Lab Guide

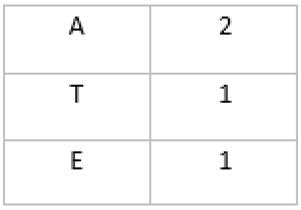
C#

M1 Improvement

Lab 1

Topics: Create the Product class as described in Requirement 2(C# Case Study-Requirement 2)

Create a class Shop that maintains a list of all products available in the shop.
Add/implement the below members inside the Shop class: public string shopname
public static List<Product> plist: list to maintain all products of the shop.
public void PreFillProducts(): A method that prepares a list of Product objects (Hardcoded values).
public Product FindProduct(string code,List<Product> plist): A method that finds a product based on the product code passed as first argument , inside the list plist and returns that object.
public List<Product> FindProducts(char catcode, List<Product> plist):A method that returns list of products having category code catcode(first argement) inside the list plist.
public List<Product> FindProducts(double price, List<Product> plist):A method that returns list of products for a given price(passed as first argument) inside the list plist.
public Dictionary<string cat,int count> CategoryWiseCount(List<Product> plist) :A method that returns the count of products under each category in the plist.
To understand, lets consider that there are 2 products availaible for a category code "A" , one product for category code T, one product for Category "E" .
The method CategoryWiseCount() is expected to return a structure that gives the below output when printed.



A	2
T	1
E	1

Note : First field in the above output represents the category code and the second field represents number of products for the given category.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
public class Product
{
```

```
public string productCode, productName;
public double productPrice;
public char categoryCode;
public Product() { }
   public Product(string PCode, string Name, double Price, char Code)
   {
      productCode = PCode;
      productName = Name;
      productPrice = Price;
      categoryCode = Code;
   }
   public string PrintProduct()
   {
      return ("Code-" + productCode.ToString() + "\nName-" + productName +
"\nPrice-" + productPrice.ToString() + "\nCategory-" + categoryCode.ToString());
   }
public string ReadProductDetails(string s1, string s2)
   {
      string[] i1 = s1.Split(',');
      string[] i2 = s2.Split(',');
Product p1 = new Product {
productCode=i1[0],productName=i1[1],productPrice=double.Parse(i1[2]),categoryCo
de=i1[3][0]};
      Product p2 = new Product { productCode = i2[0], productName = i2[1],
productPrice = double.Parse(i2[2]), categoryCode = i2[3][0] };
      if ((p1.productCode != p2.productCode) | (p1.productName.ToLower() !=
p2.productName.ToLower()))
         return "Product 1 and  Product 2 are different";
      else
         return "Product 1 is same as Product 2";
   }
}
plist.Add(new Product
{productCode="1",productName="p1",productPrice=1.00,categoryCode='A' });
      plist.Add(new Product {productCode = "2", productName = "p2", productPrice =
1.00, categoryCode = 'A' });
```

```
        plist.Add(new Product { productCode = "3", productName = "p1", productPrice =
2.00, categoryCode = 'B' });
public Product FindProduct(string code, List<Product> plist)
    {
Product p = plist.FirstOrDefault(x => x.productCode == code);
        return p;
    }
public List<Product> FindProducts(char catcode, List<Product> plist)
    {
List<Product> l1 = plist.FindAll(x => x.categoryCode == catcode);
        return l1;
    }
public List<Product> FindProducts(double price, List<Product> plist)
    {
        List<Product> l1 = plist.FindAll(x => x.productPrice==price);
        return l1;
    }
public Dictionary<string , int > CategoryWiseCount(List<Product> plist)
{
        Dictionary<string ,int > d1=new Dictionary<string,int>();
       var res = from p in plist
group p by p.categoryCode into catgroup
                  select new { category = catgroup.Key, count = catgroup.Count() };
       foreach(var r1 in res)
        {
            d1.Add(r1.category.ToString(), r1.count);

        }
return d1;

