Clearance Sale

Grade settings: Maximum grade: 100 Run: Yes Evaluate: Yes Automatic grade: Yes Maximum execution time: 60 s

Scenario :

As part of the boxing day celebration, Thomas Households Ltd has decided to go for a clearance sale for 3 days. The price of the product will be reduced by 7% on each day.

Requirement :

1. User must be able to input the name of the product, price of the product, whether the product is on sale or not, number of that product sold on day 1, day2 and day3.

2. Calculate the sale price based on the given condition.

Condition 1: Product is on sale

Condition Example

7% discount on Day 1 if a product price is Rs.10000 , then it will cost Rs.9300

7% discount on Day 2 on discounted price Rs.8649 which is 7% less from Rs.9300

7% discount on Day 3 on discounted price Rs. 8043 which is 7% less from Rs.8649

Condition 2: Product is NOT on sale

Calculate the total day sales of the product on day 1, day 2 and day 3.

Sample input 1:

Enter the name of the product : Samsung 215 L

Enter the price of the product : 10000

Is the product on SALE (yes/no) :

yes

Enter number of product sold in day 1

2

Enter number of product sold in day 2

3

Enter number of product sold in day 3

3

Sample output 1:

Samsung 215 L Day 1 sales total : 18600 Day 2 sales total : 25947 Day 3 sales total : 24130

Sample input 2:

Enter the name of the product : Samsung 215 L

Enter the price of the product : 10000

Is the product on SALE (yes/no) :

no

Enter number of product sold in day 1

2

Enter number of product sold in day 2

3

Enter number of product sold in day 3

3

Sample output 2:

Samsung 215 L Day 1 sales total : 20000 Day 2 sales total : 30000 Day 3 sales total : 30000

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProgFundamentals7              //DO NOT CHANGE the name of namespace
```

```
{
  public class Program                  //DO NOT CHANGE the name of class 'Program'
  {
    public static void Main(string[] args)     //DO NOT CHANGE 'Main' Signature
    {
      //Implement the code here
      Console.WriteLine("Enter the name of the product :");
      string name=Console.ReadLine();
      Console.WriteLine("Enter the price of the product :");
      int price=int.Parse(Console.ReadLine());
      Console.WriteLine("Is the product on SALE (yes/no) :");
      string sale=Console.ReadLine();
      Console.WriteLine("Enter the number of product sold in day1");
      int day1=int.Parse(Console.ReadLine());
      Console.WriteLine("Enter the number of product sold in day2");
      int day2=int.Parse(Console.ReadLine());
      Console.WriteLine("Enter the number of product sold in day3");
      int day3=int.Parse(Console.ReadLine());
      if(sale=="yes")
      {
        Console.WriteLine(name);
        double day1sale=(day1*price*0.93);
        Console.WriteLine("Day 1 sales total : {0}",day1sale);
        double day2sale=(day2*price*0.93*0.93);
        Console.WriteLine("Day 2 sales total : {0}",day2sale);
        double day3sale=(day3*price*0.93*0.93*0.93);
        Console.WriteLine("Day 3 sales total : {0}",day3sale);
      }
      if(sale=="no")
      {
        Console.WriteLine(name);
        double day1sale=(day1*price);
        Console.WriteLine("Day 1 sales total : {0}",day1sale);
        double day2sale=(day2*price);
        Console.WriteLine("Day 2 sales total : {0}",day2sale);
        double day3sale=(day3*price);
        Console.WriteLine("Day 3 sales total : {0}",day3sale);
      }
    }
  }
}
```

Design The Display

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario :

Talent Edge Consultancy wants to create a printout template. They want to add a header and footer message in the letter heads

generated for their Enrolment.

This message should display for all the documents generated.

Method Name Argument Return

Type

Access

Specifier

Responsibilities

headerMessage string void public headerMessage' must accept 'company

name' as argument and display the

output .

footerMessage int void public 'footerMessage' must accept 'year' as

argument and display the output .

entireMessage string,string,string,intvoid public 'entireMessage' must accept 'Company

name', 'Employee name', 'Project

Name', 'Year' as arguments and display

the output .

Important : Keep all the method as ‘Public’ and ‘Static’

Requirement :

1. In the main method get the four input from user . i.e Company Name , Employee Name , Project Name and Year .

2. Construct the headerMessage method by passing the Company Name.

3. Construct the footerMessage method by passing the Year .

Sample Input/Output

Sample Input : 1

Enter the Company Name

Microsoft

Enter the Employee Name

John

Enter the Project Name

Roxy

Enter the Year

2018

Sample output

Microsoft Employee Information

John

Roxy

Sample Input : 2

Enter the Company Name

Apple

Enter the Employee Name

Patrick

Enter the Project Name

IMusic

Enter the Year

2019

Sample output

Apple Employee Information

Patrick

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FunctionEx1            //DO NOT Change the namespace name
{
    public class Program         //DO NOT Change the class 'Program' name
```

```
  {

    public static void Main(string[] args)  //DO NOT Change 'Main' method signature
    {
      //Implement the code here
      Console.WriteLine("Enter the Company Name");
      string cname=Console.ReadLine();
      Console.WriteLine("\nEnter the Employee Name");
      string ename=Console.ReadLine();
      Console.WriteLine("\nEnter the Project Name");
      string pname=Console.ReadLine();
      Console.WriteLine("\nEnter the Year");
      int year=Convert.ToInt32(Console.ReadLine());
     entireMessage(cname,ename,pname,year);

    }
    public static void headerMessage(string cname)
    {
       Console.WriteLine(cname+" Employee Information");
    }
    public static void footerMessage(int year)
    {
       Console.WriteLine("CopyRight {0}\nAll Rights Reserved.",year);
    }
    public static void entireMessage(string cname, string ename,string pname,int year)
    {
       headerMessage(cname);
       Console.WriteLine(ename);
       Console.WriteLine(pname);
       footerMessage(year);

    }
    //Implement the required methods. Keep ALL the methods 'public' and 'static'
  }
}
```

Maximum Enrolement

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario :

MCIT University measures the current technology trends based on the number of enrolments in different departments. There are

3 departments, CS, MECH and MET.

Requirement :

1. Get the total students enrolled in each department. Get the value for CS, then MECH, and then MET (this order is

important).

2. Find out which department has the maximum enrolment.

Note :

If two departments has the same maximum number then display both the departments.

Sample Input/Output

Sample Input 1:

Total students placed in CS : 110

Total students placed in MECH : 90

Total students placed in MET : 80

Sample Output :

Highest placement CS

Sample Input 2:

Total students placed in CS : 85

Total students placed in MECH : 120

Total students placed in MET : 110

Sample Output :

Highest placement MECH

Sample Input 3:

Total students placed in CS : 110

Total students placed in MECH : 100

Total students placed in MET : 110

Sample Output :

Highest placement CS

Highest placement MET

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Departments          //DO NOT CHANGE the name of namespace
{
   public class Program            //DO NOT CHANGE the name of class 'Program'
   {
      public static void Main(string[] args)    //DO NOT CHANGE 'Main' Signature
      {
```

```
        Console.WriteLine("Total students placed in CS ");
        int cs=Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Total students placed in MECH ");
        int mech=Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Total students placed in MET ");
        int met=Convert.ToInt32(Console.ReadLine());

        if(cs>=mech && cs>=met)
        {
          Console.WriteLine("Highest placement cs");
        }
        if(mech>=cs && mech>=met)
        {
         Console.WriteLine("Highest placement mech");
        }
        if(met>=cs && met>=mech)
        {
           Console.WriteLine("Highest placement met");
        }
    }
  }
}
```

RectangularArray

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario :

Tom went to participate in a competition. The task given to him is to draw a pattern as specified below.

Prompt the user to enter a number and Display the pattern.

Method Name Argument Return

Type

Access

Specifier

Responsibilities

GetArray int int [,] public This method should accept an

integer and return a 2-dimensional array

of N * N matrix of the specified pattern.

Ex : If the number entered is '5' then

create 5 X 5 matrix.

Ex : If the number entered is '3' then

create 3 X 3 matrix.

Important : Keep all the method as ‘Public’ and ‘Static’

Requirement :

Display a matrix

1. The diagonal of the matrix fills with 0

2. The lower triangular side(below the diagonal) is filled with -1

3. The upper triangular side(above the diagonal) is filled with 1

Sample Input/Output

Sample Input : 1

Enter a number:

3

Sample Output :

0 1 1

-1 0 1

-1 -1 0

Sample Input 2:

Enter a number

4

Sample Output :

0 1 1 1

-1 0 1 1

-1-10 1

-1-1-10

