PN

PASSERELLES NUMERIQUES

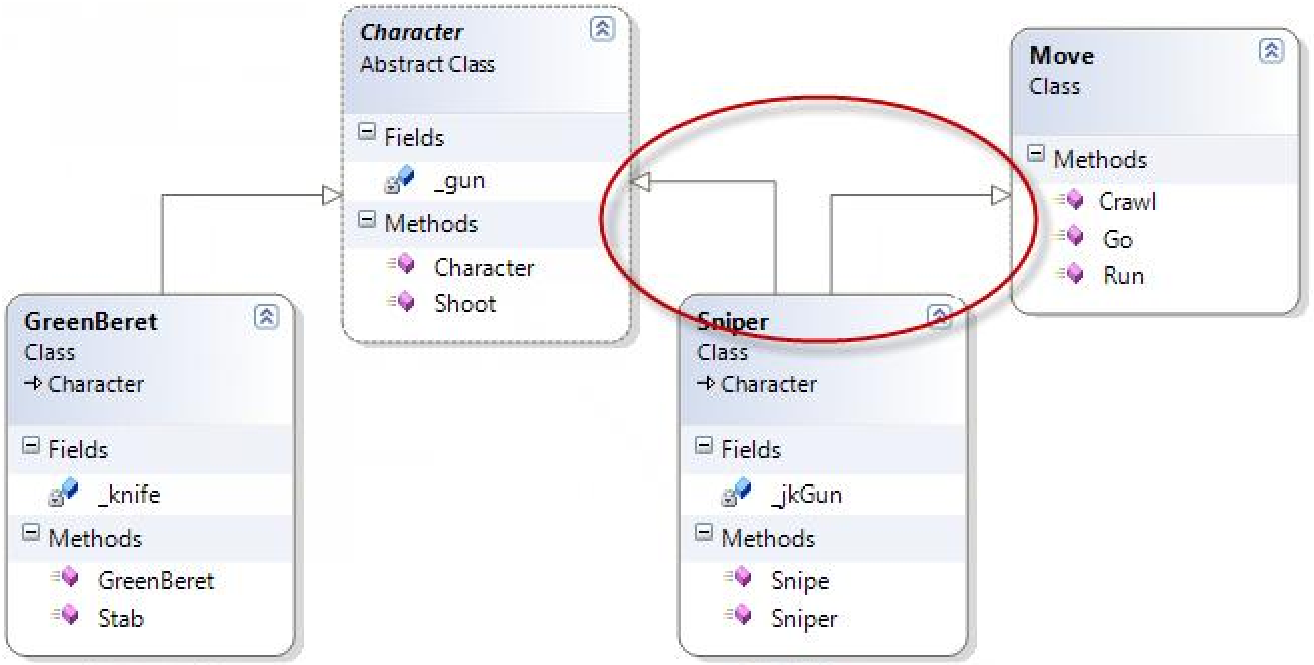
Microsoft®

.NET

LESSON 2. INTERFACES

- Introduction
- Definion
- Purpose
- Snippet
- Interface and Multi Inheritance

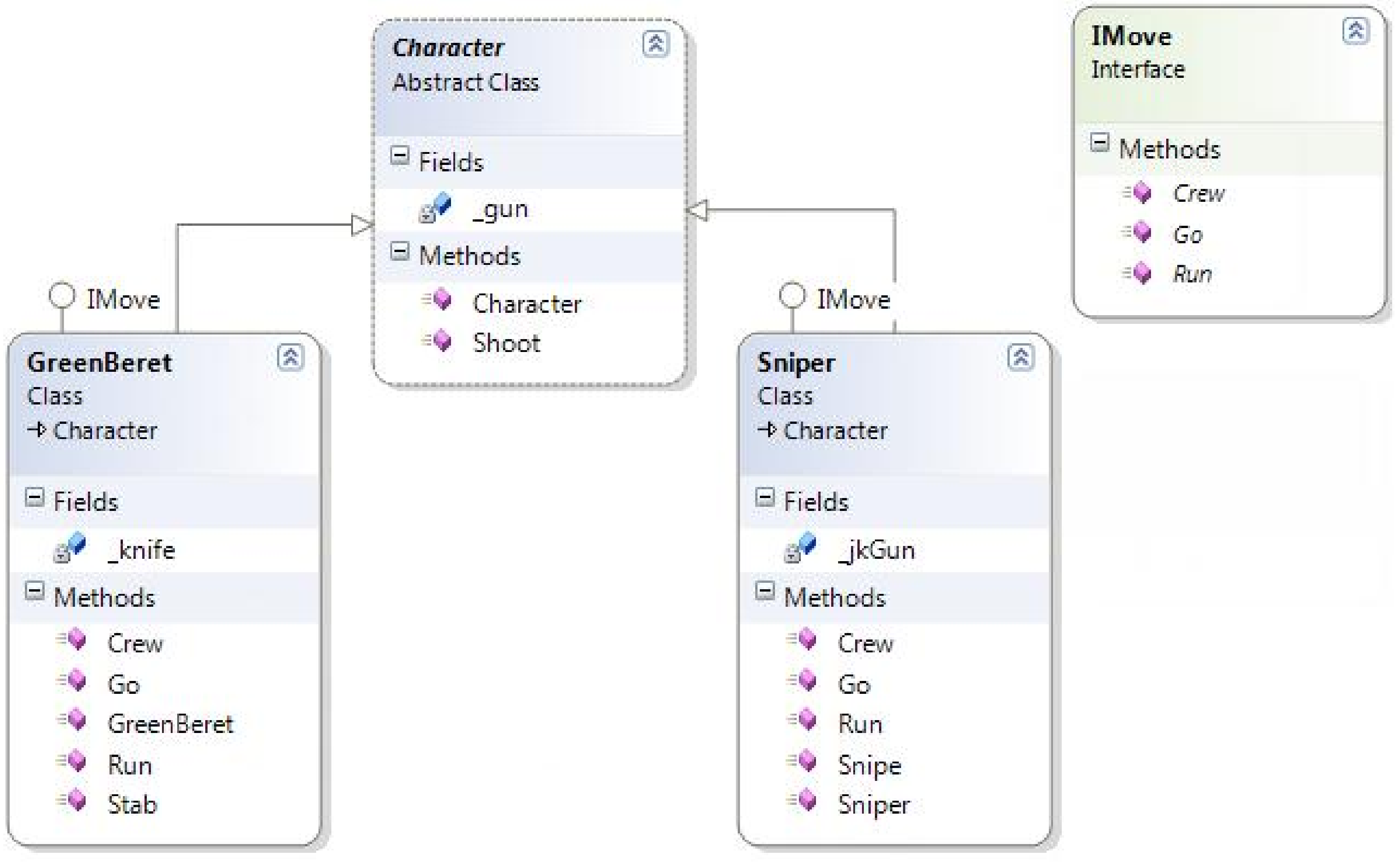
Introduction



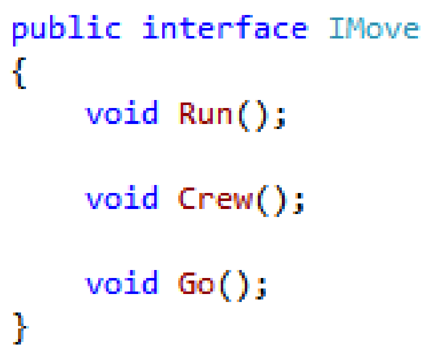
Introduction

- Interface declaration is like a class declaration, but it provides no implementation for its members, since all its members are implicitly abstract.
