Trent University
COIS1020H
Lab 5 (Windows Version)

1) Introduction to Objects (with Constructors and Properties)

a) Enter the following program. Notice that it consists of two classes that will go into the same project. Please put each class into its own code file (use Lab5_1 for the Project name and Lab5_1_1 / Lab5_1_2 for the Code File names). This program focuses on classes, objects and properties. Class *EmployeeDemo* is a static class that contains *Main()*. It is the driver class for the program and it creates dynamic objects from the non-static class *Employee* using the three different constructors. Enter the program into Microsoft Visual C# (ignore the line numbers).

Please note that there are text versions of these classes named Lab5_1_1.txt and Lab5_1_2.txt (without the line numbers) available in the Labs folder on BlackBoard for copying and pasting. Also, please be aware that the line numbers in the lab exercise may not be the same as the line numbers you see in Visual Studio.

```
1. using System;            // Code File Lab5_1_1.cs
2. public static class EmployeeDemo
3. {
4.    public static void Main()
5.    {
6.       // Create Employee objects utilizing different constructors
7.       Employee emp1 = new Employee();
8.       // Employee emp2 = new Employee("Joy Rogers", 10127, "HR", "Manager");
9.       // Employee emp3 = new Employee("Mark Jones", 10065);
10.
11.      // Use Properties to assign values to data fields
12.      // emp1.Name = "Susan Meyers";
13.      // emp1.IdNumber = 10012;
14.      // emp1.Department = "Accounting";
15.      // emp1.Position = "Vice President";
16.
17.      // emp3.Department = "IT";
18.      // emp3.Position = "Support";
19.
20.      // Use Properties to output data from Employee objects
21.      Console.WriteLine("Employee 1: {0}, {1}, {2}, {3}",
22.         emp1.Name, emp1.IdNumber, emp1.Department, emp1.Position);
23.      // Console.WriteLine("Employee 2: {0}, {1}, {2}, {3}",
24.      //    emp2.Name, emp2.IdNumber, emp2.Department, emp2.Position);
25.      // Console.WriteLine("Employee 3: {0}, {1}, {2}, {3}",
26.      //    emp3.Name, emp3.IdNumber, emp3.Department, emp3.Position);
27.      Console.ReadLine();
28.   }
29. }
```

```
1. using System;      // Code File Lab5_1_2.cs
2. public class Employee    // this is a dynamic class (no "static")
3. {
4.    private int idNumber;
5.    private string name, department, position;
6.
7.    public Employee() // Constructor 1: No Arg
8.    {
9.       Name = "";
10.      IdNumber = 0;
11.      Department = "";
12.      Position = "";
13.   }
14.
15.   public Employee(string eName, int eIdNum) // Constructor 2: Two Arg
16.   {
17.      Name = eName;
18.      IdNumber = eIdNum;
19.      Department = "";
20.      Position = "";
21.   }
22.
23.   // Constructor 3: Four Arg
24.   public Employee(string eName, int eIdNum, string eDept, string ePos)
25.   {
26.      Name = eName;
27.      IdNumber = eIdNum;
28.      Department = eDept;
29.      Position = ePos;
30.   }
31.
32.   public string Name    // Property for priviate data - name
33.   {
34.      get { return name; }
35.      set { name = value; }
36.   }
37.
38.   public int IdNumber   // Property for priviate data - idNumber
39.   {
40.      get { return idNumber; }
41.      set { idNumber = value; }
42.   }
43.
44.   public string Department    // Property for priviate data – department
45.   {
46.      get { return department; }
47.      set { department = value; }
48.    }
49.
50.   public string Position      // Property for priviate data - position
51.
52.   {
53.      get { return position; }
54.      set { position = value; }
55.   }
56. }
```

b) Build and execute the program (F5) … resolve any syntax mistakes (if you make any) until the program compiles.

What is the output?

Which constructor from the Employee class was invoked to instantiate `emp1` object?

Why are the contents of the `emp1` object appear to be empty when printed?

c) Uncomment Lines 12-15 in the *EmployeeDemo* class to render them active (i.e, remove the // from in front of the C# commands). Compile and execute the program (F5).

What is the output?

Why is the output for Employee 1 different that in Part (b)?

d) Uncomment Lines 8, 23, 24 in the *EmployeeDemo* class to render them active, execute the program (F5).

What is the output?