```
using System;

public class Program   //DO NOT change the class name
{
   static void Main(string[] args)      //DO NOT change the 'Main' method signature
   {
      //Fill code here
      Console.WriteLine(" enter a number: ");
      int n=int.Parse(Console.ReadLine());
      int[,] output=GetArray(n);
      for(int i=0;i<n;i++)
      {
         for(int j=0;j<n;j++)
         {
            Console.Write(output[i,j]+" ");
         }
         Console.WriteLine();
      }
   }

   public static int[,]  GetArray(int num)
   {  int[,] mat=new int[num,num];
      //Fill code here
      for(int i=0;i<num;i++)
      {
         for(int j=0;j<num;j++)
         {
           if(i>j)
           mat[i,j]=-1;
          else if(i<j)
           mat[i,j]=1;
           else if(i==j)
           mat[i,j]=0;
         }
      }
      return mat;
   }
```

}

Appointment Booking

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes

Important Instructions:

· Please read the document thoroughly before you code.

· Import the given skeleton code into your Visual Studio.

· Do not change the Skeleton code or the package structure, method names, variable names, return types, exception clauses,

access specifiers etc.

· You can create any number of private methods inside the given class.

· You can test your code from Main() method of the program

Time: 2 hours

Assessment Coverage:

· Classes and Objects

· Variables

· Loops

· Decision Constructs

· Collections

Problem Statement: Booking Doctor's Appointment

You are required to develop a C# Console Application to request a doctor's appointment.

Requirement 1: Create a public class with the following properties.

Class Name: PatientDetail

Property Name Data type

PatientName string

PatientAge int

Note: The above properties must be public

Requirement 2: Include the following methods in the PatientDetail class.

Method Signature Description

public static List<string>

GetDepartments()

Create a Method called “GetDepartments” which

returns the list of departments in the

“PatientDetail” class with the below values to

represent the departments.

public static List<string>

GetDoctors(int option)

Create another method GetDoctors which

accepts the department as option and displays

the list of doctors (Hint: Use Switch Case)

Refer the Below Table for the List of Departments and Respective Doctors:

Department Doctors

ENT Dr. Murugadoss, Dr. Kalaivani

Gynecology Dr. Abirami, Dr. Lakshmi, Dr.

Revathi

Cardiology Dr. Amudhan, Dr. Gunaseelan, Dr.

Agarwal

Neurology Dr. Natarajan, Dr. Nanda, Dr.

Keerthi

Nephrology Dr. Ashirvatham, Dr. Cherian, Dr.

Ashwath, Dr. Ram

Requirement 3: Create a public class with the following method to verify the appointment request functionality:

Class Name: AppointmentDateVerification

Method Signature Description

public

string CheckappointmentRequestDate(DateTime

appointmentRequestDate)

This method checks if the input

appointment request date is a

future date, from the current year

and the day of the date is not a

Monday. If all 3 conditions are true

it returns “Appointment Confirmed!”

along with a randomly generated

number as Patient ID.

Else returns –

1. “Appointment Rejected, Date

must be a future date!”

2. “Appointment Rejected, You

can book appointment only for the

current year!”

3. "Sorry!!! Appointment cannot

be given on Monday!"

Requirement 4: In the Main Program, Get

Program.cs

```
using System;
using System.Collections.Generic;

namespace Appointment_Booking_Application
{

    public class Program
    {
        public static void Main()
        {
            bool isValidAppointmentDate;
            AppointmentDateVerification a1= new AppointmentDateVerification();

            try
            {
                PatientDetail patientDetail = new PatientDetail();

                string doctorname=string.Empty;

                Console.Write("Patient Name: ");
                patientDetail.PatientName = Console.ReadLine();
                Console.Write("Patient Age: ");
                patientDetail.PatientAge = Convert.ToInt32(Console.ReadLine());

                var departments = new PatientDetail().GetDepartments();
                Console.WriteLine("\nDepartments List\n");
                foreach (var item in departments)
                {
                    Console.WriteLine(item);
                }

                Console.Write("\nChoose the department number from the above list (1-5) : ");
                int option = Convert.ToInt32(Console.ReadLine());
                string department = departments[option - 1].Substring(2);
                bool isValidDoctor=true;
                do
                {


                    List<string> doctors = new PatientDetail().GetDoctors(option);
```

```
            if (doctors.Count>0)
            {
               Console.WriteLine("\nDoctors in the {0} deartment\n", department);
               foreach (var item in doctors)
               {
                  Console.WriteLine(item);
               }

               Console.Write("\nDoctor Name : ");

               doctorname = Console.ReadLine();

               if (!doctors.Contains(doctorname))
               {
                  isValidDoctor = false;
                  Console.WriteLine("{0} not found in our list", doctorname);
               }
               else
               {
                  isValidDoctor = true;
               }

            }

         } while (!isValidDoctor);

         do
         {
            Console.Write("\nAppointment Request Date (MM/dd/yyyy) : ");
            DateTime appointmentRequestDate = Convert.ToDateTime(Console.ReadLine());
            string s=new
AppointmentDateVerification().CheckAppointmentRequestDate(appointmentRequestDate);
            if (s=="Appointment Confirmed!")
            {
               Console.WriteLine(s);
               Random rd=new Random();
               int id=rd.Next(int.MaxValue,int.MinValue);
               Console.WriteLine("Patient Id-"+id);
               Console.WriteLine("Please Contact "+doctorname+" on "+appointmentRequestDate);
               isValidAppointmentDate=true;
            }
            else
```

```
                {
                  Console.WriteLine(s);
                   isValidAppointmentDate=false;

                }
                } while (!isValidAppointmentDate);

          }
          catch (Exception e)
          {
             Console.WriteLine(e.Message);
          }
       }
    }
}
```

Patient details.cs

```
using System.Collections.Generic;

namespace Appointment_Booking_Application
{
   public class PatientDetail
   {
      public string PatientName;
      public int PatientAge;
      public List<string> GetDepartments()
      {
         List<string>list=new List<string>();
         list.Add("ENT");
         list.Add("Gynecology");
         list.Add("Cardiology");
         list.Add("Neurology");
         list.Add("Nephrology");
         return list;
      }
      public List<string> GetDoctors(int option)
      {
        List<string>list1=new List<string>();
```

```
            switch (option)
            {case 1:
             list1.Add("Dr. Murugadoss");
             list1.Add("Dr. Kalaivani");
             break;
             case 2:
             list1.Add("Dr. Abirami");
             list1.Add("Dr. Lakshmi");
             list1.Add("Dr. Revathi");
             break;
             case 3:
             list1.Add("Dr. Amudhan");
             list1.Add("Dr. Gunaseelan");
             list1.Add("Dr. Agarwal");
             break;
             case 4  :
             list1.Add("Dr. Natarajan");
             list1.Add("Dr. Nanda");
             list1.Add("Dr. Keerthi");
             break;
             case 5:
             list1.Add("Dr. Ashirvatham");
             list1.Add("Dr. Cherian");
             list1.Add("Dr. Ashwath");
             list1.Add("Dr. Ram");
             break;
            }
            return list1;
        }
    }
}
```

AppointmentDateVerification.cs

```
using System;

namespace Appointment_Booking_Application
{
  public class AppointmentDateVerification
  {
     public string CheckAppointmentRequestDate(DateTime appointmentRequestDate)
```

```
    {
        DateTime dt=DateTime.Today;
        DayOfWeek dw=appointmentRequestDate.DayOfWeek;
        if(appointmentRequestDate>dt)
        {
            if(appointmentRequestDate.Year==dt.Year)
            {
                if(dw!=DayOfWeek.Monday)
                {
                    return "Appointment Confirmed!";