- Classes and structs that implement the interface must implement all members of the interface.
- In interface can contain only methods, properties, event, and indexers.
- An interface can not be instantiated directly.

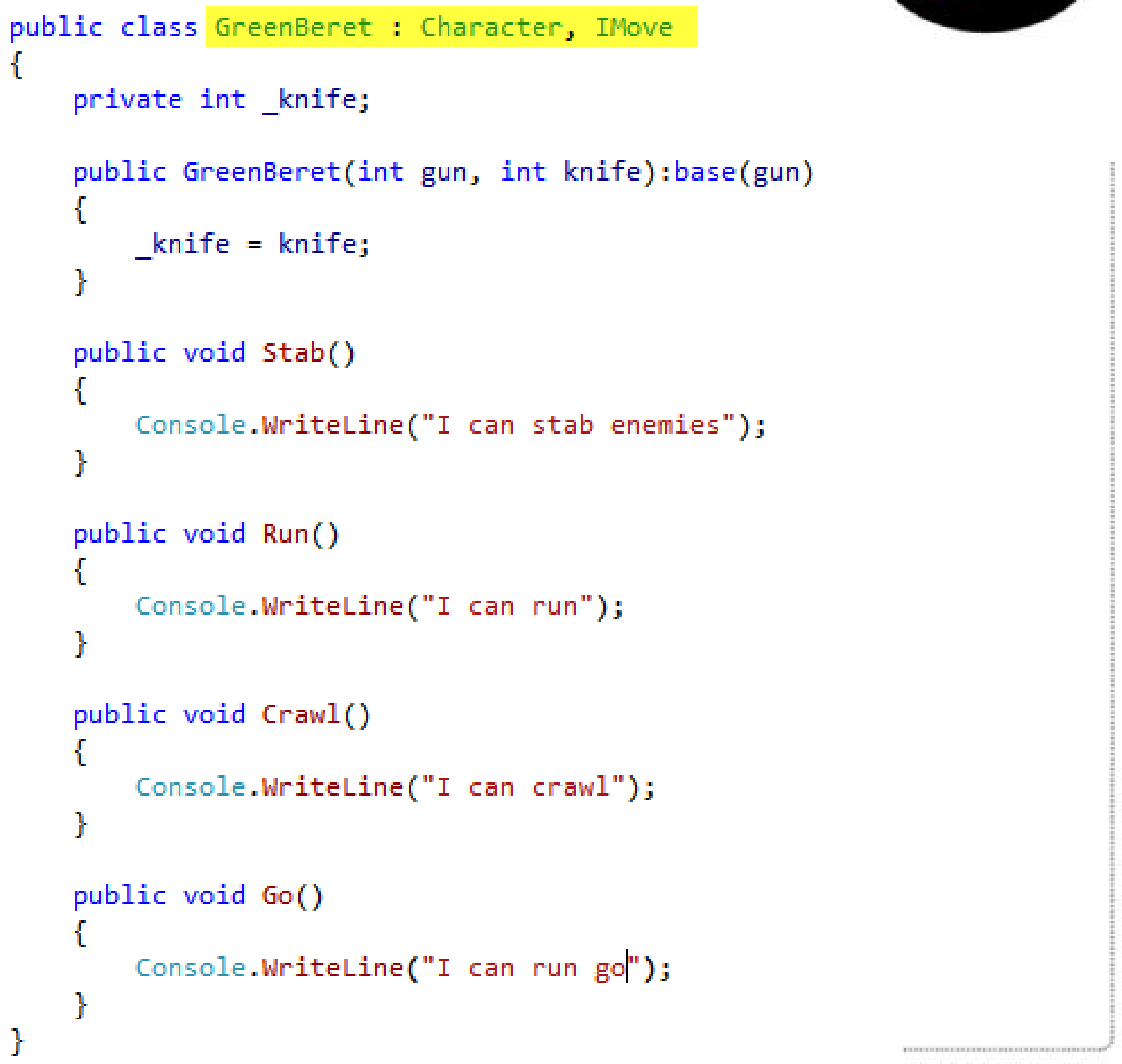
Snippet


Character
Abstract Class
Fields
_gun
Methods
Character
Shoot
IMove
Interface
Methods
Crew
Go
Run
IMove
GreenBeret
Class
Character
Fields
_knife
Methods
Crew
Go
GreenBeret
Run
Stab
IMove
Sniper
Class
Character
Fields
_jkGun
Methods
Crew
Go
Run
Snipe
Sniper