Which constructor from the Employee class was invoked to instantiate `emp2` object?

e) Uncomment Lines 9, 25 and 26 in the *EmployeeDemo* class to render them active, execute the program (F5).

What is the output?

Which constructor from the Employee class was invoked to instantiate `emp3` object?

f) Uncomment Lines 17 and 18 in the *EmployeeDemo* class to render them active, execute the program (F5).

What is the output?

Why is the output for Employee 3 different that in Part (e)?

g) Put a Breakpoint (F9) on Line 7, 8, and 9 in the *EmployeeDemo* class and execute the program in the debug mode (F5). When the program stops on Line 7, examine the contents of the *Locals* window.

What do you see in the *Locals* window?

h) Use (F11) to make one step.

Where did you step to (be specific)?

i) Use (F5) move to the next Breakpoint (should be on Line 8). Re-examine the Locals window (be sure to expand the view of the objects in the *Locals* window by clicking on the "+" sign; otherwise, you will not be able to take notice of any changes).

How has it changed from Part (g)?

j) Use (F11) to make one step.

Where did you step to (be specific)?

k) Use (F5) move to the next Breakpoint (should be on Line 9). Re-examine the Locals window (be sure to expand the view of the objects in the *Locals* window by clicking on the "+" sign; otherwise, you will not be able to take notice of any changes).

How has it changed from Part (i)?

l) Continue stepping through the program (F11) examining the contents of the *Locals* window and observing how the program transfers control between *Main* to *Employee*.

Comment on your experience.

m) Remove all thee breakpoints (F9). In Line 13, change *IdNumber* to *idNumber* in the *EmployeeDemo* class (basically you are substituting a public property for a private name). Run the program (F5).

What happens and why?

n) In Line 4 of the *Employee* class (note the different file here), change *private* to *public*. Run the program (F5).

What happens and why?

Is it a good idea to solve the problem this way? Why?

Answer all the highlighted questions in a file and then submit a PDF of this file (called it Lab5_1.pdf) to the Lab 5 dropbox. When asked "What is the output", simply type in what is seen in the output window.

2) Introduction to Objects (with Constructors and Properties)

a) Enter the following program that consists of two classes in the same project. Please put each class into its own code file (use Lab5_2 for the Project name and Lab5_2_1 / Lab5_2_2 for the Code File names). Class *CircleDemo* is a static class that contains *Main()*. It is the driver class for the program and it creates dynamic objects the *Circle* class. Enter the program into Microsoft Visual C# (ignore the line numbers). There is a syntax error in one of these classes.

Please note that there are text versions of these classes named Lab5_2_1.txt and Lab5_2_2.txt (without the line numbers) available in the Labs folder on BlackBoard for copying and pasting.

Also, please be aware that the line numbers in the lab exercise may not be the same as the line numbers you see in Visual Studio.

```
1. using System;     // Code File Lab5_2_1.cs
2. public static class CircleDemo
3. {
4.    public static void Main()
5.    {
6.       double inpRadius;
7.       // create a circle object with no arg constructor
8.       Circle cir1 = new Circle();
9.       // create a circle object with one arg constructor
10.      Circle cir2 = new Circle(10);
11.      // Circle cir3;
12.
13.      // print out the area of the circles
14.      Console.WriteLine("Circle 1 has radius {0} and area {1:F2}", cir1.Radius,
15.                cir1.GetArea);
16.      Console.WriteLine("Circle 2 has radius {0} and area {1:F2}", cir2.Radius,
17.                cir2.GetArea());
18.      //prompt the user for a radius
19.      do
20.      {
21.         Console.Write("Enter a positive radius => ");
22.         inpRadius = Convert.ToDouble(Console.ReadLine());
23.      } while (inpRadius < 0);
24.      cir1.Radius = inpRadius;
25.
26.      // print out the area of the circles
27.      Console.WriteLine("Circle 1 has radius {0} and area {1:F2}", cir1.Radius,
28.                cir1.GetArea());
29.      Console.WriteLine("Circle 2 has radius {0} and area {1:F2}", cir2.Radius,
30.                cir2.GetArea());
31.
32.      Console.ReadLine();
33.   }
34. }
```

```
1. public class Circle     // Code File Lab5_2_2.cs
2. {
3.    private double radius;
4.    private const double PI = 3.14159;
5.
6.    // no argument Constructor
7.    public Circle()
8.    {
9.       radius = 0;
10.   }
11.   // one argument Constructor
12.   public Circle(double rad)
13.   {
14.      // make sure the parameter is not negative
15.      if (rad < 0)
16.         radius = 0;
17.      else
18.         radius = rad;
19.   }
20.   // Radius Property
21.   public double Radius
22.   {
23.      // return the private data value on
```

```
24.       get
25.       {
26.          return radius;
27.       }
28.       // make sure the implicit parameter is not negative on set
29.       set
30.       {
31.       if (value < 0)
32.          radius = 0;
33.       else
34.          radius = value;
35.       }
36.    }
37.    // GetArea Method
38.    public double GetArea()
39.    {
40.      double area;
41.      // compute the area = radius^2 * PI
42.      area = radius * radius * PI;
43.      return area;
44.    }
45. }
```