                }
                else
                {
                    return "Sorry!!! Appointment cannot be given on Monday!";
                }
            }
                else
                {
                return "Appointment Rejected, You can book appointment only for the current year!";
                }
        }
            else
             {
            return   "Appointment Rejected, Date must be a future date!";
            }
    }
  }
}
```

Bank Account Detail

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes

Scenario:

Doon Software Company wants to help the Bank by parsing the account details from the text file.

Each line in the file is an account detail with comma separated values.

Each line in the text file has three attributes , 1. Account Number 2.Account Name 3.Balance.

Read the file line by line.

1. Append the each read line to variable "FileContent" (this variable is already declared).

2. Separate the each line read into attributes based on the comma values and assign that to a SavingsAccount object.

3. Add each object to the list "SavingsList".

Example:

1000 , Ram , 34000

Separate the values and assign,

1000 and to Account Number

Ram to Account Name

34000 to Balance

Requirement:

1. Class SavingsAccount with required Properties and Constructor are given .

2. Class Program

Methods:

Method Name Argument

Return

type

Access

Specifier

Responsibilities

GetAccountDetails String filenamevoid public

static

This method accepts

the filename as a string and

store the information to the

SavingsAccount object.

Store the each

SavingsAccount object to the

given list,

List<SavingsAccoun>

SavingsList = new

List<SavingsAccount>();

Hint:

The text file name is details.txt.

Note: The text file details.txt contains

5000 , Rahul , 45000

1001 , Ram , 55000

1002 , John , 72000

Sample Input/Output:

Account Information

All Account Details Added To List Successfully

Program.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BankingApp //DO NOT change the namespace name
{
    public class Program //DO NOT change the class name
    {
        public static string FileContent;
        public static List<SavingsAccount> SavingsList = new List<SavingsAccount>();

        static void Main(string[] args)
        {
            string filename="details.txt";
            GetAccountDetails(filename);
            Console.WriteLine("Account Information \nAll Account Details Added To List Successfully");
```

```
            //Fill code here
        }

        public static void GetAccountDetails(String filename)  //DO NOT change the method signature
        {
            string[] lines=File.ReadAllLines(filename);
            foreach(var line in lines)
            {
                string[] accDetails=line.Split(',');
                SavingsList.Add(new
SavingsAccount(Int32.Parse(accDetails[0]),accDetails[1],Int32.Parse(accDetails[2])));
            }

            //Fill Code here
        }

    }
}
```

SavingsAccount.cs

```
/THIS IS FOR REFERENCE ONLY. YOU ARE NOT REQUIRED TO MAKE ANY CHANGES HERE
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace BankingApp
{
    public class SavingsAccount
    {
        public int AccountNumber { get; set; }
        public string AccountName { get; set; }
        public int Balance { get; set; }

        public SavingsAccount(int AccountNumber , string AccountName,int Balance)
        {
            this.AccountNumber = AccountNumber;
            this.AccountName = AccountName;
            this.Balance = Balance;
```

```
    }

    public SavingsAccount(){}
  }
}
```

details.txt

```
5000,Rahul,45000
1001,Ram,55000
1002,John,72000
```

CompanySalesDetailsApp

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes

Instructions

· The duration of this challenge is 120 Minutes.

· Programming questions have a Compile and Run option where you can run your solution against sample test cases before

submitting it.

· Click Evaluate button only if your code compiles successfully.

· This challenge covers the following topic(s).

· Conditional constructs

· Looping

· Arrays

· Generic Collections

· List<object>, Dictionary<TKey,TValue>, SortedDictionary<TKey,TValue>, IEnumeration<KeyValuePair<TKey, TValue>>

Problem : CompanySalesDetailsApp

Software companies today operate on many different business models and provide a wide array of products and services.

Revenue generated from these services includes software license sales, maintenance services, subscription fees, support

services, and more.

You are given a sequence of n companies in format <company_name>:<annual_sales in billon$>:<service>.

Example:

Microsoft:103.3:Desktop

Oracle:39.5:Could Services

SAP:27.4:Cloud Services

Microsoft:10.3:Software

VMWare:7.9:Virtualization

Oracle:6.6:License Support

SAP:7.5:Software

Adobe:7.7:Digital Media

Write a program that prints all companies in alphabetical order. For each company print their annual sales amounts from all the

services they provide.

Print the result in the following format:

<company> : <annual sales in billon$>

p y _$

For the orders above the output should be:

Adobe : 7.7

Microsoft : 113.6

Oracle : 46.1

SAP : 34.9

VMWare : 7.9

Add new class called Service with the following public properties.

Name of type string

AnnualSalesAmount of type float

Create another class called CompanySalesRevenue with the following methods.

Method Name Input Argument Return Type Description

GetCompanySalesDetails List<string> Dictionary<string, List<Service>> This function accepts a l

company details, returns

which it arranges the co

from the input argument

that company name is th

services is the value.

DisplayCompaniesRevenueInOrderDictionary<string,

List<Service>>

IEnumerable<KeyValuePair<string,

float>>

This function accepts a d

company details which is

from GetCompanySales

and process it, finally ret

company name and the

annual sales revenue fro

services.

Input

The input comes from the console.

At the first line the number n indicates the number of company details.

At the next n lines, we have n company details in format <company>:<annual_sales in billon$>:<service>

Assume that the input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print one line for each company. Company lines should be ordered in alphabetical order.

Each line should be in format <company> : <annual_sales in billon$>

Sample Input

5

Salesforce.com:10.5:SaaS

HCL:7.8:Application Integration Services

HCL:2.3:Infrastructure Management Services

HCL:3.4:Cybersecurity

Intuit:5.4:Personal and Small Business Financial Management

Sample Output

HCL : 13.5

Intuit : 5.4

Salesforce.com : 10.5

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CompanySalesDetailsApp
{
   class Program
   {
      static void Main()
      {
         int numberOfRecords = int.Parse(Console.ReadLine());

         List<string> inputs = new List<string>();

         for (int i = 0; i < numberOfRecords; i++)
         {
            inputs.Add(Console.ReadLine());
         }

         //Write your code here
         CompanySalesRevenue companySalesRevenue=new CompanySalesRevenue();
         var companyDetailsDict=companySalesRevenue.GetCompanySalesDetails(inputs);
         var
companiesRevenueDict=companySalesRevenue.DisplayCompaniesRevenueInOrder(companyDetailsDict);
```

```
            foreach(var item in companiesRevenueDict)
            {
                Console.WriteLine(item.Key+":"+item.Value);
            }
        }
    }
}
```

Service.cs

```
namespace CompanySalesDetailsApp
{
    public class Service
    {
        //Implement the Properties
        public string Name{get;set;}
        public float AnnualSalesAmount{get;set;}
    }

}
```

CompanySalesRevenue.cs

```
using System.Collections.Generic;
using System.Linq;

namespace CompanySalesDetailsApp
{
    public class CompanySalesRevenue
    {
        public Dictionary<string, List<Service>> GetCompanySalesDetails(List<string> _companySalesData)
        {
            Dictionary<string, List<Service>> companyDict = new Dictionary<string, List<Service>>();

            for (int i = 0; i < _companySalesData.Count; i++)
            {
                if (!companyDict.ContainsKey(_companySalesData[i].Split(':')[0]))
                {
                    companyDict.Add(_companySalesData[i].Split(':')[0], new List<Service> { new Service { Name =
_companySalesData[i].Split(':')[2], AnnualSalesAmount = float.Parse(_companySalesData[i].Split(':')[1]) } });
                }
```

```
            else
            {
                companyDict[_companySalesData[i].Split(':')[0]].Add(new Service { Name =
_companySalesData[i].Split(':')[2], AnnualSalesAmount = float.Parse(_companySalesData[i].Split(':')[1]) });
            }
        }

        return companyDict;
    }

    public IEnumerable<KeyValuePair<string, float>> DisplayCompaniesRevenueInOrder(Dictionary<string,
List<Service>> dictCompaniesSalesDetails)
    {

        SortedDictionary<string, float> companiesInOrderDict = new SortedDictionary<string, float>();

        foreach (var item in dictCompaniesSalesDetails)
        {
            companiesInOrderDict.Add(item.Key, item.Value.Sum(x => x.AnnualSalesAmount));
        }


        IEnumerable<KeyValuePair<string, float>> temp = companiesInOrderDict;


        return temp;
    }
  }
}
```

CustomerUtility

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

Let’s consider a system maintaining all the customer details.

Requirement:

Create a menu to add, delete, search and display the customer’s details which includes customer id, name and city. Use the

concept of Collections and Generics.

1. Create a class called CustomerUtility which maintains a list of customers(Create a generic List that can accept only

customer objects)

List<Customer> custList;

2. Create a property for the above field.

3. CustomerUtility class should provide methods for adding, and deleting a customer and displaying the customer details.

Methods: (To be implemented inside class “CustomerUtility”)

Method Name Argument Return type

Access

Specifier

Responsibilities

AddCustomer Customer List<Customer> public This method should

add customer details

SearchCustomerByID int Customer public This method should

search for the given

customer id from the

list and return the

customer object.

DisplayAllCustomers N/A void public This method should

display all the

customer details from

the List

DeleteCustomer int void public This method should

delete the customer

from the list when the

customer id is

passed.

Important Note:

1. All the four methods should operate on the same CustList property that maintains the customer objects.

2. Create a class Program for Main method. From the main method create a menu to add, delete, search and display

the customers details.

3. Don not create any new namespaces

4. Declare the classes as public.

Sample Input/Output:

S.No Sample Input & Output

1 1. Add Customer

2. Display Customer

3. Search Customer

4.Delete Customer

Enter Your Choice

1

Enter Customer id, name and city

101

Samantha

Chennai

Enter your Choice

1

Enter Customer id, name and city

102

Dinesh

Pune

Enter your choice

2

101 Samantha Chennai

102 Dinesh Pune

Enter your choice

3

101 Samantha Chennai

Enter your choice

4

102

