validate the email of the User.

a)Create a Class **Program** with the following static methods:

Method Name	**Description**
static bool ValidateEmail(string email)	Validate the Email based on the rules given below. Returns **true** if Email is valid else return **false**

b) While validating email follow the below rules. The format of the email id is given below

username@domain.TLD

where, TLD - Top Level Domain

1. The email should start only with alphabets(lowercase).
2. The username part of email can contain alphabets(lowercase), numbers and the special characters (. and _).
3. The username part of email should not contain any special characters other than **" . "** and **" _ "**.
4. After the username special character @ should present.
5. The email domain should contain only alphabets(lowercase).
6. After email domain, a dot (**.**) should be present.
7. The email Top Level Domain should contain only alphabets(lowercase) and it should have only 2 to 6 characters.

Example: **harson_Wells.01@google.com** is a valid email id.
Since the user name contain only alphabets, numbers and a special character (_ and .), then the @ symbol is present. The domain name should contain only alphabets and the symbol dot(.) and the Top Level Domain have only 3 characters.

Note: Print "**Email is valid**" if email is valid else print "**Email is invalid**".
All the above print statements are present in the main method.

[All text in bold corresponds to input]
Sample Input and Output 1:

Enter the email to be validated:
harry@gmail.com
Email is valid

Sample Input and Output 2:

```
Enter the email to be validated:
jerin$80@yahoo.com
Email is invalid
```

Food aggregator - Requirement 3

Requirement 3:
In this requirement develop a feature in which you can search a List of Items by type and price.

a) Create a Class Item with the following properties:

Member Field Name	**Type**
_name	string
_price	double
_type	string

Mark all the properties as private,
Create / Generate appropriate Getters & Setters for properties ,
Add a default constructor and a parameterized constructor to take in all properties in the given order:
Item(string _name, double _price , string _type)

b) Create a class **ItemBO** with the following methods,

Method Name	**Description**
public List<Item> FindItem(List<Item> itemList,string type)	This method accepts a list of items and type as arguments and returns a list of items that matches with the given type.
public List<Item> FindItem(List<Item> itemList,double price)	This method accepts a list of items and price as arguments and returns a list of items that matches with the given price.

The item details should be given as a comma-separated value in the below order,
_name, _price, _type

Print format:
Console.Write("{0,-20} {1,-5} {2}\n","Name","Price","Type");

Note: The item lists are displayed in the main method.
If any other choice is selected, display "**Invalid choice**"
If the search detail is not found, display "**No such item is present**"
Display one digit after the decimal point for double Datatype.

Sample Input and Output 1:
Enter the number of items:

```
4
Paneer Fried Rice,150,Veg
Chicken Fried Rice,210,NonVeg
Bucket Chicken,479,NonVeg
Ghee Roast,75,Veg
Enter a search type:
1.By Type
2.By Price
1
Enter the Type:
Veg

Name                    Price   Type

Paneer Fried Rice       150.0   Veg

Ghee Roast              75.0    Veg
```

Sample Input and Output 2:

```
Enter the number of items:
4
Paneer Fried Rice,150,Veg
Chicken Fried Rice,210,NonVeg
Bucket Chicken,479,NonVeg
Ghee Roast,75,Veg
Enter a search type:
1.By Type
2.By Price
2
Enter the Price:
479

Name                  Price    Type

Bucket Chicken        479.0    Non Veg
```

Sample Input and Output 3:

```
Enter the number of items:
4
Paneer Fried Rice,150,Veg
Chicken Fried Rice,210,NonVeg
Bucket Chicken,479,NonVeg
Ghee Roast,75,Veg
Enter a search type:
1.By Type
2.By Price
3
Invalid choice
```

Food aggregator - Requirement 4

Requirement 4:

In this requirement, you need to sort the list of items based on name and price.

a) Create a Class Item with the following properties:

Member Field Name	Type
_name	string
_price	double
_type	string

Mark All properties as private

Create / Generate appropriate Getters & Setters for properties

Add a default constructor and a parameterized constructor to take in all properties in the given order:

Item(string _name,double _price, string _type)

b) Create the following static methods in the Item class,

Method Name	Description
static Item CreateItem(string detail)	This method accepts a string. The item detail separated by commas is passed as the argument. Split the details and create a item object and returns it.

The item details should be given as a comma-separated value in the below order,
_name, _type, _price

c) The Item class should implement the **IComparable** interface which sorts the Item list based on **name**(case-insensitive). While comparing, all the name properties in the list are unique.

d) Create a class **PriceComparator** which implements **Comparer** interface and sort the Item list based on **price**. While comparing, all the price properties in the list are unique.

Get the number of Items and item details and create a item list. Sort the Items according to the given option and display the list.

When the "item" object is printed, it should display the following details
Print format:

```
Console.Write("{0,-20} {1,-10} {2,-12}\n","Name","Price","Type");
```

Note: Display one digit after decimal point for double datatype.