Snippet



```
public interface IMove
{
    void Run();

    void Crew();

    void Go();
}
```

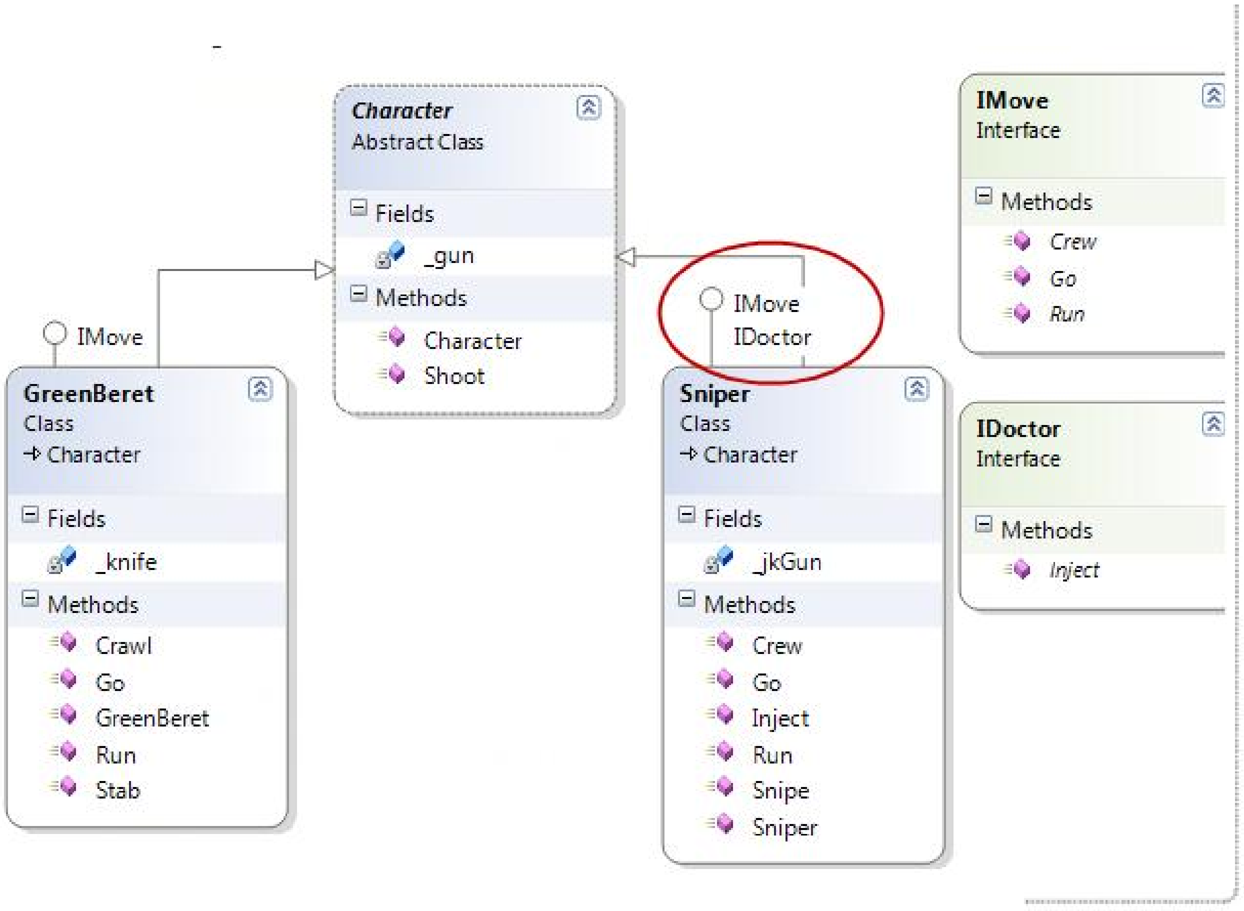


```
public class GreenBeret : Character, IMove
{
    private int _knife;

    public GreenBeret(int gun, int knife):base(gun)
    {
        _knife = knife;
    }

    public void Stab()
    {
        Console.WriteLine("I can stab enemies");
    }

    public void Run()
    {
        Console.WriteLine("I can run");
    }

    public void Crawl()
    {
        Console.WriteLine("I can crawl");
    }

    public void Go()
    {
        Console.WriteLine("I can run go");
    }
}
```

Interface and Multi inheritance

- Classes (and structs) can implement more than one interface
- An interface can inherit from one or more base interfaces