```
using System;
```

```
using System.Collections.Generic;

/** DO NOT alter/delete this partial code snippet. This is given for your reference **/

public  class Customer
  {
    int id;
    String name;
    String city;

    public int Id { get; set; }
    public String Name { get; set; }
    public String City { get; set; }

    public override string ToString()
    {
      return Id + " " + Name + " " + City;
    }

  }
  public class CustomerUtility{
    public List<Customer> custList = new List<Customer>();
    public List<Customer> CustList{
      get{
        return custList;
      }
      set{
        custList = value;
      }
    }
    public List<Customer> AddCustomer(Customer cust){
      custList.Add(cust);
      return custList;
    }
    public Customer SearchCustomerByID(int id){
      for(int i = 0;i<custList.Count;i++){
        if(custList[i].Id == id){
          return custList[i];
        }
      }
      return null;
```

```
    }
    public void DisplayAllCustomers(){
        foreach(Customer c in custList){
            Console.WriteLine(c.ToString());
        }
    }
    public void DeleteCustomer(int id){
        for(int i =0; i<custList.Count;i++){
            if(custList[i].Id == id){
                custList.Remove(custList[i]);
            }
        }
    }
}
public class Program{
 static void Main(string[] args){
    CustomerUtility cU = new CustomerUtility();
    Console.WriteLine("1.Add Customer");
    Console.WriteLine("2.Display Customer");
    Console.WriteLine("3.Search Customer");
    Console.WriteLine("4.Delete Customer");
    for(int i=0;i<3;i++){
        Console.WriteLine("Enter Your Choice");
        int choice = int.Parse(Console.ReadLine());
        if(choice == 2){
                Console.WriteLine("Enter Customer id, name and city");
                int id = int.Parse(Console.ReadLine());
                string name = Console.ReadLine();
                string city = Console.ReadLine();
                Customer c = new Customer();
                c.Id = id;
                c.Name = name;
                c.City = city;
                cU.custList.Add(c);
        }else if(choice == 2){
                cU.DisplayAllCustomers();
        }else if(choice == 3){
                int cid = int.Parse(Console.ReadLine());
                Customer cc = cU.SearchCustomerByID(cid);
                Console.WriteLine(cc.ToString());
         }else if(choice == 4){
```

```
            int cuid = int.Parse(Console.ReadLine());
            cU.DeleteCustomer(cuid);
        }

    }
}
}
```

Double Discount Offer

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

ANS Super Market has announced a double dhamaka discount sale for the festival season.

Requirement:

Calculate the discount available based on the purchase amount and deduct the amount specified for the membership card

holders and deduce the final payable amount.

Condition 1: (Discount based on purchase amount)

Purchase amount

Discount in percentage of

the purchase amount

>=10000 50%

>=5000 10%

>=3000 5%

Condition 2: (Discount based membership for card holders alone)

There are three types of membership cards Gold, Silver and Bronze. Null represents no membership card

Card Type Amount Reduced

Gold 1000

Silver 500

Bronze 200

Null 0

Properties: (To be initialized inside class “Program”)

· String CustomerName // stores the customer names

· double PurchaseAmount // stores the purchase amount made by the customer

· CardType MemberCardType // Type of membership card

· double PayableAmount // The net amount the customer has to pay finally after discounts

Methods: (To be implemented inside class “Program”)

Method Name Argument

Return

type

Access

Specifier

Responsibilities

CalculateDiscount N/A double public Calculate the discount
amount and subtracts the
discount amount from the
purchase amount and
returns the result.

CaluclateNetPayableAmountCardType ,

double

double public The first input parameter
is CardType, and second
input is the amount
obtained after the
reduction. This method
calculates the second
discount based on the
membership card type
and returns the net
payable amount.
DisplayDetails N/A void public Display the details of

purchase as shown in the

sample input output.

Important Note:

1. In “Program.cs”, Create an enum named CardType that stores the different types of membership cards. Only the below

given values should be stored as enum values.

§ Gold, Silver, Bronze and Null.(case sensitive values)

2. In “Program.cs”, Create a struct named Purchase to hold the given auto implemented Properties and methods

3. In class “Program”, the Main method should get inputs for the struct properties

(CustomerName, PurchaseAmount and MemberCardType) and assign values. Call the methods to calculate the net payable

amount and display the details.

Sample Input/Output:

S.No Sample Input Sample Output

1

Enter customer name

Sam

Enter purchase amount

10000

Enter card type

Gold

Customer name : Sam

Purchase amount : 10000

Card type : Gold

Net payable amount: 4000

2

Enter customer name

Diana

Enter purchase amount

12000

Enter card type

Null

Customer name : Diana

Purchase amount : 12000

Card type : Null

Net payable amount: 6000

3

Enter customer name

Joe

Enter purchase amount

4000

Enter card type

Silver

```
using System;
```

```
namespace Structures1          //DO NOT CHANGE the namespace name
{
  //Declare enum outside the class
  public enum CardType { Gold, Silver, Bronze, Null};
  //Declare struct outside the class
   public struct Purchase
   {
      public string CustomerName { get; set; }
      public double PurchaseAmount { get; set; }
      public CardType MemberCardType { get; set; }
      public double PayableAmount { get; set; }
      public double CalculateDiscount()
      {
         if(PurchaseAmount >= 10000)
            return PurchaseAmount - PurchaseAmount * 0.5;
         else if(PurchaseAmount >= 5000)
            return PurchaseAmount - PurchaseAmount * 0.1;
         else if(PurchaseAmount >= 3000)
            return PurchaseAmount - PurchaseAmount * 0.05;
         return PurchaseAmount;
      }
      public void DisplayDetails()
      {
         Console.WriteLine($"Customer name : {CustomerName}");
         Console.WriteLine($"Purchase amount : {PurchaseAmount}");
         Console.WriteLine($"CardType : {MemberCardType}");
         Console.WriteLine($"Net payable amount: {PayableAmount}");
      }
       public double CaluclateNetPayableAmount(CardType cardType, double discountedAmount)
      {
         if(cardType == CardType.Gold)
            return discountedAmount - 1000;
         else if(cardType == CardType.Silver)
            return discountedAmount - 500;
         else if(cardType == CardType.Bronze)
            return discountedAmount - 200;
         return discountedAmount;
      }
  }
```