b) Run the program using input -2, 5

What is the syntax error and how did you fix it?

What is the output?

c) Add the following lines of code after Line 30 in the *CircleDemo* class (Lab5_2_1.cs)

```
cir1.Radius = -8;
// print out the area of the circles
Console.WriteLine("Circle 1 has radius {0} and area {1:F2}", cir1.Radius,
          cir1.GetArea());
Console.WriteLine("Circle 2 has radius {0} and area {1:F2}", cir2.Radius,
          cir2.GetArea());
```

What is the output (assume an input of 5)?

How did the program arrive at this result for Circle 1?

d) Add a new instance method to the *Circle* class in Lab5_2_2.cs to compute the circumference of a circle (recall that it is 2 * PI * radius). The new method should have the following header

```
public double GetCircumference()
```

Now in the *Main()* method in Lab5_2_1.cs, modify the `Console.WriteLine` statements in Lines 14/15, 16/17, 27/28, and 29/30 to also print out the values of circumference.

Show the details of the *GetCircumference()* method?

Show the details of ONE of the modified *Console.WriteLine* statements?

What is the output when run with an input of 5?

e) Finally, let's look at including an overloaded + operator to our *Circle* class. Not that this is based in any Math textbook, but let us define that when two circles are added, the radius of the new circle will be the square root of the sum of the areas divided by PI. Add the following overloaded operator method to the Circle class:

```
public static Circle operator+(Circle circle1, Circle circle2)
{
      Circle circle3 = new Circle();
      Double sumArea, newRadius;
      // compute the sum of the areas of the two circles
      sumArea = circle1.GetArea() + circle2.GetArea();
      // compute the radius of the new circle
      newRadius = Math.Sqrt(sumArea) / PI;
      // assign the radius to the new circle
      circle3.Radius = newRadius;
      // return the new circle
      return circle3;
}
```

Now uncomment Line 11 in the *CircleDemo* class (Lab5_2_1.cs) and then after Line 30, add lines of code to add `cir1` to `cir2`, store the result in `cir3`, and print out the radius, area and circumference of `cir3`.

Show the details of the lines of code added to the *CircleDemo* class?

What is the output when run with an input of 5?

Answer all the highlighted questions in a file and then submit a PDF of this file (called it Lab5_2.pdf) to the Lab 5 dropbox. When asked "What is the output", simply type in what is seen in the output window.

3) Putting it All Together

Below you will find two classes that will allow you to implement and test a program to work with simple bank accounts. This program consists of two classes: a dynamic BankAccount and a static BankAccountDemo (which contains Main()). The sole purpose of Main() is to test the various properties and methods of BankAccount objects. Enter the following program into C# (ignore the line numbers) and replace the lines marked with // ******* with the appropriate C# code (may require more than one line of code to complete the missing parts). Please put each class into its own code file (use Lab5_3 for the Project name and Lab5_3_1 / Lab5_3_2 for the Code File names).