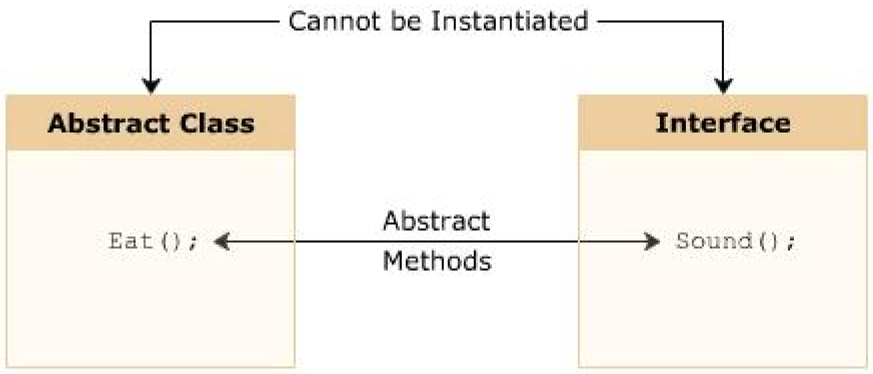
=> **Multiple inheritance**



ABSTRACT CLASS AND INTERFACE

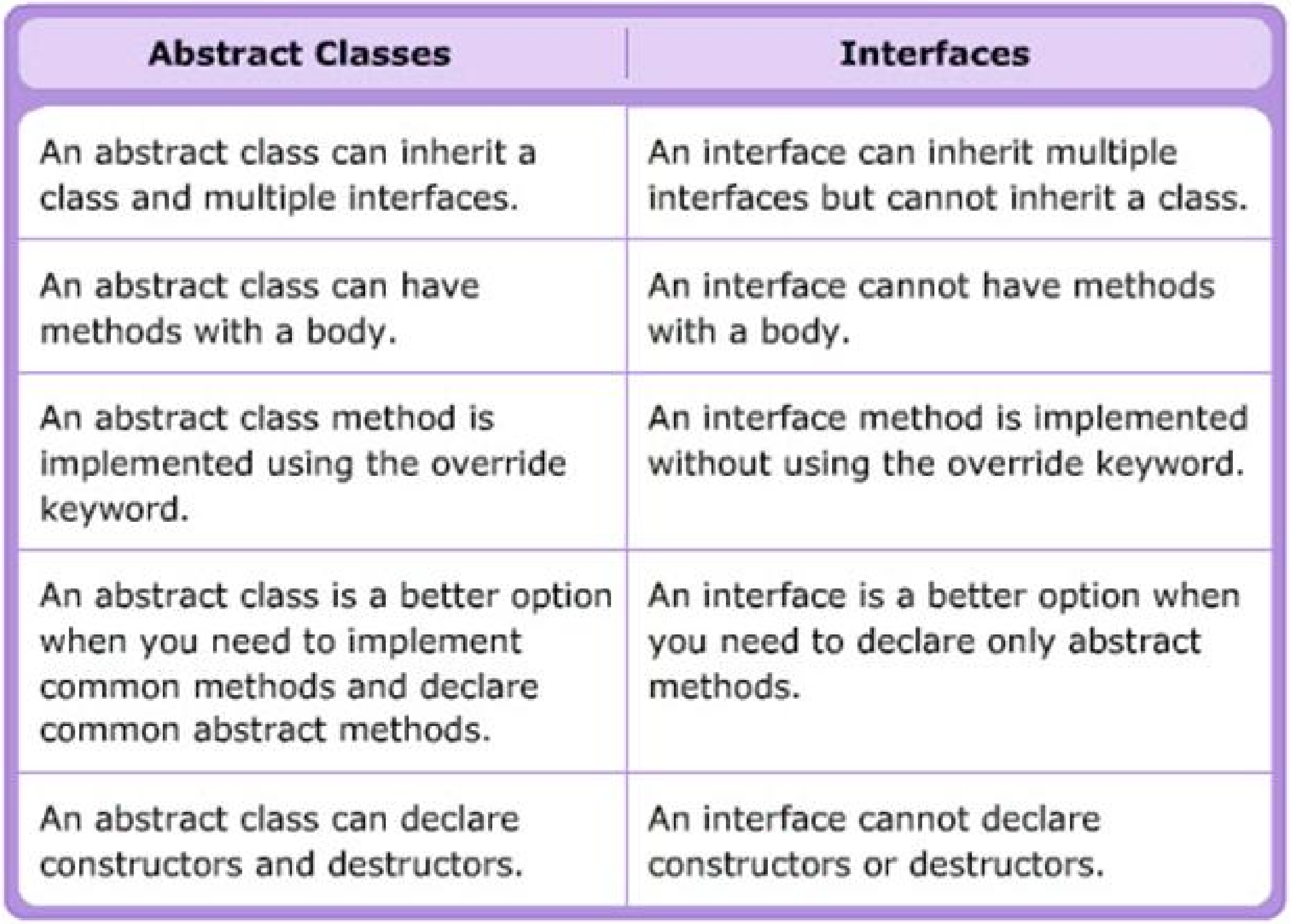
- **Similarities**

- Neither an abstract class nor an interface can be **instantiated**
- Both, asbstract class as well as interfaces, contain **abstract methods** which are implement by the inheriting subclass
- Both, abstract class as well as interface, can **inherit multiple interfaces**