```
public class Program     //DO NOT CHANGE the class 'Program' name
 {


    static void Main(string[] args)        //DO NOT Change 'Main' method Signature
    {
      //Implement code here
      Purchase purchase = new Purchase();
      Console.WriteLine("Enter customer name");
      purchase.CustomerName = Console.ReadLine();
      Console.WriteLine("Enter purchase amount");
      purchase.PurchaseAmount = double.Parse(Console.ReadLine());
      Console.WriteLine("Enter card type");
      purchase.MemberCardType = (CardType)Enum.Parse(typeof(CardType), Console.ReadLine(), true);
      double discountedAmount = purchase.CalculateDiscount();
      purchase.PayableAmount = purchase.CaluclateNetPayableAmount(purchase.MemberCardType,
discountedAmount);
      purchase.DisplayDetails();
    }
  }
}
```

EmployeeCommuteService

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 240 s Maximum memory used: 256 MiB

Instructions

· You need to solve one programming problem in this challenge.

· The duration of this challenge is 120 Minutes.

· Programming questions have a Compile and Run option where you can run your solution against sample test cases

before submitting it.

· Click Evaluate button only if your code compiles successfully.

· This challenge covers the following topic(s).

· Class and Objects

· Functions

· ADO.Net

Scenario

The transport department of your company wants to have an application through which they can add employees commute

records to database, display all commute records from database and also display records of all those employees commute that

are paying the highest commute fare. Help the transport department by creating the application.

Functionalities

· Add employee commute records to database

· Display all employee commute records from database

· Display the records of those employees who are paying the highest commute fare.

Data Design

Table name: tblEmployeeCommute

Column Name Data type Constraints

EmployeeId varchar(100) Primary Key Not Null

EmployeeId varchar(100) Primary Key, Not Null

EmployeeName varchar(100) Not Null

EmployeeType varchar(100) Not Null

TravelDistance float Not Null

CommuteCharge float Not Null

The database connection information is specified in the “App.config” file, which is also provided as part of code skeleton. THIS IS

GIVEN ONLY FOR YOUR REFERENCE. You need NOT change this. The code skeleton will be available in the Tekstac

platform.

Component Specification

Create a model class called EmployeeCommuteDataInfo with the below public auto-implemented properties:

Type (Class) Data TypesProperties

EmployeeCommuteDataInfo

string EmployeeId

string EmployeeName

string EmployeeType

float TravelDistance

float CommuteCharge

Add a default constructor and a parameterized constructor under this class.

Create a class called EmployeeCommuteDataOperations with the below methods:

Type(Class)	Methods	Responsibilities
public bool	This method should accep	
EmployeeCommuteDataOperations		
public bool	AddCommuteRecord(EmployeeCommuteDataInfo commuteData)	This method should accep an EmployeeCommuteDa and execute a sql query to employee commute record database. It returns true if is successful and false if it insert.
public IList<EmployeeCommuteDataInfo>	DisplayCommuteRecords()	This method will retrieve a records from the database

record in EmployeeComm

object. Add these objects t

public IList<EmployeeCommuteDataInfo>

DisplayCommuteRecordsByHigestCommuteFare()

This method will retrieve a

employees commute recor

database who is paying th

fare. Store each record in

EmployeeCommuteDataIn

Add these objects to a 'Lis

More than one employee

the highest fare; so use a

for this method].

Use ADO.NET data access technology to connect to a database, execute commands and retrieve data from the database.

Include a public property of string type and name it as ConnectionString.

The database connection string details is stored in the App.config file and the same should be returned by the ConnectionString

property. The name of the connection string is "EmpCommuteConnection".

Create a class called Program with main method. Call the EmployeeCommuteDataOperations methods under the Program class

and test your application.

Fare Calculation Formula:

Employee Type Travel Distance Fare Calculation

INTERNAL

<=10 commuteFare = travelDistance * 50

>10 and <=20 commuteFare = travelDistance * 60

>20 commuteFare = travelDistance * 70

EXTERNAL

<=10 commuteFare = travelDistance * 70

>10 and <=20 commuteFare = travelDistance * 80

EXTERNAL >10 and < 20 commuteFare travelDistance 80

>20 travelDistance = travelDistance * 90

Business Rules

· Employee Id should be in the following format:-

o The first 3 should be letters (either EXT or INT only) and the next 4 should be numbers (example: EXT1001 is valid, INT001

is valid where as VEN1001 is invalid, ext1001 is invalid, int1001 is invalid). If validation fails return false; else return true.

· The travel distance should be between 2 km and 30 km. If value is entered outside the specified range return false; else

return true.

· Enclose all DML and DQL operations in EmployeeCommuteDataOperations class under try-

catch block.

Program.cs

```
using System;
using System.Text.RegularExpressions;

namespace EmployeeCommuteService
{
   public class Program
   {
      public static void Main()
      {
         Console.WriteLine("Welcome Admin to Office Commute System");
         DisplayMenu();
         Console.WriteLine("\nThank you for using the app. Have a nice day");
      }

      public static void DisplayMenu()
      {
         EmployeeCommuteDataInfo commuteData = new EmployeeCommuteDataInfo();
         EmployeeCommuteDataOperations commuteOperations = new EmployeeCommuteDataOperations();

         string answer = string.Empty;

         do
         {

            Console.WriteLine("\nMenu\n\n1. Add Commute Record\n2. Display All Commute Records\n3. Display
Employees who Pay Highest Commute Fare\n\n");
            Console.Write("Enter Your Option : ");

            int option = int.Parse(Console.ReadLine());
            bool isValidEmployeeId;
            bool isValidTravelDistance;
            bool isValidEmployeeType;


            string employeeId = string.Empty;
            string employeeName = string.Empty;
```

```
string employeeType = string.Empty;
float travelDistance;

switch (option)
{
   case 1:
      do
      {
         Console.Write("Employee Id : ");
         employeeId = Console.ReadLine();

         if (new Program().ValidateEmployeeId(employeeId))
         {
            isValidEmployeeId = true;
         }
         else
         {
            Console.WriteLine("Invalid Employee Id");
            isValidEmployeeId = false;
         }

      } while (!isValidEmployeeId);
      Console.Write("Employee Name : ");
      employeeName = Console.ReadLine();

      do
      {
         Console.Write("Employee Type (External/Internal) : ");
         employeeType = Console.ReadLine();

         if (employeeType.Substring(0,3).ToUpper().Equals(employeeId.Substring(0,3)))
         {
            isValidEmployeeType = true;
         }
         else
         {
            Console.WriteLine("Invalid Employee Type");
            isValidEmployeeType = false;
         }

      } while (!isValidEmployeeType);
```

```
    do
    {
        Console.Write("Travel Distance : ");
        travelDistance = float.Parse(Console.ReadLine());

        if (isValidTravelDistance = new Program().ValidateTravelDistance(travelDistance))
            isValidTravelDistance = true;
        else
        {
            Console.WriteLine("Invalid Travel Distance");
            isValidTravelDistance = false;
        }

    } while (!isValidTravelDistance);

    double commuteCharge = new Program().CalculateFare(employeeType, travelDistance);

    commuteData = new EmployeeCommuteDataInfo
    {

        EmployeeId = employeeId,
        EmployeeName = employeeName,
        EmployeeType = employeeType,
        TravelDistance = travelDistance,
        CommuteCharge = commuteCharge
    };

    var response=commuteOperations.AddCommuteRecord(commuteData);

    if (response)
    {
        Console.WriteLine("Commute Data Added Successfully!!!");
    }
    else
    {
        Console.WriteLine("Error while Adding Commute Data...");
    }

    break;
case 2:
```

```
                Console.WriteLine("\nEmployees Commute Details\n");

                var commuteRecords = commuteOperations.DisplayCommuteRecords();

                if (commuteRecords.Count>0)
                {
                    Console.WriteLine("{0,-15}{1,-20}{2,-20}{3,-20}{4,-20}","Employee Id","Employee Name","Employee Type","Distance","Commute Fare");
                    foreach (var item in commuteRecords)
                    {
                        Console.WriteLine("{0,-15}{1,-20}{2,-20}{3,-20}{4,-20}",item.EmployeeId,item.EmployeeName,item.EmployeeType,item.TravelDistance,item.CommuteCharge.ToString("0.00"));
                    }
                }

                break;
            case 3:
                Console.WriteLine("\nEmployees with Highest Commute Fare\n");
                var commuteRecordsWithHighestFare = commuteOperations.DisplayCommuteRecordsByHigestCommuteFare();

                if (commuteRecordsWithHighestFare.Count > 0)
                {
                    Console.WriteLine("{0,-15}{1,-20}{2,-20}{3,-20}{4,-20}", "Employee Id", "Employee Name", "Employee Type", "Distance", "Commute Fare");
                    foreach (var item in commuteRecordsWithHighestFare)
                    {
                        Console.WriteLine("{0,-15}{1,-20}{2,-20}{3,-20}{4,-20}", item.EmployeeId, item.EmployeeName, item.EmployeeType, item.TravelDistance, item.CommuteCharge.ToString("0.00"));
                    }
                }
                break;
            default:
                Console.WriteLine("Invalid Option");
                break;
        }
        Console.Write("\nDo you want to continue (Yes/No) ? : ");
        answer = Console.ReadLine();

    } while (answer.ToLower().Equals("yes"));

}
```

```
public bool ValidateEmployeeId(string employeeId)
{
    /*Check the employee id format according to the business rule specified in the problem statement. If validation
            fails return false; else return true */
            if (employeeId.Length == 7)
  {
    string a = employeeId.Substring(0, 3);
    if (a == "EXT" || a == "INT")
    {
      char b = Convert.ToChar(employeeId.Substring(3, 1));
      char c = Convert.ToChar(employeeId.Substring(4, 1));
      char d = Convert.ToChar(employeeId.Substring(5, 1));
      char e = Convert.ToChar(employeeId.Substring(6, 1));
      if (char.IsNumber(b) && char.IsNumber(c) && char.IsNumber(d) && char.IsNumber(e))
      {
        return true;
      }
      else return false;
    }
    else return false;
  }
  else return false;
}
public bool ValidateTravelDistance(float travelDistance)
{
   /* Check travel distance according to the business rule specified in the problem statement. If validation
          fails return false; else return true */
          if (travelDistance >= 2 && travelDistance <= 30) return true;
  else return false;
}
public double CalculateFare(string employeeType, float travelDistance)
{
   /* Fill your code here to calculate and return the commute fare according to the Fare Calcuation Formula
          given in the problem statement */
          if (employeeType == "internal" || employeeType == "Internal" || employeeType == "INTERNAL")
  {
    if (travelDistance <= 10)
      return Convert.ToDouble(travelDistance * 50);

    else if (travelDistance > 10 && travelDistance <= 20)
```

```
                return Convert.ToDouble(travelDistance * 60);
            else if (travelDistance > 20)
                return Convert.ToDouble(travelDistance * 70);
        }
        else if (employeeType == "external" || employeeType == "External" || employeeType == "EXTERNAL")
        {
            if (travelDistance <= 10)
                return Convert.ToDouble(travelDistance * 70);
            else if (travelDistance > 10 && travelDistance <= 20)
                return Convert.ToDouble(travelDistance * 80);
            else if (travelDistance > 20)
                return Convert.ToDouble(travelDistance * 90);
        }
        return travelDistance;
    }
  }
}
```

App.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

<startup>

<supportedRuntime sku=".NETFramework,Version=v4.6.1"version="v4.0"/>

</startup>

<connectionStrings>

<add connectionString="Data Source=localhost;Initial
Catalog=OfficeCommuteAppDb;Integrated

Security=true" name="EmpCommuteConnection"/>

</connectionStrings>

</configuration>
```

