MCSE 541: Web Computing and Mining

C# Class, Object & Method
C# Operator, Indexer & Properties

Prof. Dr. Shamim Akhter

@Text Book: Chapter 9,10

Classes and Objects

- A class is a template that define the forms of an object

Blueprint/Logical State of a Bike

Construction

Physical State of Bikes

Field Members

- Instance Variable
- Static/Class Variable

Function Members

- Constructor, Destructor
- Indexers, Events, Operators and Properties

A Simple Class

- Class is created by use of keyword class.

```
modifier class Classname {

   modifier data-type field1;
   modifier data-type field2;
   ...
   modifier data-type fieldN;

   modifier Return-Type methodName1(parameters) {
      //statements
   }

   ...

   modifier Return-Type methodName2(parameters) {
      //statements
   }
}
```

Instance Variable

Handling Static and Non-static variable

1) Non-static variable cannot be referenced without creating class instance, WHY?
2) Static variable is referenced without class instance but error when referred with instance. WHY?

1)

Program.cs*

Winter_Prog1 | Winter_Prog1.Program | Main(string[] args)

```
1   using System;
2
3   namespace Winter_Prog1
4   {
        2 references
5       class Program
6       {
7           int myVariable;
8           static int data = 30;
9
            0 references
10          static void Main(string[] args)
11          {
12              Program p = new Program();
13
14              Console.WriteLine(myVariable);
15              Console.WriteLine(p.data
16          }
17      }
```

(field) int Program.myVariable

CS0120: An object reference is required for the non-static field, method, or property 'Program.myVariable'

2)

```
1   using System;
2
3   namespace Winter_Prog1
4   {
        2 references
5       class Program
6       {
7           int myVariable;
8           static int data = 30;
9
            0 references
10          static void Main(string[] args)
11          {
12              Program p = new Program();
13
14              Console.WriteLine(p.myVariable);
15              Console.WriteLine(p.data);
16          }
17      }
```

(field) static int Program.data

CS0176: Member 'Program.data' cannot be accessed with an instance reference; qualify it with a type name instead

Show potential fixes (Alt+Enter or Ctrl+.)

Output

Show output from: Package Manager

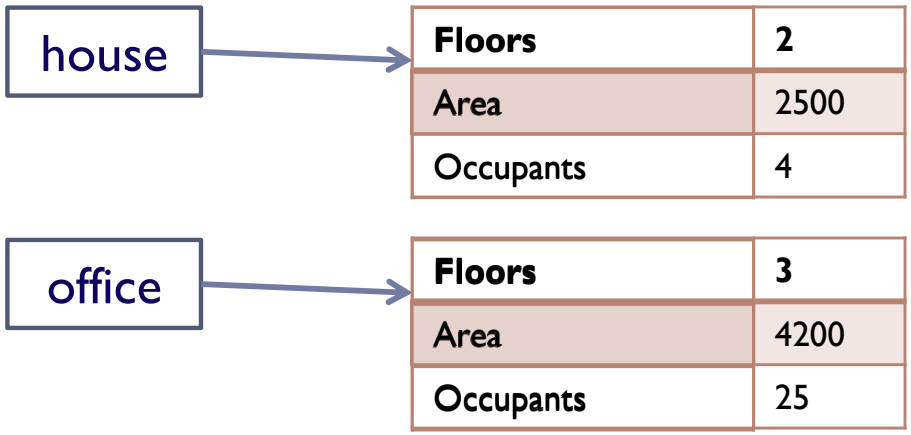
@Text Book: Chapter 6

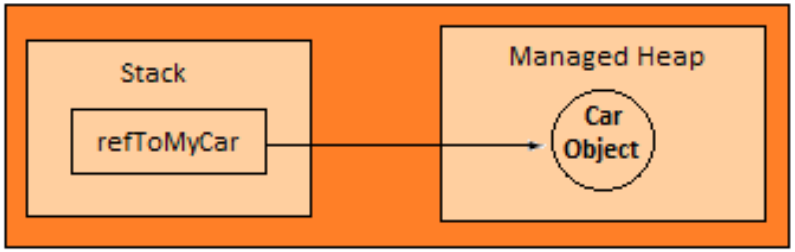
Reference Variable and Assignment

```
using System;

class Building{
        public int Floors;
        public int Area;
        public int Occupants;

        public void areaPerPerson(){
                Console.WriteLine(" "+ Area/Occupants + " area per person");
        }
}
public class Demo{
    static void Main(){
        Building house  = new Building();
        Building office = new Building();
        house.Occupants=4;   house.Area=2500;  house.Floors=2;
        office.Occupants=25;  office.Area=4200;   office.Floors=3;
        house.areaPerPerson();
        office.areaPerPerson();
    }
}
```