    }
}
```

Lab 2

Create a class named Employee with the following **private** instance variables

- empCode of type String
- empName of type String
- empSal of type double
- deptCode of type char (E- electronics, H - human resource, A – admin)

Include a private static variable empCounter of type int initialized to 1000.

Declare all above members in the same order as mentioned above.

Create getters and setters for all variables

Create a private method generateEmployeeCode which will return the employee code as String. Employee code is derived by concatenating deptCode and incremented empCounter.

Include a parameterized constructor with 3 parameters (empName, empSal, deptCode). deptCode should be assigned in constructor by using generateEmployeeCode method. Initialize all the member variables.

Include an overloaded parameterized constructor with 2 parameter(empName, empSal). deptCode should be assigned in constructor by using generateEmployeeCode method. Department code should be assigned to ‘A’. Initialize all the member variables.

Include a method getEmployeeDetails to format the employee details. This method should return a string containing the employee details in the below format

Code–1001E,Name–Sagar,Salary-45000.00,Department–A

Assuming that the above employee object is constructed with the help of below statement

Employee e=new Employee("Sagar",45000.00, 'A');

Note :While implementing Getters and Setters , use the instance variable name with first letter replaced by upper case letter

Example :

if variable is declared as

int age ;

then corresponding property for the above variable is :

```
public int Age

{

// implementation code goes here

}
```

Note: Do not implement the Main Method

```
public class Employee

  {

    private string empCode, empName;

    private double empSal;

    private char deptCode;

    private static int empCounter = 1000;

   public string EmpCode

   {

     get

     {

       return empCode;

       }

     set

     {

       empCode = value;
```

```
      }
    }

    public string EmpName
    {
      get
      {
        return empName;
      }
set
      {
        empName = value;
      }
    }

    public double EmpSal
    {
      get
      {
        return empSal;
      }

      set
      {
        empSal = value;
      }
    }

    public char DeptCode
    {
```

```
  get
  {
    return deptCode;
  }

  set
  {
    deptCode = value;
  }
}
private string generateEmployeeCode(char cc)
{
  empCounter++;
  string str = empCounter.ToString() + cc;
  return str;
}



public Employee(string Name, double Sal, char Code)
{
  empCode = generateEmployeeCode(Code);
 // empCounter++;
  empName = Name;
 empSal = Sal;
  deptCode = Code;
}
public Employee(string Name, double Sal)
{
  char cc = 'A';
  empCode = generateEmployeeCode(cc);
```

```
            empCounter++;

            empName = Name;

           empSal = Sal;


        }
        public string  getEmployeeDetails()
        {


          // Code–101E, Name – Laptop price - 45000.00, Category – E

            // Code–1001E,Name–Sagar,Salary-45000.00, Department–A

            return ("Code-"+empCode.ToString() + ",Name-" + empName + ",Salary-" + empSal.ToString() + ",Department-" + deptCode.ToString());


        }
    }
```

Lab 3

1) Create a class Product with following public attributes.

ProductCode:string

Name:string

Price:double

Brand:string

2) Create a class Shop with following public attributes

Name:string

ProdList:List<Product>

3) Add default constructor and parameterised constructor to initialize all member fields of the class.The order of initialization is as given below: Shop(string name,List<Product> productList)
4) Create the following methods in the Shop class

   Public void AddProductToShop(Product p) : This method accepts a product object and adds the product to the product list of the current shop .If the product with same code and name already available in the list , then product should not be added to the list.

   Public bool RemoveProductFromShop(string productCode) :This method will get the productCode of the product and deletes the product with specified code from the current shop.If a product with a given code is found , then deletes the product and returns true.If product with productCode is not found then returns false.

   public HashTable BrandWiseCount(List<Product> productList) : This method accepts a list of products and returns an object that contains brand-wise count of products.

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

class Product

  {

       public string ProductCode;

       public string Name;

       public double Price;

       public string Brand;

   }
```

```
class Shop
{
    public string Name;
    public List<Product> ProdList=new List<Product>();
    public Shop() { }
    public Shop(string name, List<Product> productList)
    {
        Name = name;
        ProdList = productList;
    }
    public void AddProductToShop(Product p)
    {
        if (ProdList.Count == 0)
            ProdList.Add(p);
        else
        {

            Product obj = ProdList.Find(x => x.Name.ToLower() == p.Name.ToLower() && x.ProductCode == p.ProductCode);
            if ((ProdList.Contains(obj)) == false)
                ProdList.Add(p);


        }


          }
```

```
public bool RemoveProductFromShop(string productCode)
        {
            bool flag;
            Product obj = ProdList.Find(x =>  x.ProductCode == productCode);
            if ((ProdList.Contains(obj)) == true)
            {
                flag = true;
                ProdList.Remove(obj);
            }
            else
                flag = false;
            return flag;
        }


    }
    class ProductBO
    {
        public List<Product> FindProduct(List<Product> productList, string
brand)
            {
            var res = from p in productList
                      where p.Brand == brand
                      select p;
            return res.ToList();
            }


        public List<Product> FindProduct(List<Product> productList, double
price)
```

```
        {
            var res = from p in productList
                      where p.Price == price
                      select p;
            return res.ToList();
        }
        public Hashtable BrandWiseCount(List<Product> productList )
        {
var res= from p in productList
                      group p by p.Brand into g
                      select new  { brand=g.Key,count=g.Count() };
            Hashtable ht1 = new Hashtable();
            foreach(var x in res)
            {
                ht1.Add(x.brand, x.count);

            }
            return ht1;

        }
    }
```