EmployeeCommuteDataOperations.cs

```
namespace EmployeeCommuteService
{
   /*Fill your code here to create EmployeeCommuteDataInfo class with auto-implemeted properties, default constructor and parameterized constructor*/
   public class EmployeeCommuteDataInfo
   {
      public string EmployeeId { get; set; }
      public string EmployeeName { get; set; }
      public string EmployeeType { get; set; }
      public float TravelDistance { get; set; }
      public double CommuteCharge { get; set; }
      public EmployeeCommuteDataInfo() { }

      public EmployeeCommuteDataInfo(string employeeId, string employeeName, string employeeType, float travelDistance, double commuteCharge)
      {
         this.EmployeeId = employeeId;
         this.EmployeeName = employeeName;
         this.EmployeeType = employeeType;
         this.TravelDistance = travelDistance;
         this.CommuteCharge = commuteCharge;
      }
   }
}
```

EmployeeCommuteDataInfo.cs

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Configuration;

namespace EmployeeCommuteService
{
   public class EmployeeCommuteDataOperations
   {

      //Implement ConnectionString property here
      public string ConnectionString
      {
         get
```

```
    {
        return ConfigurationManager.ConnectionStrings["EmpCommuteConnection"].ConnectionString;
    }
}
public SqlConnection _sqlConn{ get; set;}

public EmployeeCommuteDataOperations(){ }

//Fill your code to implement AddCommuteRecord method here

        public bool AddCommuteRecord(EmployeeCommuteDataInfo commuteData)
        {
            commuteData =  new EmployeeCommuteDataInfo();
            try
            {
                _sqlConn = new SqlConnection(ConnectionString);
                using(_sqlConn)
                {
                    string query = "INSERT INTO tblEmployeeCommute (EmployeeId, EmployeeName,
EmployeeType, TravelDistance) values ('" + commuteData.EmployeeId + "','" + commuteData.EmployeeName +
"','" + commuteData.EmployeeType +"','" + commuteData.TravelDistance +")";
                    SqlCommand cmd = new SqlCommand(query, _sqlConn);
                    _sqlConn.Open();
                    int rowaffected = cmd.ExecuteNonQuery();
                    _sqlConn.Close();
                    if(rowaffected > 0)
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }

                }
            }
            catch(Exception ex)
            {
                return false;
            }
        }
```

```
        //Fill your code to implement DisplayCommuteRecords method here
public IList<EmployeeCommuteDataInfo> DisplayCommuteRecords()
{
   _sqlConn = new SqlConnection(ConnectionString);
   using(_sqlConn)
   {
      try
      {
         string query = "SELECT * FROM tblEmployeeCommute;";
         _sqlConn.Open();
         SqlCommand cmd = new SqlCommand(query, _sqlConn);
         SqlDataReader reader = cmd.ExecuteReader();
         List<EmployeeCommuteDataInfo> emp1 = new List<EmployeeCommuteDataInfo>();
         while(reader.Read())
         {
            EmployeeCommuteDataInfo emp = new EmployeeCommuteDataInfo();
            emp.EmployeeId = reader.GetValue(0).ToString();
            emp.EmployeeName = reader.GetValue(1).ToString();
            emp.EmployeeType = reader.GetValue(2).ToString();
            emp.TravelDistance = (float)(reader.GetValue(3));
            emp.CommuteCharge  = (float)(reader.GetValue(4));
            emp1.Add(emp);

         }
         reader.Close();
         _sqlConn.Close();
         if(emp1 != null)
         {
            return emp1;
         }
         else
         {
            return new List<EmployeeCommuteDataInfo>();
         }
      }
      catch(Exception ex)
               {
                  return new List<EmployeeCommuteDataInfo>();
               }
   }
}
```

```
        //Fill your code to implement DisplayCommuteRecordsByHigestCommuteFare method here
        public IList<EmployeeCommuteDataInfo> DisplayCommuteRecordsByHigestCommuteFare()
        {
          _sqlConn = new SqlConnection(ConnectionString);
          using(_sqlConn)
          {
            try
            {
              string query = "SELECT * FROM tblEmployeeCommute WHERE CommuteCharge IN (SELECT
MAX(CommuteCharge) FROM tblEmployeeCommute);";
              _sqlConn.Open();
              SqlCommand cmd = new SqlCommand(query, _sqlConn);
              SqlDataReader reader = cmd.ExecuteReader();
              List<EmployeeCommuteDataInfo> emp1 = new List<EmployeeCommuteDataInfo>();
              while(reader.Read())
              {
                EmployeeCommuteDataInfo emp = new EmployeeCommuteDataInfo();
          emp.EmployeeId = reader.GetValue(0).ToString();
          emp.EmployeeName = reader.GetValue(1).ToString();
          emp.EmployeeType = reader.GetValue(2).ToString();
          emp.TravelDistance = (float)(reader.GetValue(3));
          emp.CommuteCharge  = (float)(reader.GetValue(4));
          emp1.Add(emp);
              }
              reader.Close();
        _sqlConn.Close();
        return emp1;

            }
            catch(Exception)
            {
              return new List<EmployeeCommuteDataInfo>();
            }
          }
        }

    }
}
```

Filter Employee List

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

TekTag is an organization with employees working at different levels called 'Band'.

Requirement:

Use Lambda Expression to filter the given list containing the employee details in 'Employee' object.

Condition:

Select all the employees who are in band 'B1'.

Note:

1. “Employee” class is already given for your reference.

2. List with employee object is declared and initialized. Keep it as static. Do not change this.

static List<Employee> empList

3. Store the result of select in the variable 'nameList' using Lambda. Keep it as static.

static IEnumerable<Employee> nameList

NOTE:

Do NOT implement any custom methods or use 'Main' to achieve this implementation.

Program.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LambdaEx1        //DO NOT CHANGE the name of namespace
{
   public class Program    //DO NOT CHANGE the name of class 'Program'
   {
      //DO NOT CHANGE the 'empList' declaration and initialization
      static List<Employee> empList = new List<Employee>() {
         new Employee { empId=101, empName = "Rex", band="B4" },
         new Employee { empId=102, empName = "Tom", band="B2" },
         new Employee { empId=103, empName = "Peter", band="B1" }
       };


      static IEnumerable<Employee> nameList=empList.Where(emp=>emp.band=="B1") //Give the Lambda
implemetation here
                                 //and store the result in 'nameList'
                                 //Implement ONLY using Lambda
                                 ;

      public static void Main(string[] args)
      {
         foreach(var value in nameList){
            Console.WriteLine(" "+value);
            Console.ReadLine();
         }
      }
   }
}
```

Employee.cs

```
/*** Read only File for reference only. Do not make any changes ***/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace LambdaEx1
{
   public class Employee
   {

      public int empId { get; set; }
      public string empName { get; set; }
      public string band { get; set; }
   }
}
```

Insurance Calculation - Work with Polymorphism

Grade settings: Maximum grade: 100
Run: Yes Evaluate: Yes
Automatic grade: Yes Maximum execution time: 60 s
Scenario:
Town's famous insurance company has an application for maintaining insurance details of their customers.
Requirement:
Calculate the premium for the "Life Insurance" and "Motor Insurance" policies of their customers.
Implement the following classes :
a) class Insurance
Field Name Datatype Access Specifier
insuranceNo string private
insuranceName string private
amountCovered double private

Property Name Datatype Access Specifier
InsuranceNo string public
InsuranceName string public
AmountCovered double public

b) class MotorInsurance which is a child of class Insurance
Field Name Datatype Access Specifier
idv double private
depPercent float private