Sample Input & Output 1:

```
Enter the number of items:
4
Paneer Fried Rice,150.0,Veg
Chicken Fried Rice,210.0,NonVeg
Bucket Chicken,479.0,NonVeg
Ghee Roast,75.0,Veg
Enter a type to sort:
1.Sort by Name
2.Sort by Price
1
Name                    Price      Type
Bucket Chicken        479.0      NonVeg
Chicken Fried Rice   210.0      NonVeg
Ghee Roast               75.0       Veg
Paneer Fried Rice     150.0      Veg
```

Sample Input & Output 2:

```
Enter the number of items:
4
Paneer Fried Rice,150.0,Veg
Chicken Fried Rice,210.0,NonVeg
Bucket Chicken,479.0,NonVeg
Ghee Roast,75.0,Veg
Enter a type to sort:
1.Sort by Name
2.Sort by Price
2
Name                    Price      Type
Ghee Roast               75.0       Veg
Paneer Fried Rice     150.0      Veg
Chicken Fried Rice   210.0      NonVeg
Bucket Chicken        479.0      NonVeg
```

Food Aggregator Requirement - 5

Requirement 5:
In this requirement develop a feature to compute the price for every purchase based on the price of each item, quantity and coupon code.

a) Create a Class **Purchase** with the following properties:

Member Field Name	**Type**
_id	int
_price	double
_couponCode	string
_purchaseDate	DateTime
_orderList	List<Order>

Mark all the properties as private,
Create / Generate appropriate Getters & Setters for properties,
Add a default constructor and a parameterized constructor to take in all properties in the given order:
Purchase(int _id, string _couponCode, DateTIme _purchaseDate, List<Order> _orderList)

b) Create the following methods for **Purchase** class,

Method Name	**Description**
public static void ComputePrice(List<Purchase> list)	This method accepts a list of purchase objects as arguments and sets the computed price value for each purchase.

c) Create a Class **Order** with the following properties:

Member Field Name	**Type**
_quantity	int
_item	Item

Mark all the properties as private,

Create / Generate appropriate Getters & Setters for properties,
Add a default constructor and a parameterized constructor to take in all properties in the given order:
Order(int _quantity, Item _item)

d) Create a Class **Item** with the following properties:

Member Field Name	Type
_name	string
_price	double
_type	string

Mark all the properties as private,
Create / Generate appropriate Getters & Setters for properties,
Add a default constructor and a parameterized constructor to take in all properties in the given order:
Item(string name, string _type, double _price)

e) Create the following methods for **Item** class,

Method Name	Description
public static List<Item> Prefill()	This method returns a list of Item objects that are prefilled in the template code.

The purchase details should be given as a comma-separated value in the below order,
_id, _couponCode, _purchaseDate
The order details should be given as a comma-separated value in the below order,
_quantity, _itemName

Print format:

```
Console.Write("{0,-5} {1,-10} {2,-12} {3}\n","Id","Price","Coupon Code","Purchase Date");
```

Note: use the Prefill() method, given in the template, to get the Item objects.
Display one digit after the decimal point for Double Datatype.

While computing price, multiply the quantity with the item's price in the orderList.

If any one of the coupon codes are applied, discount the price based on the below criteria.

FIRST	Rs.100 discount
BUYFIVE	Rs.500 discount
ORDER50	50% discount
ORDER75	75% discount
DEAL25	25% discount
CHICKEN70	70% discount

Sample Input and Ouput:

```
Enter the number of Purchase
2
Enter purchase detail 1
1,FIRST,12-02-2018
Enter the number of Orders
2
2,Grill
1,Margherita
Enter purchase detail 2
2,ORDER50,13-02-2018
Enter the number of Orders
3
2,Tandoori Chicken
1,Barbeque Chicken
1,Ghee Roast
Id   Price     Coupon Code  Purchased Date
1    799.0     FIRST             12-02-2018
2    737.5     ORDER50       13-02-2018
```

Food Aggregator - Requirement 6

Create a **User** class with the following private properties

Member Field Name	**Type**
_name	string
_email	string
_phoneNumber	string
_location	string
_purchaseList	List<Purchase>

Mark All properties as private
Create / Generate appropriate Getters & Setters for properties
Add a default constructor and a parameterized constructor to take in all properties in the given order:
User (string _name, string _email, string _phoneNumber , string location ,List<Purchase> _purchaseList)

The following methods are present in the User class

Method Name	**description**
static List<User> Prefill()	This method returns a List of User objects that are prefilled in the template code.
static User GetValuableUser(List<User> userList,string month)	This method accepts List of User objects and month name as arguments. This method returns a user object that has the maximum cumulative purchase amount in the specified month.

Create a **Purchase** Class with the following private properties

Member Field Name	**Type**
_id	int
_price	double
_couponCode	string

_purchaseDate	DateTime
_user	User

Mark All properties as private
Create / Generate appropriate Getters & Setters for properties
Add a default constructor and a parameterized constructor to take in all properties in the given order:
Purchase(int i_d, double _price, string c_couponCode, DateTime _purchaseDate, User _user)

The inputs to the purchase is given the order below,
_id, _price, _couponCode, _purchaseDate, _username

Create a driver class called Program. In the main method, obtain purchase details and display the valuable user of the specified month by calling appropriate methods.

Note: The output should be in the format "The valuable user of the month [month name] is [valuable user name] (Refer Input/Output specification)
Use "dd-MM-yyyy" for purchaseDate property.

Sample Input/Output 1:

```
Enter the number of purchases:
4
1,750,FIRST,05-06-2018,Rob
2,1500,DEAL25,05-06-2018,Brandon
3,2500,FIRST,07-07-2018,Joe
4,800,DEAL25,08-06-2018,Rob
Enter the month:
June
The valuable user of the month June is Rob
```

Sample Input/Output 2:

```
Enter the number of purchases:
5
120,1500,CHICKEN70,06-04-2018,Oliver
160,2500,CHICKEN70,08-04-2018,Harry
190,3000,FIRST,09-04-2018,Oliver
```

```
225,4500,BUYFIVE,08-04-2018,Oliver
280,500,BUYFIVE,12-04-2018,Rob
Enter the month:
April
The valuable user of the month April is Oliver
```