Lab 4

Problem Statement - Utility Static Methods

Complete the static methods in the class Utility as per following requirements

Method fahrenheitToCelcius :

- This method should convert farhenheit in to celcius based on the formula [celcius = (farhenheit - 32) X 5 / 9]
- The method takes farhenheit(double) as input parameter
- Method should return calculated temperature in celcius rounded to an integer

Method getLevel:

- Takes an integer array as input parameter
- Should calculate the sum of all array elements and return a String as per below rules
- HIGH - when sum is greater than or equal to 100, MEDIUM - when sum is greater than or equal to 70, LOW - when sum is less than 70

Complete the main method in class StaticMethods as below

- Program should take console input and call appropriate methods of Utility class based on the input
- Input and Output sample formats are given below in Example section
- First input should be option 1 or 2. Option 1 is for Celcius calculation;Option 2 for finding Level
- In case of option 1, the second input should be temperature in farhenheit
- In case of option 2, the second input should be number of elements in the array, followed by the array elements
- In case of incorrect option, program should display 'Invalid Option'

Example

```
Sample Input:
1                          // Option
100                        // temperature in Farhenheit
Expected Output:
38

Sample Input:
1
95.5
Expected Output:
35
```

```
Sample Input:
2                          // option
3                          // number of elements in array
40                         // array elements
50                         // array elements
11                         // array elements
Expected Output:
HIGH


Sample Input:
2
4
10
20
5
10
Expected Output:
LOW
```

Instructions

- Do not change the provided class/method names unless instructed
- Ensure your code compiles without any errors/warning/deprecations
- Follow best practices while coding
- Avoid too many & unnecessary usage of white spaces (newline, spaces, tabs, ...), except to make the code readable
- Use appropriate comments at appropriate places in your exercise, to explain the logic, rational, solutions, so that evaluator can know them
- Try to retain the original code given in the exercise, to avoid any issues in compiling & running your programs
- Always test the program thoroughly, before saving/submitting exercises/project
- For any issues with your exercise, contact your coach

Warnings

- Take care of whitespace/trailing whitespace
- Trim the output and avoid special characters
- Avoid printing unnecessary values other than expected/asked output

```
using System;


namespace LearnCsharp

{

public class Utility {
    public static int fahrenheitToCelcius(double farhenheit) {
        //CODE START
        return (int) Math.Round(5.0 / 9.0 * (farhenheit - 32));
        //CODE END
    }

    public static string getLevel(int[] array) {
        //CODE START
        int sum = 0;
        foreach (int a in array) {
            sum += a;
        }
        if (sum >= 100) {
            return "HIGH";
        } else if (sum >= 70) {
            return "MEDIUM";
        } else {
            return "LOW";
        }
        //CODE END
    }
}



public  class StaticMethods{
     public static void Main(string[] args) {
        //CODE START
        int option = Convert.ToInt32(Console.ReadLine());
        switch (option) {
        case 1:
            double farhenheit = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine(Utility.fahrenheitToCelcius(farhenheit));
            break;
        case 2:
            int count = Convert.ToInt32(Console.ReadLine());
            int[] arr = new int[count];
            for (int i = 0; i < count; i++) {
                arr[i] = Convert.ToInt32(Console.ReadLine());
            }
            Console.WriteLine(Utility.getLevel(arr));
            break;
        default:
            Console.WriteLine("Invalid Option");
            break;
        }
        //CODE END

     }
    }

}
```

