using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

// We cover public, private, protected, constants,

// read-only fields, constructors, setters, getters,

// properties and more on static

namespace CSharpTut

{

 class Animal

 {

 // Define fields that are protected with private

 // Private fields can only be accessed by

 // methods in the class and they are not

 // accessible by subclasses

 // protected fields can only be accessed by

 // methods in the class and by subclasses

 private string name;

 private string sound;

 // You can define constants

 public const string SHELTER = "Derek's Home for Animals";

 // You can define read-only fields that are set

 // at runtime in constructors, but then can't

 // be changed

 public readonly int idNum;

 // Method (Capabilities)

 public void MakeSound()

 {

 Console.WriteLine("{0} says {1}",

 name, sound);

 }

 // Default constructor

 public Animal()

 :this("No Name", "No Sound") { }

 // Constructor called if only name is passed

 // this passes the parameters to the next

 // constructor

 public Animal(string name)

 : this(name, "No Sound") { }

 // Constructor that receives parameters

 public Animal(string name,

 string sound)

 {

 SetName(name);

 Sound = sound;

 // Increment the number of animals

 // property

 NumOfAnimals = 1;

 // Define the read-only value which is

 // the same for all Animals

 Random rnd = new Random();

 idNum = rnd.Next(1, 2147483640);

 }

 // Setters (Mutators) protect the fields

 // from receiving bad data

 public void SetName(string name)

 {

 // Any checks if any character in the string

 // is a number and if so returns true

 // Since we won't allow numbers we will

 // protect our data

 if (!name.Any(char.IsDigit))

 {

 this.name = name;

 } else

 {

 // We have this duplicated code because

 // everything isn't a property

 this.name = "No Name";

 Console.WriteLine("Name can't contain numbers");

 }

 }

 // Getters (Accessors) can provide output

 // aside from the value stored

 public string GetName()

 {

 return name;

 }

 // The preferred way to define getters and

 // setters is through properties

 public string Sound

 {

 get { return sound; }

 set

 {

 // value is assigned the value passed in

 if (value.Length > 10)

 {

 sound = "No Sound";

 Console.WriteLine("Sound is too long");

 } else

 {

 sound = value;

 }

 }

 }

 // You can have the getters and setters

 // generated for you like this and also

 // set the default value

 public string Owner { get; set; } = "No Owner";

 // You can also define static properties

 public static int numOfAnimals = 0;

 public static int NumOfAnimals

 {

 get { return numOfAnimals; }

 set { numOfAnimals += value; }

 }

 }

}