ABSTRACT CLASS AND INTERFACE

- **Differences**

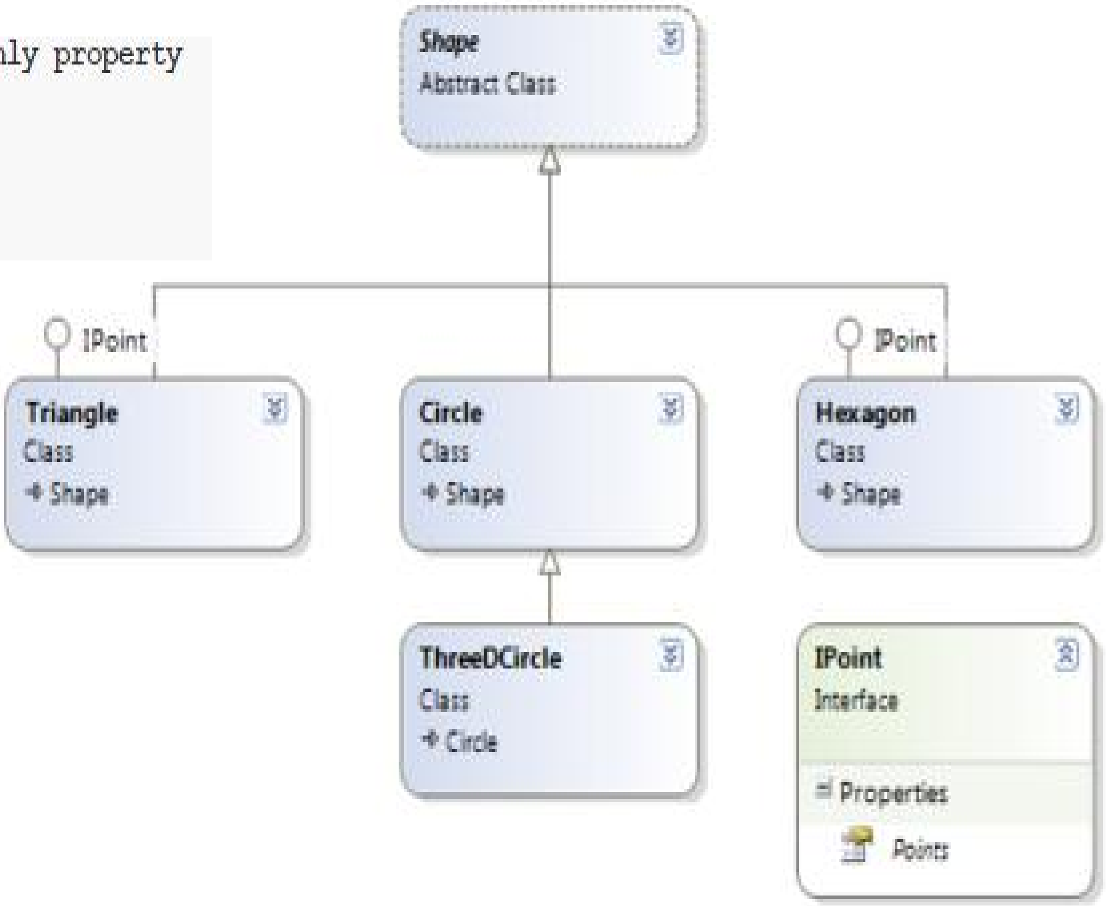


Abstract Classes	Interfaces
An abstract class can inherit a class and multiple interfaces.	An interface can inherit multiple interfaces but cannot inherit a class.
An abstract class can have methods with a body.	An interface cannot have methods with a body.
An abstract class method is implemented using the override keyword.	An interface method is implemented without using the override keyword.
An abstract class is a better option when you need to implement common methods and declare common abstract methods.	An interface is a better option when you need to declare only abstract methods.
An abstract class can declare constructors and destructors.	An interface cannot declare constructors or destructors.

Exercise

- **Interface**

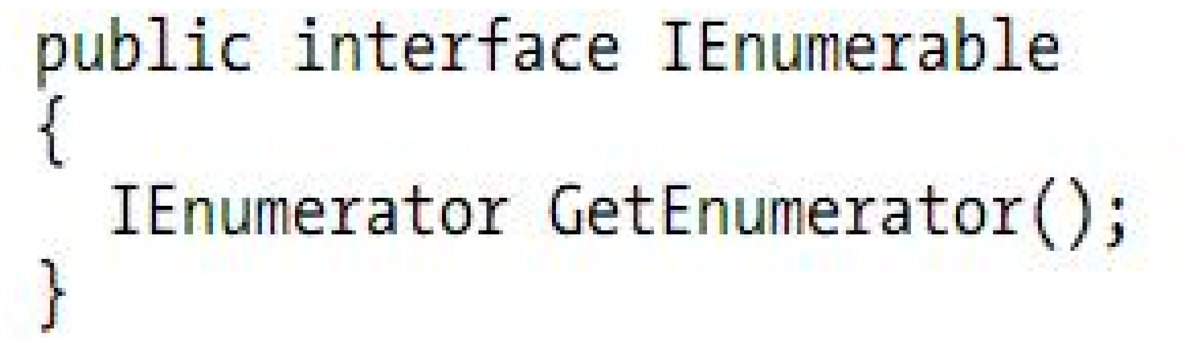
```
// The IPoint behavior as a read-only property
public interface IPoint
{
    byte Points { get; }
}
```



LESSION 3. IENUMERABLE AND IENUMERATOR INTERFACES

IEnumerable Interface

- **Definition**

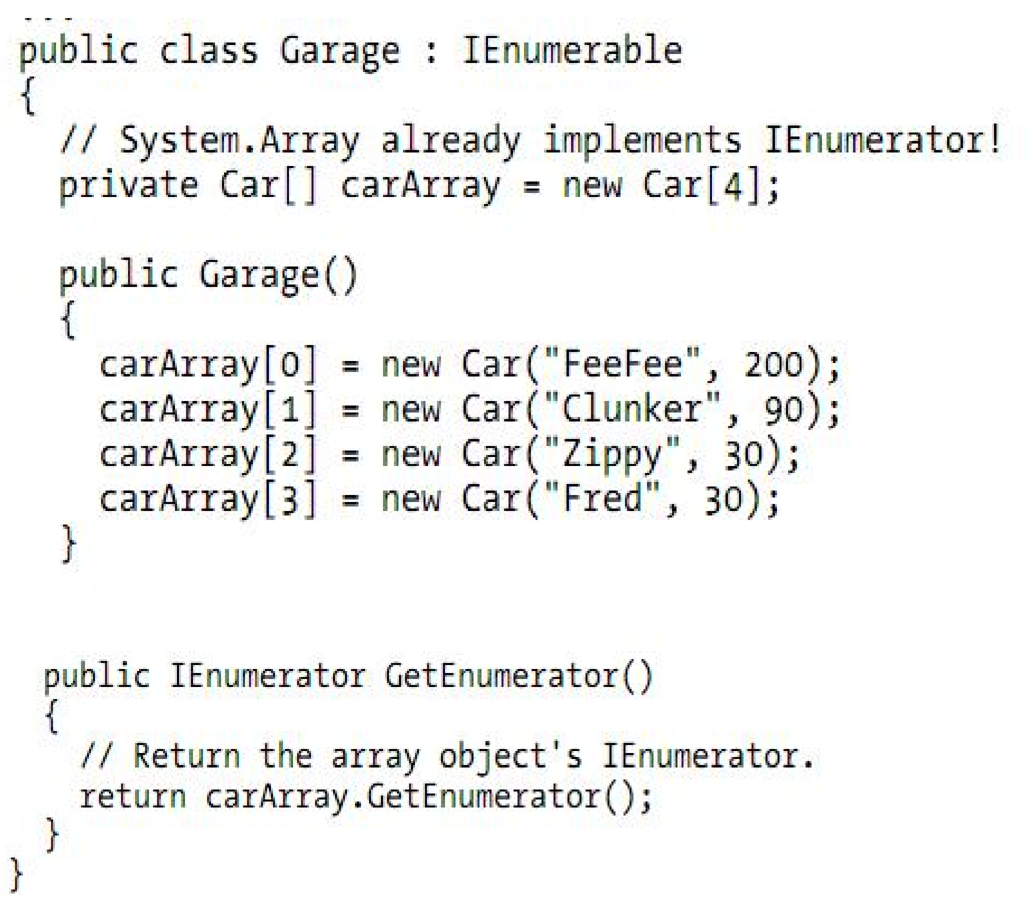


```
public interface IEnumerable
{
   IEnumerator GetEnumerator();
}
```