Building house1= new Building();
Building house2= house1

Constructor

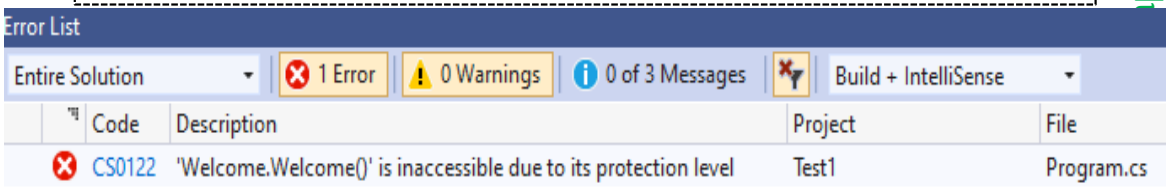
- Constructors
  - are special methods called when a class is instantiated.
  - will not return anything.
  - name is same as class name.
  - By default C# will create default constructor internally.
  - with no arguments and no body is called default constructor.
  - with arguments is called parameterized constructor.
  - by default public.
  - We can create private constructors.
  - Constructor allocates memory for all instance variables of its class.

Private Constructor

```
using System;
namespace ConstructorSample
{
    public class Welcome
    {
        private Welcome()   // // Default private constructor
        {
          Console.WriteLine("Default Private Constructor...");
        }

        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
}
```

```
public class demo {
        static void Main(string[] args)
        {
            Welcome obj = new Welcome();
            Console.Read();
        }
    }
```



Parameterized Constructor

```
using System;
 public class MyClass
 {
     public int x;

     public MyClass (int i)
     {
         x=i;
     }
 }
  public class Demo{
     static void Main( )
     {
        MyClass t1=new MyClass (10);
        MyClass t2=new MyClass (88);
        Console.WriteLine(t1.x + " " + t2.x);
     }
 }
```

Garbage Collection and Destructors

Memory allocation/deallocation rule:

IF YOU ALLOCATE IT (use a new),
YOU MUST DEALLOCATE IT (delete it)!

- Recovery of free memory from unused object
  - C++ delete operator is used to free allocated memory
  - C# and Java: Garbage Collection- automatically
    - Can't know or make assumptions about the timing of garbage collection
    - Non-deterministic
- Garbage Collector cleans the memory by three ways,
  1. Destructor
  2. Dispose()
  3. Finalize

- Destructor
  - is called/invoked automatically by .NET Garbage Collector (GC). We can not manually invoke or control it.
  - It ensures that a system resource owned by an objet is released.
  - Is declared like constructor except proceed with ~(tilde) -tilda
  - No return type and takes no arguments.

Destructor Example

```
using System;

  public class Destructor
  {
     public int x;

     public Destructor (int i)
     {
        x=i;
     }
     ~Destructor(){
        Console.WriteLine("Destructing " + x);
     }

     public void Generator(int i)
     {
        Destructor o = new Destructor(i);
     }
  }
```

```
public class Demo{
        static void Main( )
        {
           int count;
           Destructor ob = new Destructor(0);
           for(count=1; count<1000; count++)
                ob.Generator(count);

            Console.WriteLine("Done");
        }
    }
```

```
Destructing 755
Destructing 940
Destructing 14
Destructing 199
Destructing 384
Destructing 569
Destructing 754
Destructing 939
Destructing 13
Destructing 198
Destructing 383
Destructing 568
Destructing 753
Destructing 938
Destructing 12
Destructing 197
Destructing 382
Destructing 567
Destructing 752
Destructing 937
Destructing 11
Destructing 196
Destructing 381
Destructing 566
Destructing 751
Destructing 936
Destructing 10
Destructing 195
Destructing 380
Destructing 565
Destructing 750
Destructing 935
```

- No serialization
- Non deterministic

this Keyword

```
using System;
class Rect{
    public int Width;
    public int Height;
   //public Rect(){ this(3, 2); }
   public Rect(int Width, int Height){
            this.Width=Width;
            this.Height=Height;
    }
    public int Area(){
            return this.Width * this.Height;
    }
}
            class UseRect{
                static void Main(){
                    Rect r1= new Rect(4,5);
                    Rect r2= new Rect(7, 9);
                    Console.WriteLine("Area of r1:" +r1.Area());
                    Console.WriteLine("Area of r1:" +r2.Area());
                }
            }
```