Please note that there are text versions of these classes named Lab5_3_1.txt and Lab5_3_2.txt (without the line numbers) available in the Labs folder on BlackBoard for copying and pasting. Also, please be aware that the line numbers in the lab exercise may not be the same as the line numbers you see in Visual Studio.

```
1. // Name: xxxxxxxxxxxxx                    // Code File Lab5_3_1.cs
2. // Student Number: xxxxxxx
3. // Lab 5, Part 3
4. // Program Description: This program consists of two classes: a dynamic BankAccount
5. //    and a static BankAccountDemo (which contains Main()).  The sole purpose of
6. //    Main() is to test the various properties and methods of BankAccount objects.
```

```
7.
8. using System;
9. public static class BankAccountDemo
10. {
11. public static void Main()
12. {
13.    int acctNumber;
14.    double amount;
15.    BankAccount savings = new BankAccount();
16.    BankAccount chequing = new BankAccount(12345, 350.45);
17.    BankAccount newAcct;
18.
19.    // input a 5 digit account number and balance for savings
20.    do {
21.       Console.Write("Enter a 5-digit account number => ");
22.       acctNumber = Convert.ToInt32(Console.ReadLine());
23.    } while ((acctNumber < 10000) || (acctNumber > 99999));
24.    savings.AcctNum = acctNumber;
25.
26.    // print out the account information
27.    Console.WriteLine("Account {0} contains {1:C2}", savings.AcctNum,
28.            savings.Balance);
29.    Console.WriteLine("Account {0} contains {1:C2}", chequing.AcctNum,
30.            chequing.Balance);
31.
32.    // prompt the user to enter an amount to deposit to savings
33.    // *** Insert code
34.
35.    // perform the deposit to savings
36.    // *** Insert code
37.
38.    // print out the savings account information
39.    // *** Insert code
40.
41.    // prompt the user to enter an amount to withdraw from chequing
42.    // *** Insert code
43.
44.    // perform the withdrawal from chequing
45.    // *** Insert code
46.
47.    // print out the chequing account information
48.    // *** Insert code
49.
50.    // apply the interest to savings
51.    // *** Insert code
52.
53.    // print out the savings account information
54.    // *** Insert code
55.
56.    // combine chequing and savings into newAcct using overloaded operator
57.    // *** Insert code
58.
59.    // print out the newAcct account information
60.    Console.WriteLine("Account {0} contains {1:C2}", newAcct.AcctNum,
61.           newAcct.Balance);
62.    Console.ReadLine();
63.    }
64. }
```

```
1. // BankAccount Class
2. // Class Description: Objects of this class will possess the data and operations
3. //    consistent with bank accounts.  There are two private data members to
```

```
4. //    store the account number and balance.  The account number has a read/write
5. //    property while the property for balance is read-only. The balance is updated
6. //    using one of the methods: Deposit, Withdrawal, or Interest. Finally, there
7. //    is an overloaded operator that permits two bank account objects to be
8. //    combined to create a new bank account.
9.
10. using System;
11. public class BankAccount
12. {
13.    private int acctNum;
14.    private double balance;
15.    public const double SERVICE_CHARGE = 1.00;  // for Withdrawals only
16.    public const double INTEREST_RATE = 0.015;  // fixed interest rate
17.
18.    // no arg constructor
19.    public BankAccount()
20.    {
21.       acctNum = 0;
22.       balance = 0;
23.    }
24.
25.    // two arg constructor
26.    public BankAccount(int aNumber, double bal)
27.    {
28.       acctNum = aNumber;
29.       // ensuring the balance is not negative
30.       if(bal < 0)
31.          balance = 0;
32.       else
33.          balance = bal;
34.    }
35.
36.    // AcctNum Property
37.    public int AcctNum
38.    {
39.       set
40.       { acctNum = value; }
41.       get
42.       { return acctNum; }
43.    }
44.
45.    // Balance Property (read-only)
46.    public double Balance
47.    {
48.    get
49.    { return balance; }
50.    }
51.
52.    // Deposit Method
53.    public void Deposit(double amt)
54.    {
55.    // check to see that the deposit amount is positive
56.    if (amt > 0)
57.       balance += amt;
58.    }
59.
60.    // Withdrawal Method (a Service Charge)
61.    public void Withdrawal(double amt)
62.    {
63.       // *** Insert code
64.    }
65.
```

```
66.    // instance method to add interest onto the balance
67.    public void Interest()
68.    {
69.       // *** Insert code
70.    }
71.
72.    // overloaded operator+ to combine the contents of two accounts
73.    //        Assume new account number will be the average of the
74.    //        two account numbers and the balance will be the sum
75.    // Parameters: the two accounts to be combined
76.    // Returns: the new account with the
77.    public static BankAccount operator+(BankAccount acc1, BankAccount acc2)
78.    {
79.       // *** Insert code
80.    }
81. }
```

Once you are comfortable that the program works correctly, demonstrate it to the lab personnel, and then submit your Lab5_3_1.cs and Lab5_3_2.cs files to the Lab 5 dropbox.