- **The GetEnumerator() method returns a reference to yet another interface named System.Collections.IEnumerator**

IEnumerable Interface

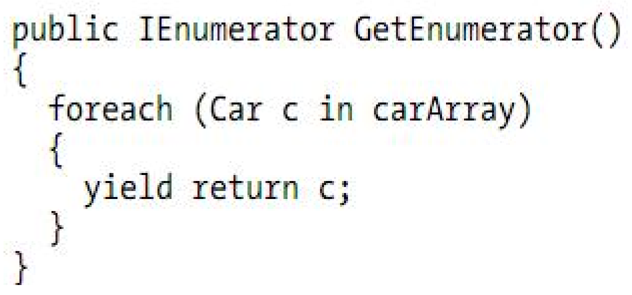
- **Implement IEnumerable interface**



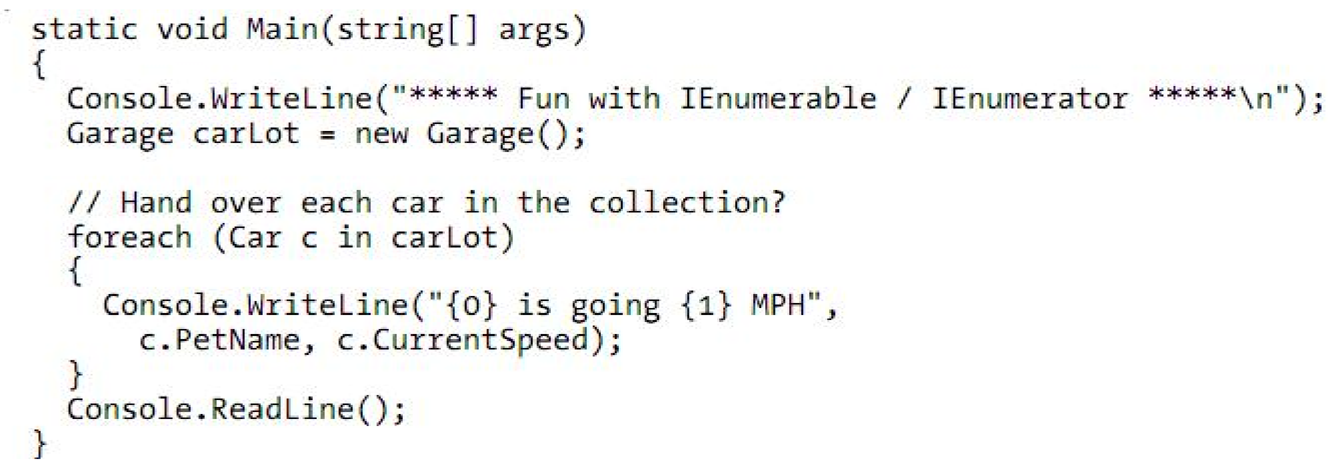
```
...
public class Garage : IEnumerable
{
  // System.Array already implements IEnumerator!
  private Car[] carArray = new Car[4];

  public Garage()
  {
    carArray[0] = new Car("FeeFee", 200);
    carArray[1] = new Car("Clunker", 90);
    carArray[2] = new Car("Zippy", 30);
    carArray[3] = new Car("Fred", 30);
  }


 public IEnumerator GetEnumerator()
 {
   // Return the array object's IEnumerator.
   return carArray.GetEnumerator();
 }
}
```

IEnumerable Interface

- **The yield keyword is used to specify the value (or values) to be returned to the caller’s foreach construct.**



```
public IEnumerator GetEnumerator()
{
  foreach (Car c in carArray)
  {
    yield return c;
  }
}
```

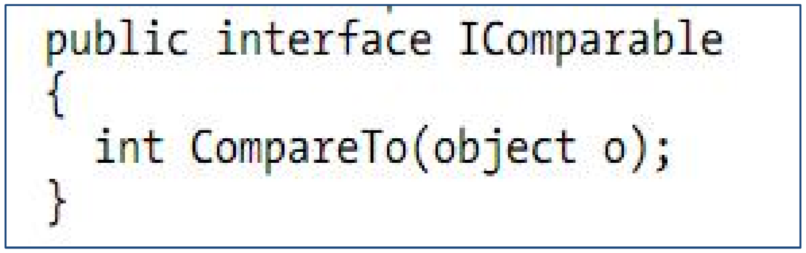


```
static void Main(string[] args)
{
  Console.WriteLine("***** Fun with IEnumerable / IEnumerator *****\n");
  Garage carLot = new Garage();

  // Hand over each car in the collection?
  foreach (Car c in carLot)
  {
    Console.WriteLine("{0} is going {1} MPH",
      c.PetName, c.CurrentSpeed);
  }
  Console.ReadLine();
}
```