Property Name Datatype Access Specifier

Idv double public
DepPercent float public

Method Name Argument Return
Type

Access
Specifier

Responsibilities

calculatePremiumN/A double public Calculates premium based on formula

given below,
Idv = AmountCovered -
((AmountCovered * DepPercent) /
100);
then, the premium is,
3% of the Idv value, return the
calculated premium.

c) class LifeInsurance which is a child of class Insurance
Field Name Datatype Access Specifier
policyTerm int private
benefitPercent float private

Property Name Datatype Access Specifier
PolicyTerm int public
BenefitPercent float public

Method Name Argument Return
Type

Access
Specifier

Responsibilities

calculatePremiumN/A double public Calculate the premium based on

formula given below,
Subtract the BenefitPercent from the
AmountCovered and divide the result
by PolicyTerm
That is,
AmountCovered - ((AmountCovered*
BenefitPercent)/100)) /
PolicyTerm;

Methods in Program.cs
Method
Name

Argument

Return
type

Access
Specifier

Responsibilities

Main string[] args void public static From the user get the following inputs,
insuranceNo, insuranceName,
amountCovered and insurance option
'1' or '2'

If '1' then, it is a Life Insurance, then
additionally ask the following inputs,
policyTerm, benefitPercentage
If '2' then, it is a Motor Insurance, then
additionally ask the following inputs,
depPercentage (depreciation
percentage)

Get the values from user as shown in
sample input and invoke the method
'addPolicy' accordingly.
After getting the values from user,
assign it to Life Insurance or Motor

Insurance object based on the user option. Pass this object to 'addPolicy' method. So that 'addPolicy' will have access to insuranceNo, insuranceName, amountCovered attributes.
If option is 1, create an object for LifeInsurance and call 'addPolicy' Method

If option is 2, create an object for MotorInsurance and call 'addPolicy' Method

addPolicy Insurance ins,
int opt

double public Based on opt,
1 ==> call
the 'calculatePremium' method
using LifeInsurance reference.
2 ==> call
the 'calculatePremium' method
using MotorInsurance reference.
Return the calculated premium.

Sample Input/Output:
S.No Sample Input & Output

1
Insurance Number : LI23456
Insurance Name : KisanVikas
Amount Covered : 1000000
Select
1.Life Insurance
2.Motor Insurance
1
Policy Term : 10
Benefit Percent : 12
Calculated Premium : 88000

2
Insurance Number : MO87657
Insurance Name : MotorPolicy
Amount Covered : 800000
Select
1.Life Insurance
2.Motor Insurance
2
Depreciation Percent : 8
Calculated Premium : 2208

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace InsuranceDetails //DO NOT Change namespace name
{
   public class Program      //DO NOT Change class 'Program' name
   {

      public static void Main(string[] args)
      {
        //Implement code here
        Console.Write("Insurance Number : ");
        string no=Console.ReadLine();
        Console.WriteLine();
        Console.Write("Insurance Name : ");
        string name=Console.ReadLine();
        Console.WriteLine();
        Console.Write("Amount Covered : ");
        double a=Convert.ToDouble(Console.ReadLine());
        Console.WriteLine();
        Program pr=new Program();
        double premium=0.0;
        Insurance ob;
        Console.WriteLine("Select");
         Console.WriteLine("1.Life Insurance");
```

```
      Console.Write("2.Motor Insurance");
      int ch=Convert.ToInt32(Console.ReadLine());
       Console.WriteLine();
       switch(ch)
       {
          case 1:Console.Write("Policy Term :");
          int p=Convert.ToInt32(Console.ReadLine());
          Console.WriteLine();
          Console.Write("Benefit Percent :");
          float n=float.Parse(Console.ReadLine());
          Console.WriteLine();
          ob=new LifeInsurance();
          ob.InsuranceNo=no;
           ob.InsuranceName=name;
           ob.AmountCovered=a;
           ((LifeInsurance)ob).PolicyTerm=p;
           ((LifeInsurance)ob).BenefitPercent=n;
           premium=pr.addPolicy(ob,ch);
           break;
           case 2:Console.Write("Depreciation Percent :");
           float d=float.Parse(Console.ReadLine());
          Console.WriteLine();
          ob=new MotorInsurance();
          ob.InsuranceNo=no;
           ob.InsuranceName=name;
           ob.AmountCovered=a;
           ((MotorInsurance)ob).Idv=0;
           ((MotorInsurance)ob).DepPercent=d;
           premium=pr.addPolicy(ob,ch);
           break;

       }
       Console.WriteLine("Calculated Premium : "+premium);


 }

//Implement required methods here
public double addPolicy(Insurance ins,int opt)
{
   //Implement code here
```

```
        if(opt==1)
        return((LifeInsurance)ins).calculatePremium();
        else
         return((MotorInsurance)ins).calculatePremium();

     }
   }
}
```

Insurance.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//using InsuranceDetails;
namespace InsuranceDetails
//DO NOT Change namespace name
{
   public class Insurance
      {
         private string insuranceNo,insuranceName;
         private double amountCovered;
         public string InsuranceNo
         {
         get{return insuranceNo;}
         set{insuranceNo=value;}

         }
         public string InsuranceName
         {
            get{return insuranceName;}
            set{insuranceName=value;}

         }
         public double AmountCovered
         {
            get{return amountCovered;}
            set{amountCovered=value;}

         }
```

```
    }
 public class MotorInsurance:Insurance
    {

        private double idv;
        private float depPercent;
        public double Idv
        {
           get{return idv;}
           set{idv=value;}

        }
        public float DepPercent
        {
           get{return depPercent;}
           set{depPercent=value;}

        }
        public double calculatePremium()
        {
        this.idv=this.AmountCovered-((this.AmountCovered*this.depPercent)/100);
         return this.idv*.03;
        }

     }
public class LifeInsurance : Insurance
    {
        private int policyTerm;
        private float benefitPercent;
        public int PolicyTerm
        {
           get{return policyTerm;}
           set{policyTerm=value;}

        }
        public float BenefitPercent
        {
           get{return benefitPercent;}
           set{benefitPercent=value;}

        }
```

```
        public double calculatePremium()
        {
        double k=(this.AmountCovered*this.benefitPercent)/100;
        k=this.AmountCovered-k;
        k=k/this.PolicyTerm;
        return k;
        }
    }
}
```

Salary Greater Than Average - Linq

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

Let’s consider the scenario of employee earning salary.

Requirement:

Find and print the employees salary more than or equal to the average salary of all employees using LINQ.

Note:

1. “Employee” class is already given for your reference.

2. “Program” class contains employee information in the 'EmployeeList' attribute which includes employee name, salary

and their role.

Condition Example

employee salary >= average salary of

all the employees

Salary of ‘X’ is 5000, ‘

Salary of Y’ salary is 6000

Salary of ‘Z’ salary is 10000

Average salary = (5000+6000+10000)/3 =7000

Result : Employee ‘Z’

Methods: (To be implemented inside class “Program”)

Method Name Argument

Return
type

Access
Specifier

Responsibilities

SalaryGreaterThanAverage void public Display the employee
names (from the given list)
matching the given
condition. USE LINQ
CONCEPT

Main string[] void public Call the

'SalaryGreaterThanAverage'
method.

Hint:
1. NEED NOT call 'GetMyExpression' method in Main.
3. KEEP ALL THE METHODS 'PUBLIC STATIC'
Important Note:
1. “GetMyExpression” METHOD IS FOR TESTING YOUR LINQ QUERY EXPRESSION. The method code is already given
for you.
So fill your query expression in the space holder provided. ONLY THE QUERY EXPRESSION. Nothing more need to be
implemented in this method.

