Peer Evaluation for Lab 6 – Chapter 13

Your name: (Your lab is the one being evaluated)	Will Schultz
Name(s) of peer evaluator(s)	
Date:	6/1/2018

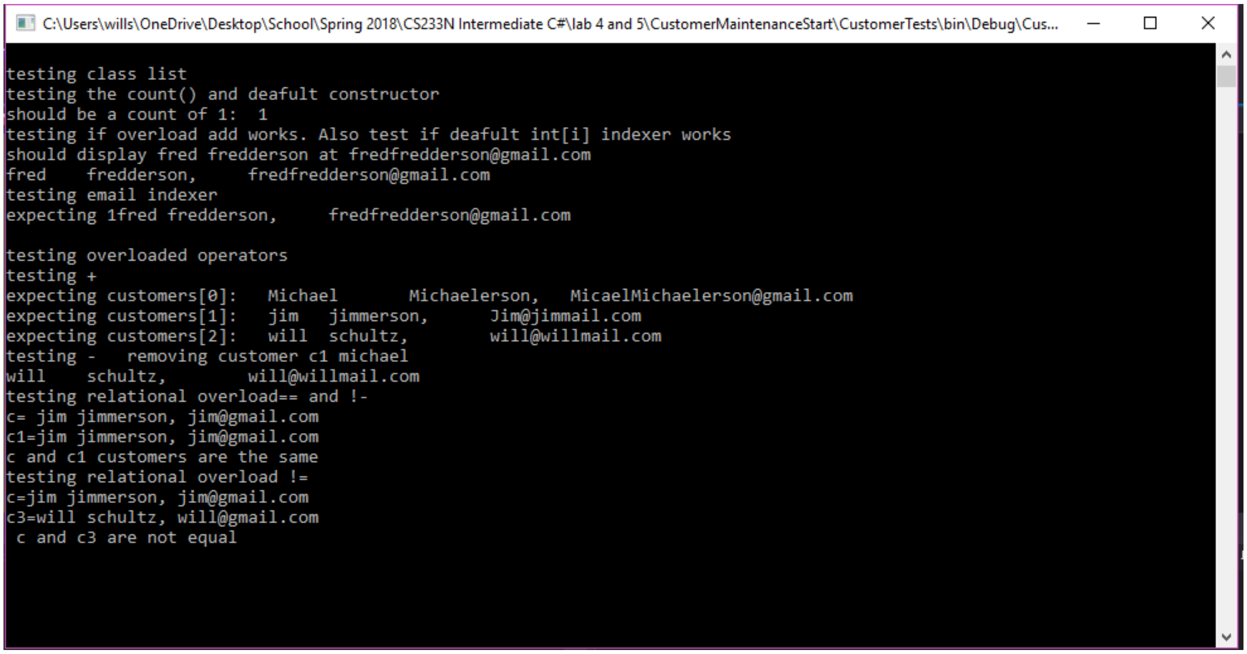
Instructions

You should have already completed Lab 6. After you and a peer have evaluated your work, you will submit this evaluation along with screen shots and source code indicated in moodle. You may make corrections to your work as a result of the evaluation.

In Class Exercises – CustomeList Class – Exercise 13.1 (NOT THE UI EXERCISES)	
Completed CustomerList Class?	
• Created a class diagram in visual studio? Screen shot included?	Yes
• Instance variables are camelCase and private?	Yes
• Property/Method names are TitleCase? Properties/Methods are public?	Yes
• Implements all properties/methods in the specification in the chart on page 419?	Yes
• Implements + and - operators?	Yes
• Implements delegate named ChangeHandler and Changed event?	No
• Implements any other methods/operators for extra credit? What are they?	No
• Completed Customer Tests? Tests all properties and methods in the class? Tests + and – operators? DOESN'T NEED TO TEST the delegate and event. Screen shot is included?	Yes
• Source code includes CustomerList class as well as test program?	Yes
One thing that you learned from writing and testing the class: I learned how to create a class list. I also learned about overloading operators, relational operators, and some list methods.	
Something that you'd like to continue working on:	
Time you spent completing and testing the class:	

Class diagram, screen shot of test running, test code, class code

CustomerList Class		
Property	Method	Indexer
Count	Add(X2)	Int[i]
	Remove	String[email]
	Save	//string[lastName]
	Fill	