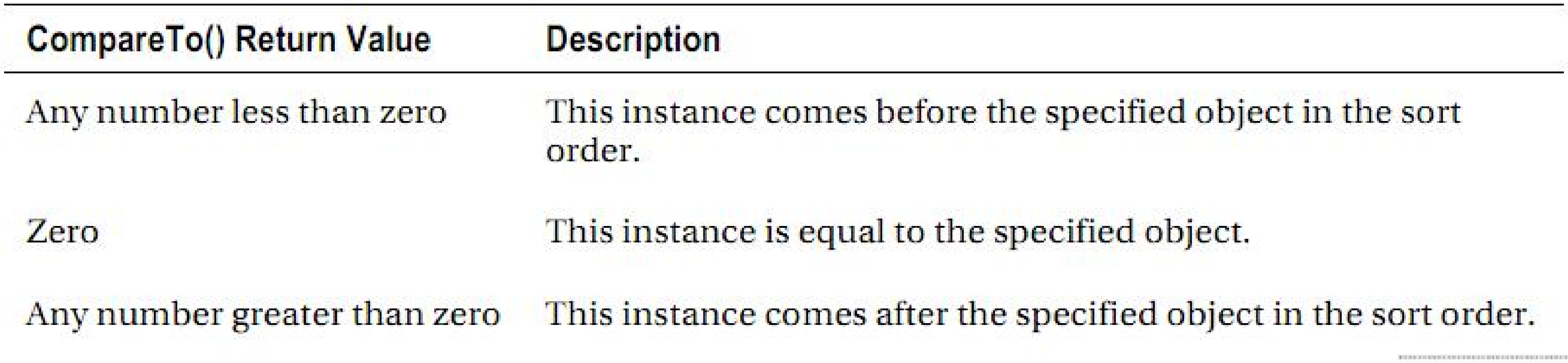
LESSION 4. ICOMPARABE & ICOMPARER INTERFACES

IComparable Interface

- **The System.IComparable interface specifies a behavior that allows an object to be sorted based on some specified key.**
- **Definition:**



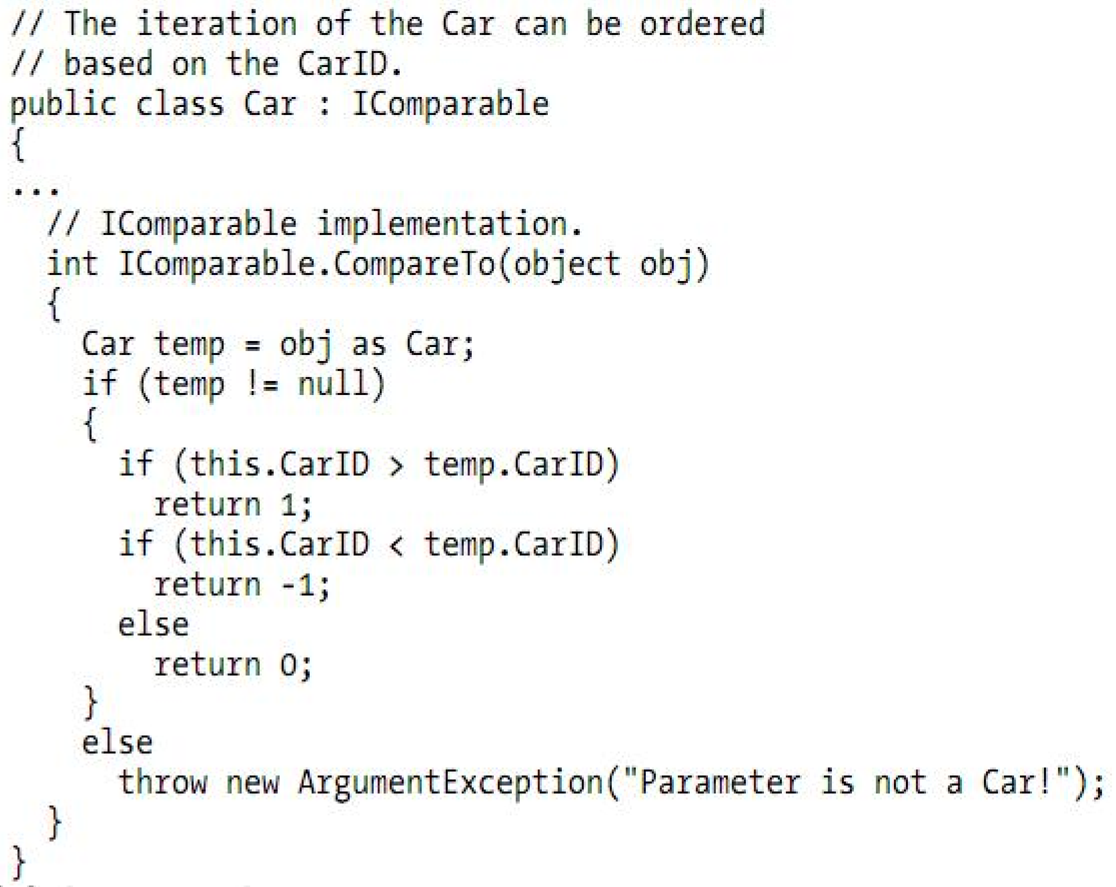
```
public interface IComparable
{
   int CompareTo(object o);
}
```



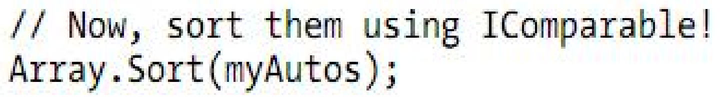
CompareTo() Return Value	Description
Any number less than zero	This instance comes before the specified object in the sort order.
Zero	This instance is equal to the specified object.
Any number greater than zero	This instance comes after the specified object in the sort order.