Methods-get( ) and set()

```
class Rectangle
  {
     private int height;
     private int width;

     public Rectangle( ) { this.height = 0; this.width = 0; }
     ~Rectangle( ) { }

     public void set(int h, int w) {
        this.height= h; this.width = w;
     }
```

Methods-get( ) and set()

```
public int getHeight( ) {
        return this.height;
    }
    public int getWidth() {
        return this.width;
    }

    public Rectangle get() {
        Rectangle rec = new Rectangle();
        rec.height = this.height;
        rec.width = this.width;
        return rec;
    }
}
```

Methods-get( ) and set()

```
class Demo {
        static void Main(string[] args)
        {
            Rectangle R1 = new Rectangle();
            R1.set(10, 20);
            Console.WriteLine("Area of R1="+ R1.getHeight()*R1.getWidth());

            Rectangle R2 = R1.get();
            Console.WriteLine("Area of R2=" + R2.getHeight() * R2.getWidth());
        }
    }
```

Properties

- Suppose, you want to accomplish limit the range of values that can be assigned to a field.
  - But the filed has private access. Now?
  - We need to use get() / set() method to access the field.
  - Lets make it a better organize and more user comfortable.

- Property, a class member
  - Combines a specific field with the method that access it.

```
element-type {
    get{     //get access code
    }
    set{    //set access code
    }
}
```

A property applies specific rule to a field's value.

```
class Program
 {
        private int prop;
        public Program() { prop = 0; }

        public int prop_Property {
            get { return prop; }
            set { if (value >= 0) prop = value; }
        }
 }
```

```
class Demo {
        static void Main(string[ ] args)
        {
            Program p = new Program( );
            Console.WriteLine(p.prop_Property);
            p.prop_Property = 100;
            Console.WriteLine(p.prop_Property);
            p.prop_Property = -10;
            Console.WriteLine(p.prop_Property);
        }
    }
```

Operator Method

- Defines the action of the operator relative to its class.
  - this process is called operator overloading and closely related to method overloading.
- The method should be a public and static method.
- The function is marked by keyword **operator** followed by the operator symbol which we are overloading.
- The return type of an operator function represents the result of an expression.

Two forms of operator methods

General form for overloading a **Unary operator**

```
public static ret-type operator op(param-type operand)
{
        //operations
}
```

General form for overloading a **Binary operator**

```
public static ret-type operator op(param-type1 operand1, param-type2 operand2, )
{
        //operations
}
```

```
public class Rectangle
 {
        private int length;
        private int breadth;
        public Rectangle ( ){length=0; breadth=0;}
        public Rectangle(int length, int breadth)
        {
            this.length = length;
            this.breadth = breadth;
        }
        public int Area()
        {
            return length * breadth;
        }

        public void DisplayArea()
        {
            Console.WriteLine(this.Area());
        }
 }
```

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
    }
}
```

Binary

Solution:

Operator Overloading

Binary Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
    }
}
```

```
public static Rectangle operator +(Rectangle rect, Rectangle rect1)
{
        Rectangle result= new Rectangle( );
        result.length    = rect.length + rect1.length;
        result.breadth  = rect.breadth + rect1.breadth;

        return result;
}
```

- N.B: The above operator method will be put after the constructor of Rectangle Class.

Unary Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect3 = -rect1;
        rect3.DisplayArea();

    }
}
```

```
int a = 10; int b = -a; // b = -10
```

If p is true, then !p will be false.
If p is false, then !p will be true.

```
public static Rectangle operator -(Rectangle rect)
{
        Rectangle result= new Rectangle( );
        result.length    = - rect.length ;
        result.breadth  = - rect.breadth ;

        return result;
}
```

- N.B: The above operator method will be put after the constructor of Rectangle Class

++ / -- Postfix and Prefix form

Basically, you've misunderstood how this line works:

```
Test obj2 = ++obj;
```

If you think of using your operator as a method, that's like saying:

```
obj = Test.operator++(obj);
obj2 = obj;
```

So yes, you end up with `obj` and `obj2` being the same reference. The result of `++obj` is the value of `obj` *after* applying the `++` operator, but that `++` operator affects the value of `obj` too.

If you use

```
Test obj2 = obj++;
```

then that's equivalent to:

```
Test tmp = obj;
obj = Test.operator++(obj);
obj2 = tmp;
```

At that point, the value of `obj2` will refer to the original object, and the vlaue of `obj` will refer to the newly-created object with a higher `x` value.

++ / -- Postfix and Prefix form