Lab 5

1) Create a class Passenger with following public attributes

Pid: string

Name:string

Email:string

ContactNo:int

2) Create a class Cab with following public attributes

Cabid: string

RegNo:string

Type :string

Capacity :int

CostPerKm:double

PassengerList: List<Passenger>

Note : Here the Type variable is used to store type of cab like Micro,Mini,Shared etc.Capacity is used to store the total number of passengers that a cab can accommodate .

3) Add default constructor and parameterised constructor to initialize all member fields of the class.The order of initialization is as given below :

Cab(string cabid,string regno,string type,int capacity,double cost,List<Passenger> paslist)

4)Create the following methods in the Cab class

· Public void AddPassengerToCab(Passenger p) : This method accepts a passenger object and adds the passenger to the passenger list of the current cab .If the passenger with same id and name already available in the list , then passenger should not be added to the list.

· Public bool RemovePassengerFromCab(string id) :This method will get the id of the passenger and deletes the passenger with specified id from the current Cab.If a passenger with a given id is found , then deletes the passenger and returns true.If paasenger with id is not found then returns false.

5)Create a CabBO class and add the below methods:

· public List<Cab> FindCab(List<Cab> cabList , string type) : This method accepts a list of cabs and cab type as input parameter..This method will search cabs from cabList and returns a list of cabs that matches the given type .

· public List<Cab> FindPCab(List<Cab> cabList , int capacity) : This method accepts a list of cabs and capacity as input parameter.This method will search cabs from cabList and returns a list of cabs that matches the given capacity .

· public HashTable CapacityWiseCount(List<Cab> cabList) : This method accepts a list of cabs and returns an object that contains capacity-wise count of cabs.

```
using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;
```

```
class Passenger
   {
      public string Pid;
public string Name;
public string Email;
public int ContactNo;


   }


   class Cab
   {
      public string Cabid;
public string RegNo;
public string Type ;
public int Capacity;
     public double CostPerKm;
      public List<Passenger> PassengerList;
      public Cab() { }
      public Cab(string cabid, string regno, string type, int capacity, double cost,
List<Passenger> paslist)
      {
         this.Cabid = cabid;
         this.RegNo = regno;
         this.Type = type;
         this.Capacity = capacity;
         this.CostPerKm = cost;
         this.PassengerList = paslist;
      }

      public void AddPassengerToCab(Passenger p)
      {
         Passenger obj = PassengerList.FirstOrDefault(x => x.Pid == p.Pid & x.Name
== p.Name);
         if (obj == null)
             PassengerList.Add(p);


      }
      public bool RemovePassengerFromCab(string id)
      {
         int count = PassengerList.FindAll(x => x.Pid == id).Count();
         bool flag = false;
         if(count > 0)
         {
             flag = true;
             Passenger obj = PassengerList.FirstOrDefault(x => x.Pid == id);
             PassengerList.Remove(obj);
         }
         return flag;

      }



   }

   class CabBO
   {

      public List<Cab> FindCab(List<Cab> cabList, string type)
      {
         List<Cab> clist1 = cabList.FindAll(x => x.Type == type);
         return clist1;
      }
      public List<Cab> FindCab(List<Cab> cabList, int capacity)
      {
         List<Cab> clist1 = cabList.FindAll(x => x.Capacity == capacity);
         return clist1;

      }

      public Hashtable CapacityWiseCount(List<Cab> cabList)
      {
         Hashtable ht = new Hashtable();
         var res = from c in cabList
                   group c by c.Capacity into grp
                   select new { capacity = grp.Key, count = grp.Count() };
         foreach( var x in res)
         {
             ht.Add(x.capacity, x.count);
         }
         return ht;

      }
   }
```