C:\Users\wills\OneDrive\Desktop\School\Spring 2018\CS233N Intermediate C#\lab 4 and 5\CustomerMaintenanceStart\CustomerTests\bin\Debug\Cus...

```
testing class list
testing the count() and deafult constructor
should be a count of 1:  1
testing if overload add works. Also test if deafult int[i] indexer works
should display fred fredderson at fredfredderson@gmail.com
fred    fredderson,     fredfredderson@gmail.com
testing email indexer
expecting 1fred fredderson,     fredfredderson@gmail.com

testing overloaded operators
testing +
expecting customers[0]:   Michael       Michaelerson,   MicaelMichaelerson@gmail.com
expecting customers[1]:   jim   jimmerson,      Jim@jimmail.com
expecting customers[2]:   will  schultz,        will@willmail.com
testing -   removing customer c1 michael
will    schultz,        will@willmail.com
testing relational overload== and !-
c= jim jimmerson, jim@gmail.com
c1=jim jimmerson, jim@gmail.com
c and c1 customers are the same
testing relational overload !=
c=jim jimmerson, jim@gmail.com
c3=will schultz, will@gmail.com
 c and c3 are not equal
```

```
namespace CustomerMaintenanceClasses
{
    public class CustomerList
    {
        private List<Customer> customers;

        public CustomerList()
        {
            customers = new List<Customer>();
        }
        public int Count()
        {
            return customers.Count;
        }
        public void Add(Customer customer)
        {
            customers.Add(customer);
        }
        public void Add(string firstName, string lastName,string email)
        {
            Customer c = new Customer(firstName, lastName, email);
            customers.Add(c);
        }
        public void Remove(Customer customer)
        {
            customers.Remove(customer);
        }
        public void Fill()
        {
            customers = CustomerDB.GetCustomers();
        }
        public void Save()
        {
            CustomerDB.SaveCustomers(customers);
        }
        public Customer this[int i]
        {
            get
            {
                if(i<0||i>=customers.Count)
                    throw new ArgumentOutOfRangeException(i.ToString());
                return customers[i];
            }


        }
        //public Customer this[string lastName]
        //{
        //    get
        //    {
        //        foreach(Customer c in customers)
        //        {
        //            if (c.LastName == lastName)
        //                return c;
        //        }
        //        return null;
        //    }
        //}
        public Customer this[string email]
        {
            get
            {
                if (email.IndexOf("@") == -1 ||
                      email.IndexOf(".") == -1)
                    throw new ArgumentException("please enter a valid email");
                else
                {
                    foreach (Customer c in customers)
                    {
                        if (c.Email == email)
                            return c;
                    }
                    return null;
                }
            }

        }
        public static CustomerList operator +(CustomerList customers, Customer customer)
        {
            customers.Add(customer);
            return customers;
        }
        public static CustomerList operator -(CustomerList customers, Customer customer)
        {
            customers.Remove(customer);
                  return customers;
        }

    }
}
```

```
public static bool operator ==(Customer c1, Customer c2)
        {
            if (object.Equals(c1, null))
                if (object.Equals(c2, null))
                    return true;
                else
                    return false;
            else
                return c1.Equals(c2);
        }
        public static bool operator !=(Customer c1, Customer c2)
        {
            return !(c1 == c2);
        }
        public override bool Equals(object obj)
        {
            if (obj == null)
                return false;
            Customer c = (Customer)obj;
            if (this.LastName == c.LastName &&
                this.FirstName == c.FirstName &&
                this.Email == c.Email)
                return true;
            else
                return false;
        }
        public override int GetHashCode()
        {
            string hashString = this.FirstName + this.LastName + this.Email;
            return hashString.GetHashCode();
        }
```

In Class Exercises – Deck class	
Completed Deck Class? • Created a class diagram in visual studio? Screen shot included? • Instance variables are camelCase and private? • Property/Method names are TitleCase? Properties/Methods are public? • Implements all properties/methods in the specification on moodle? • Implements any other methods/operators for extra credit? What are they? • Completed Deck Tests? Tests all properties and methods in the class? Screen shot is included? • Source code includes Deck class as well as test program?	
One thing that you learned from writing and testing the class:	
Something that you'd like to continue working on:	
Time you spent completing and testing the class:	

Class diagram, screen shot of tests running, test code, class code

Programming style for all programs	
Is proper indentation used? Is each property/method indented properly? Is each control structure indented properly?	Yes
Are comments used appropriately?	No
Do variable names use camel case? (camelCase for example)	Yes
Do property/method names use Title Case (or Pascal Case?)	yes

General comments and notes from the evaluator:

One thing that you learned from completing the evaluation:

Screen Shots and Source Code