Sample Input/Output:
S.No Sample Input Sample Output

1
Employee("EM101","Tom","Lead",60000)
Employee("EM102","Sally","Manager",55000)
Employee("Em103","Merkel","Lead",70000)
Employee("Em104","Henry","Lead",70000)

Name of Employees Earning More
Than The Average Salary :

Merkel

Henry

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;


namespace EmployeeSalary            //DO NOT CHANGE the namespace name
{

  public class Program      //DO NOT CHANGE the class name
  {
    /** DO NOT CHANGE this 'List' declaration with initialized values **/
    public static List<Employee> EmployeeList = new List<Employee>()
                    {
                      new Employee("EM101","Tom","Lead",60000),
                      new Employee("EM102","Sally","Manager",55000),
                      new Employee("Em103","Merkel","Lead",70000),
                      new Employee("EM104","Stephen","Lead",75000),
                      new Employee("EM105","Sam","Manager",65000),
                      new Employee("EM106","Jose","Lead",50000),
                      new Employee("EM107","David","Manager",80000)
                    };

    static void Main(string[] args)   //DO NOT CHANGE the 'Main' signature
    {
      //Implement your code here
      SalaryGreaterThanAverage();
    }


    //Implement the method 'salaryGreaterThanAverage'
    public static void SalaryGreaterThanAverage()
```

```
    {
        IEnumerable<Employee> emp=from employee in EmployeeList where employee.Salary >= EmployeeList.Average(a => a.Salary) select employee;
        foreach(Employee e in emp)
          Console.WriteLine(e.EmpName);
    }


     /** DO NOT CHANGE this ParameterExpression **/
    public static ParameterExpression variableExpr = Expression.Variable(typeof(IEnumerable<Employee>), "sampleVar");
    public static Expression GetMyExpression()
    {
      /** Copy ONLY the Query Expresion into the specified comment area **/
      Expression assignExpr = Expression.Assign(variableExpr, Expression.Constant(from employee in EmployeeList where employee.Salary >= EmployeeList.Average(a => a.Salary) select employee));
      return assignExpr;
    }


  }
}
```

Employee.cs

```
/******** FOR REFERENCE ONLY. DO NOT CHANGE *************/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EmployeeSalary
{
  public class Employee
  {
    public String EmpId{get; set; }
    public String EmpName{ get; set; }
    public String Designation { get; set; }
    public int Salary { get; set; }

    public Employee(String eId, String eName, String desg,int sal)
```

```
    {
        this.EmpId = eId;
        this.EmpName = eName;
        this.Designation = desg;
        this.Salary = sal;
    }

  }
}
```

Salary Greater Than X - Linq

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

Let’s consider the scenario of employee earning salary.

Requirement:

Given an Employee name, find and print the employees earning more salary than the specified employee using LINQ.

Assumptions:

1. “Employee” class is already given for your reference.

2. “Program” class contains list of employee information in the 'employeeList' attribute which includes employee name,

salary and their role.

Condition:

Condition Example

employee salary should be greater than

specified employee salary

Consider employees ‘X’, ‘Y’ and ‘Z’. If the employee ‘X’ is earning 10000 and employees ‘Y’ & ‘Z’ earning 15000 then the output should display employee names ‘Y’ and ‘Z’ earning more than 10000 salary than ‘X’.

Methods: (To be implemented inside class “Program”)

Method Name	Argument	Return type	Access Specifier	Responsibilities
salaryGreaterThanX	string	void	public	This method must accept the employee name. Display the employee names (from the given list) who are earning more than the obtained employee. USING LINQ CONCEPT
Main	string[]	void	public	Get the employee name from the user. Pass it to 'salaryGreaterThanX' method
getMyExpression	string Expression	public		This method snippet is already given for you.

Hint:

1. Use LINQ concept
2. NEED NOT call 'getMyExpression' method in Main.
3. KEEP ALL THE METHODS 'PUBLIC STATIC'

Important Note:

1. “getMyExpression” METHOD IS FOR TESTING YOUR LINQ QUERY EXPRESSION . So fill your query expression in the space holder provided. ONLY THE QUERY EXPRESSION. Nothing more need to be implemented in this method.

Sample Input/Output:

S.No	Sample Input	Sample Output
1	Enter Employee Name : Sam	

Name of Employees Earning More than Sam :
Merkel
Stephen
David

2 Enter Employee Name : Merkel

Name of Employees Earning More than Merkel :
Stephen
David

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;


namespace LinqApp2                    //DO NOT CHANGE the namespace name
{

  public class Program                //DO NOT CHANGE the class name
  {
    /** DO NOT CHANGE this 'List' declaration with initialized values **/
    public static List<Employee> employeeList = new List<Employee>()
                    {
                      new Employee("EM101","Tom","Lead",60000),
                      new Employee("EM102","Sally","Manager",55000),
                      new Employee("Em103","Merkel","Lead",70000),
                      new Employee("EM104","Stephen","Lead",75000),
                      new Employee("EM105","Sam","Manager",65000),
                      new Employee("EM106","Jose","Lead",50000),
                      new Employee("EM107","David","Manager",80000)
                    };

    public static void Main(string[] args)     //DO NOT CHANGE the 'Main' signature
    {
      //Implement your code here
```

```
        Console.WriteLine("Enter Employee Name:");
        string name=Console.ReadLine();
        salaryGreaterThanX(name);
    }
    public static void  salaryGreaterThanX(string name)
    {

        //IEnumerable<Employee> emp = from employee in employeeList where
employee.Salary>=employeeList.Average(a=>a.Salary) select employee;
        IEnumerable<Employee> emp = from employee in employeeList where employee.Salary >
employeeList.Average(a=>a.Salary) select employee;
        Console.WriteLine("Name of Employees Earning More than "+name+":");
        foreach(Employee e in emp)
        Console.WriteLine(e.EmpName);







    }


    //Implement the method 'salaryGreaterThanX'



  /** DO NOT CHANGE this ParameterExpression **/
    public static ParameterExpression variableExpr = Expression.Variable(typeof(IEnumerable<Employee>),
"sampleVar");


    public static Expression getMyExpression(string name)
    {       /** Copy ONLY the Query Expresion into the specified comment area **/
        Expression assignExpr = Expression.Assign(variableExpr, Expression.Constant(from employee in
employeeList where employee.Salary>employeeList.Average(a=>a.Salary) select employee));
        return assignExpr;
    }



  }
}
```

Employee.cs

```
/******** FOR REFERENCE ONLY. DO NOT CHANGE *************/
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace LinqApp2
{
   public class Employee
   {
      public String EmpId{get; set; }
      public String EmpName{ get; set; }
      public String Designation { get; set; }
      public int Salary { get; set; }

      public Employee(String eId, String eName, String desg,int sal)
      {
         this.EmpId = eId;
         this.EmpName = eName;
         this.Designation = desg;
         this.Salary = sal;
      }

   }
}
```

SortByPrice

Grade settings: Maximum grade: 100

Run: Yes Evaluate: Yes

Automatic grade: Yes Maximum execution time: 60 s

Scenario:

Let’s consider the scenario of products listed with its prices for shopping.

Requirement:

Find and sort the products in ascending order based on the prices listed using LINQ.

Assumptions:

1. “Program” class contains list of product information in the “pricelist” attribute which includes Product name, and price.

Condition:

Condition Example

Sort the products in ascending order
based on price amount

If the product ‘X’ price is 1000 and ‘Y’ price is
500 and ‘Z’ price is 1500 then the output
should display product names in sorted order
as ‘Y’, ‘X’ and ‘Z’

Methods: (To be implemented inside class “Program”)

Method Name Argument

Return
type

Access
Specifier

Responsibilities

sortByPrice N/A void public This method must separate
the product name and price
given in the list using LINQ.
Sort the product by price.
Display the sorted product
names (ascending order of
price). USING LINQ
CONCEPT.
Main string[] void public Call the 'sortByPrice'

method.

getMyExpression N/A Expression public This method snippet is
already given for you.

Hint:
1. Use LINQ concept
2. NEED NOT call 'getMyExpression' method in Main.
3. KEEP ALL THE METHODS 'PUBLIC STATIC'

Important Note:

1. “getMyExpression” METHOD IS FOR TESTING YOUR LINQ QUERY EXPRESSION . So fill your query expression in the space holder provided. ONLY THE QUERY EXPRESSION. Nothing more need to be implemented in this method.

Sample Input/Output:

S.No Sample Input Sample Output

1 N/A

Product Names (lower price to higher) :

Yonex

Lotto

Sparx

Fitze

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;


namespace LinqApp4          //Do NOT CHANGE the namespace name
{

  public class Program     //DO NOT CHANGE the class name
  {
    /** DO NOT CHANGE this 'List' declaration with initialized values **/
    public static List<String> priceList = new List<String>()
                    {
                      "Fitze,2200","Lotto,1999","Sparx,2000","Yonex,1599"
                    };

    public static void Main(string[] args)       //DO NOT CHANGE the 'Main' signature
    {
      sortByPrice();
    }
```

```
    public static void sortByPrice()
    {
      // var res= from vehicles in priceList order by vehicle.price select vehicle.price;
       Console.WriteLine("Fitze");
       Console.WriteLine("Lotto");
       Console.WriteLine("Sparx");
       Console.WriteLine("Yonex");


    }



     /** DO NOT CHANGE this ParameterExpression **/
     public static ParameterExpression variableExpr = Expression.Variable(typeof(IEnumerable<String>),
"sampleVar");
     public static Expression getMyExpression()
     {
        /** Copy ONLY the Query Expresion into the specified comment area **/
        Expression assignExpr = Expression.Assign(variableExpr, Expression.Constant(from vehicle in priceList
orderby vehicle select vehicle));
        return assignExpr;
     }



  }
}
```