```
using System; //system.console is the "system class"
//*****************************************************************
// Day 1: Using objects/Intro to Objects/Object Oriented Programming
//*****************************************************************

In this example we use encapsulation
```

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare fields/variables as `private`
- provide `public` `get` and `set` methods, through **properties**, to access and update the value of a `private` field

Why Encapsulation?

- Better control of class members (reduce the possibility of yourself (or others) to mess up the code)
- Fields can be made **read-only** (if you only use the `get` method), or **write-only** (if you only use the `set` method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

```
namespace ClassObjectDemo

//class "dog" ... object "Rexx"
{
        public class Dog
        {
            // Field name
            private string name;
            // Field color
             //because this is private we can't access it outside of this class

            private string color;
            public string Name
            {
                // Getter of the property "Name"
                get
                {
                    return this.name;
                }
                // Setter of the property "Name"
                set
                {
                    this.name = value;
                }
                //get and set in the definition of the property,
                //which perform respectively extraction of the value of the property
                //and assignment of a new value.
            }
            public string Color
            {
                // Getter of the property "Color"
                get
```

```
            {
                return this.color;
            }
            // Setter of the property "Color"
            set
            {
                this.color = value;
            }
        }
        // Default constructor
        public Dog()
        {
            this.name = "Unnamed";
            this.color = "gray";
        }
        // Constructor with parameters
        public Dog(string name, string color)
        {
            this.name = name;
            this.color = color;
        }
        //Constructer: a special method of the class,
        //which is called automatically when creating an object of this class

        // Method SayWoof
        public void SayWoof()
        {
            Console.WriteLine("Doggy {0} said: Wooof!", name);
        }

        static void Main(string[] args)
        {
            Dog firstDog = new Dog();
            //When creating an object with the operator new, two things happen:
            //memory is set aside for this object
            //its data members are initialized.
            firstDog.Name = "Gary";
            firstDog.SayWoof();

            Dog secondDog = new Dog("Lassie", "red");
                //Well built constructors save time... less code than above
            secondDog.SayWoof();
            Console.WriteLine("Dog {0} is {1}.",secondDog.Name,
secondDog.Color);
        }

    }
}

//*****************************************************************
// Day 2: Using objects/Intro to Objects/Object Oriented Programming
//*****************************************************************
//Multiple classes

//The example, which we are going to give, solves the following simple problem:
//we need a method that every time returns a value greater with one than the
//value returned at the previous call of the method. We choose the first returned
//value to be 0. Obviously this method generates the sequence of natural number.
```

```
namespace ClassObjectDemo
{
    public class Sequence
    {
        // Static field, holding the current sequence value
        private static int currentValue = 0;
        // Intentionally deny instantiation of this class
/*        private Sequence()
        {
        }*/
        // Static method for taking the next sequence value
        public static int NextValue()
        {
            currentValue++;
            return currentValue;
        }

    }
    class SequenceManipulating
    {
        static void Main()
        {
            Console.WriteLine("Sequence[1...5]: {0}, {1}, {2}, {3}, {4}",
            Sequence.NextValue(), Sequence.NextValue(),Sequence.NextValue(),
            Sequence.NextValue(), Sequence.NextValue());
        }
    }
}

//We can also separate the two classes into separate .CS files by manually adding
//a new file and having each separate class in its own file




//*********************************************************************
// Day 3: Using objects/Intro to Objects/Object Oriented Programming
//*********************************************************************
//Inheritance...
//In C#, it is possible to inherit fields and methods from one class to another.
//We group the "inheritance concept" into two categories:

//Derived Class (child) -the class that inherits from another class
//Base Class(parent) - the class being inherited from

//In the example below, the Car class (child)inherits the fields and methods
//from the Vehicle class (parent):

namespace ClassInheritance
{
    class Vehicle  // base class (parent)
    {
        public string brand = "Ford";  // Vehicle field
        public void honk()             // Vehicle method
        {
            Console.WriteLine("Tuut, tuut!");
```

```
        }
    }

    class Car : Vehicle  // derived class (child)
    {
        public string modelName = "Mustang";  // Car field
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Create a myCar object
            Car myCar = new Car();

            // Call the honk() method (From the Vehicle class) on the myCar object
            myCar.honk();

            // Display the value of the brand field (from the Vehicle class)
            // and the value of the modelName from the Car class
            Console.WriteLine(myCar.brand + " " + myCar.modelName);
        }
    }
}

// if we use "sealed" before the first class it cannot be inherited

// for example just changing the first class above

    sealed class Vehicle  // base class (parent)
    {
        public string brand = "Ford";  // Vehicle field
        public void honk()             // Vehicle method
        {
            Console.WriteLine("Tuut, tuut!");
        }
    }
```