```
class Program
{
    public  int x;
     public Program() {
        this.x = 0;
     }

     public Program(int x){
        this.x = x;
     }
     public static Program operator ++(Program P) {
        Program result = new Program();
        result.x = P.x + 1;
        return result;
     }
}
```

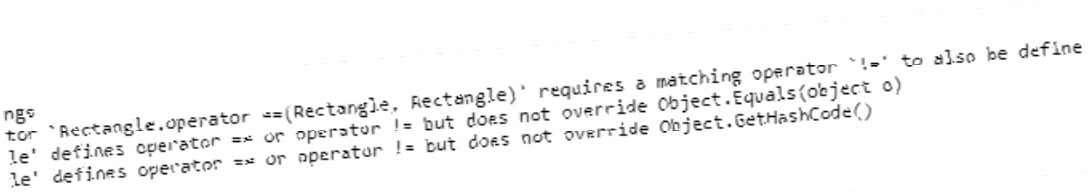
```
class Demo
{
    static void Main(string[] args)
    {
     Program P1 = new Program(2);
     Program P2 = new Program();
     Program P3 = new Program(2);
     P2 = P1++;
     Console.WriteLine("P2.x="+ P2.x+", P1.x=" + P1.x);
     P2 = ++P3;
     Console.WriteLine("P2.x=" + P2.x+", P3.x=" + P3.x);


     Console.WriteLine("Hello World!");
    }
}
```

Conditional Operator Overloading

```
public class Demo{
    public static void Main(){
        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        if (rect1 == rect2)
            Console.WriteLine("Both Rectangle have same dimensions");
    }
}
```

Both(==) and (!=) needs to define together.
Only one will keep compile error.




```
public static bool operator ==(Rectangle rect1, Rectangle rect2)
{
    return (rect1.Area() == rect2.Area()) ? true : false;
}

public static bool operator !=(Rectangle rect1, Rectangle rect2)
{
    return (rect1.Area() != rect2.Area()) ? true : false;
}
```

N.B: The above operator method will be put after the constructor of Rectangle Class

Overloading the operator methods

```
public class Demo{
    public static void Main(){

        Rectangle rect1 = new Rectangle(2, 2);
        Rectangle rect2 = new Rectangle(2, 2);
        Rectangle rect3 = rect1 + rect2;
        rect3.DisplayArea();
        int area= rec3+ 2000;

    }
}
```

```
        public static Rectangle operator +(Rectangle rect, Rectangle rect1)
        {
            return new Rectangle(rect.length + rect1.length, rect.breadth + rect1.breadth);
        }

        /// The function add the area of two rectangles provided.

        public static int operator +(Rectangle rect, int area2)
        {
            return rect.Area() + area2;
        }
```

Indexer

- Allows an object to be indexed like an array.
- [ ] operator may define to classes without operator overloading.

General form of a Indexer with get and set accessor

```
element-type this[int index] {
    get{
        //return the value specified by index
    }
    set{
        // set the value specified by index
    }
}
```

An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).

A simple Indexer

```
using System;
namespace IndexerApplication {
   class IndexedNames {
      private string[ ] namelist = new string[size];
      static public int size = 10;
      public IndexedNames()
      { for (int i = 0; i < size; i++) namelist[i] = "N.A."; }
       public string this[int index] {
            get {
                  string tmp;
                  if( index >= 0 && index <= size-1 )
                  { tmp = namelist[index]; }
                  else { tmp = ""; }
                  return ( tmp );
            }
            set {
                  if( index >= 0 && index <= size-1 )
                        { namelist[index] = value; }
            }
      }
```

```
    static void Main(string[] args)
    {
            IndexedNames names = new IndexedNames();
            names[0] = "Zara"; names[1] = "Riz";
            names[2] = "Nuha"; names[3] = "Asif";
            names[4] = "Davinder"; names[5] = "Sunil";
            names[6] = "Rubic";

            for ( int i = 0; i < IndexedNames.size; i++ ) {
                    Console.WriteLine(names[i]);
            }
            Console.ReadKey();
    }
  }
}
```

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
```

Indexers May Not Require an Underline Array

```
class Program
{
      public int this[int index] {
          get {
              if (index >= 0 && index < 16) return pwr(index);
              else return -1;
          }
          set { //no need set just read array//}
      } // index
     public int pwr(int N) {
          int result = 1;
          for (int i = 0; i < N; i++)
              result *= 2;
          return result;
      }
}
```

```
public class demo {
      static void Main(string[] args)
      {
          Program p = new Program();
          for(int i=0; i<18; i++)
              Console.WriteLine(p[i] + " ");
      }
  }
```

```
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
-1
-1
```

- Run the following link programs:
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/debugging-with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/publishing-with-visual-studio?pivots=dotnet-6-0
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/library-with-visual-studio?pivots=dotnet-6-0