IComparable Interface

- **Implement IComparable**



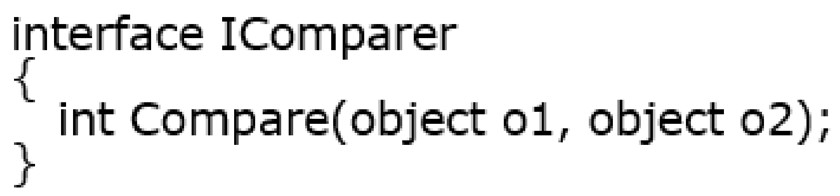
```
// The iteration of the Car can be ordered
// based on the CarID.
public class Car : IComparable
{
...
  // IComparable implementation.
  int IComparable.CompareTo(object obj)
  {
    Car temp = obj as Car;
    if (temp != null)
    {
      if (this.CarID > temp.CarID)
        return 1;
      if (this.CarID < temp.CarID)
        return -1;
      else
        return 0;
    }
    else
      throw new ArgumentException("Parameter is not a Car!");
  }
}
```



```
// Now, sort them using IComparable!
Array.Sort(myAutos);
```

IComparer Interface

- Specifying Multiple Sort Orders
- Icomparer interface is defined within the System.Collections namespace as follows:

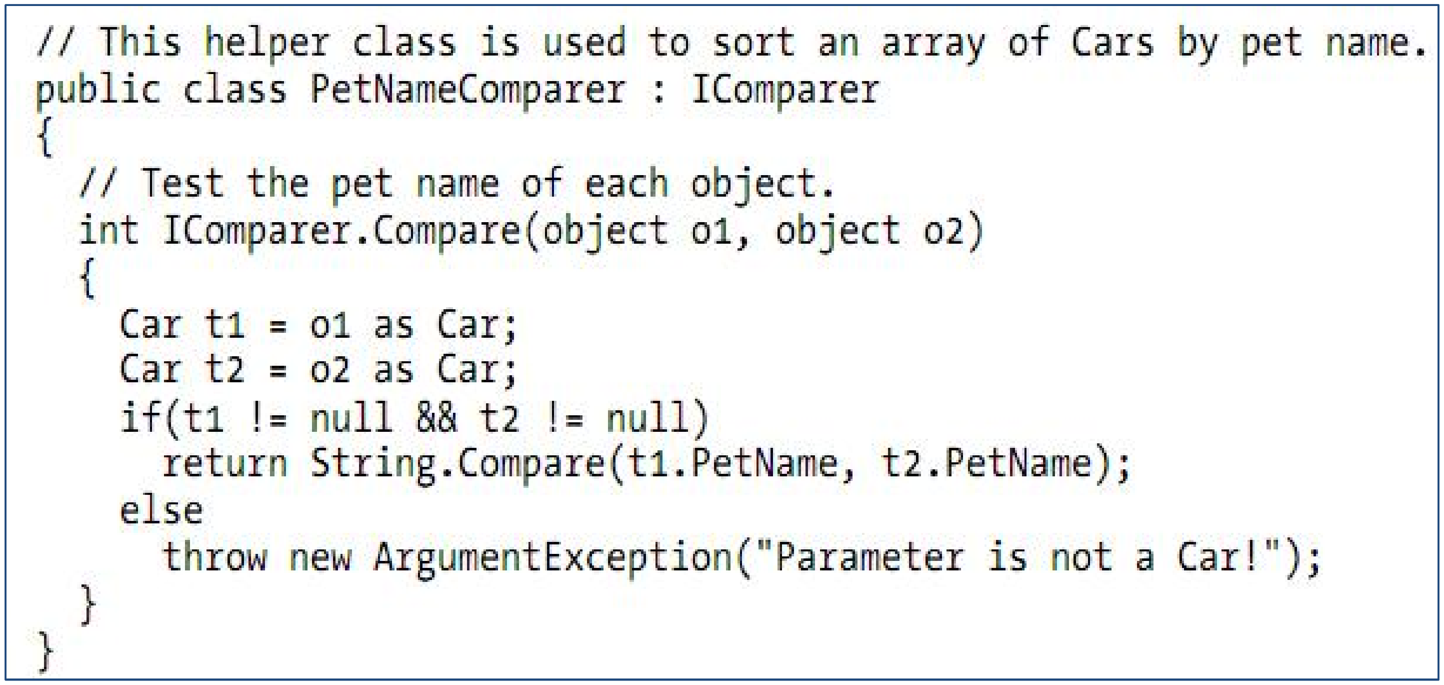


```
interface IComparer
{
    int Compare(object o1, object o2);
}
```

- You must implement this interface on any number of helper classes, one for each sort order

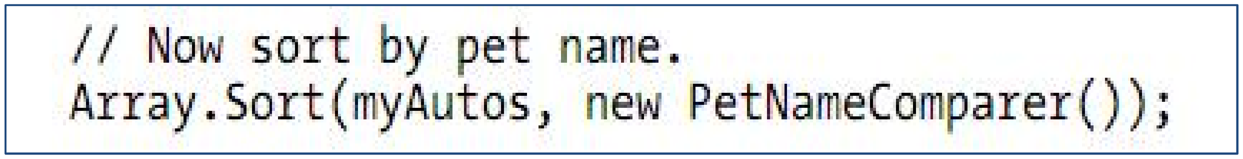
IComparer Interface

- **Ex:**



```
// This helper class is used to sort an array of Cars by pet name.
public class PetNameComparer : IComparer
{
  // Test the pet name of each object.
  int IComparer.Compare(object o1, object o2)
  {
    Car t1 = o1 as Car;
    Car t2 = o2 as Car;
    if(t1 != null && t2 != null)
      return String.Compare(t1.PetName, t2.PetName);
    else
      throw new ArgumentException("Parameter is not a Car!");
  }
}
```

- **Caller method**



```
// Now sort by pet name.
Array.Sort(myAutos, new PetNameComparer());
```

IComparer Interface

- **Custom static property in order to help the object user**
- **along when sorting by a specific data point.**

```
public class Car : IComparable
{
  ...
  // Property to return the PetNameComparer.
  public static IComparer SortByPetName
  { get
            {
                return (IComparer)new PetNameComparer();
            }
   }
}
```

- **Caller method**

```
// Sorting by pet name made a bit cleaner.
Array.Sort(myAutos, Car.SortByPetName);
```