Class and Interface

.NET

Class *is the most fundamental of C#'s types. A* ***class*** *is a data structure that combines state (fields) and actions (methods) into a single unit.*
Classes *support inheritance and polymorphism. A* ***Class*** *is a blueprint for a* ***Class*** *Object.*

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects

Class

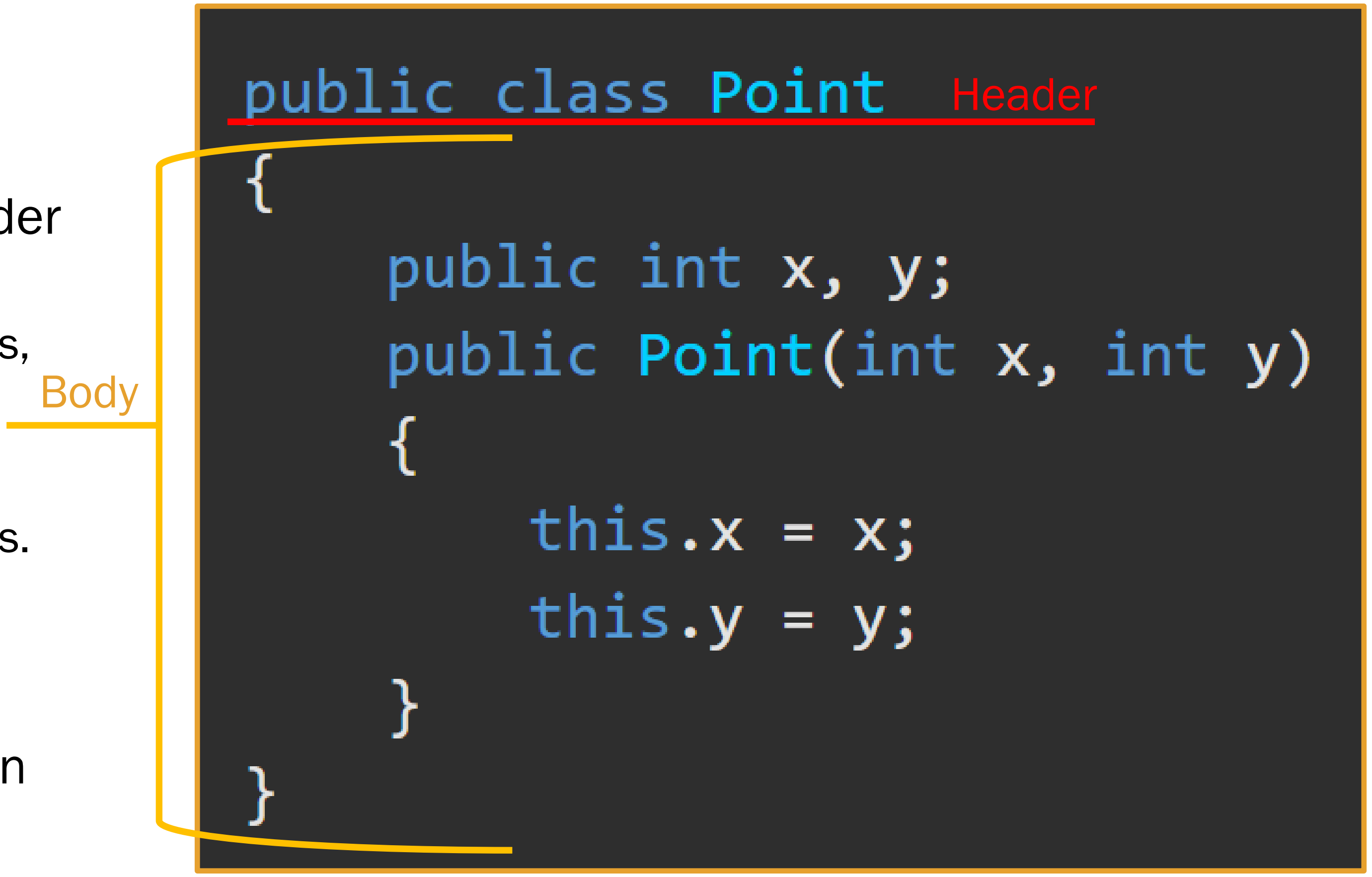
https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects

Classes are defined using class declarations.

A class declaration starts with a header that specifies

- the attributes and modifiers of the class,
- the name of the class,
- the base class (if given), and
- the interfaces implemented by the class.

The header is followed by the class ***body***, which consists of a list of member declarations written between curleyBrackets { }.



```
public class Point
{
    public int x, y;
    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

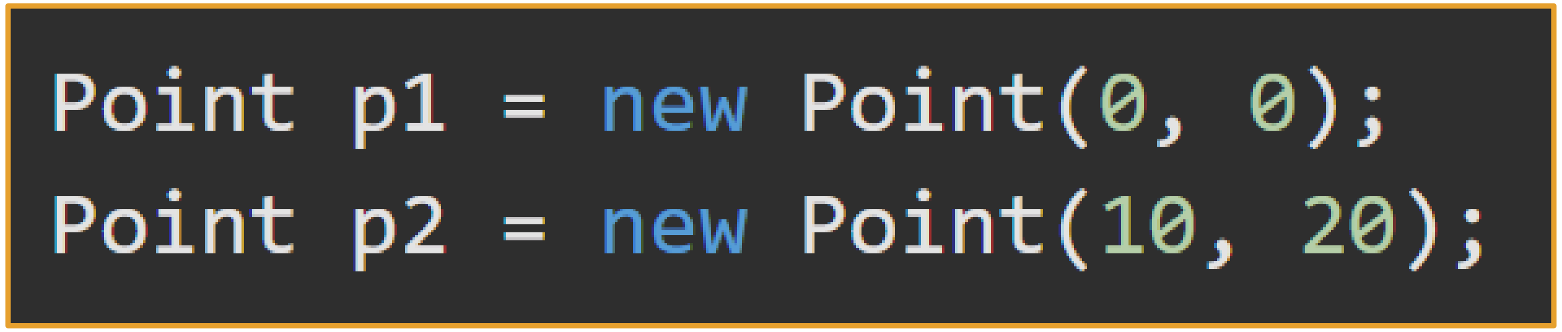
Class – Instance Instantiation

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects

Instances of classes are created using the ***new*** operator, which

- allocates memory for a new instance,
- invokes a constructor to initialize the instance
- returns a reference to the instance.

The memory occupied by an object is automatically reclaimed by the ***Garbage Collector*** when the object is out of scope (is no longer reachable).



```
Point p1 = new Point(0, 0);
Point p2 = new Point(10, 20);
```

Class - Members

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects

Members of a class are:

- Constructors - To initialize instances of the class
- Constants - Constant values
- Fields – Variables
- Methods – Computations/actions that can be performed
- Properties – Fields combined with the actions associated with reading/writing them
- Types - Nested types declared by the class

Class members can be:

- static - belong to classes. Envoked with: ClassName.MethodName();
- instance - belong to ***instances*** of classes. Envoked with: InstanceName.MethodName();

Accessibility of Classes

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

- ***Classes*** and ***structs*** declared directly in a ***namespace*** (not nested in another class or struct) can be either ***public*** or ***internal***.
- ***Derived*** classes can't have greater accessibility than their base class.
- ***Internal*** is default if no access modifier is specified.

Class – Member Accessibility

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/types#classes-and-objects
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

Access Modifiers control the regions of a program that can access a class member.

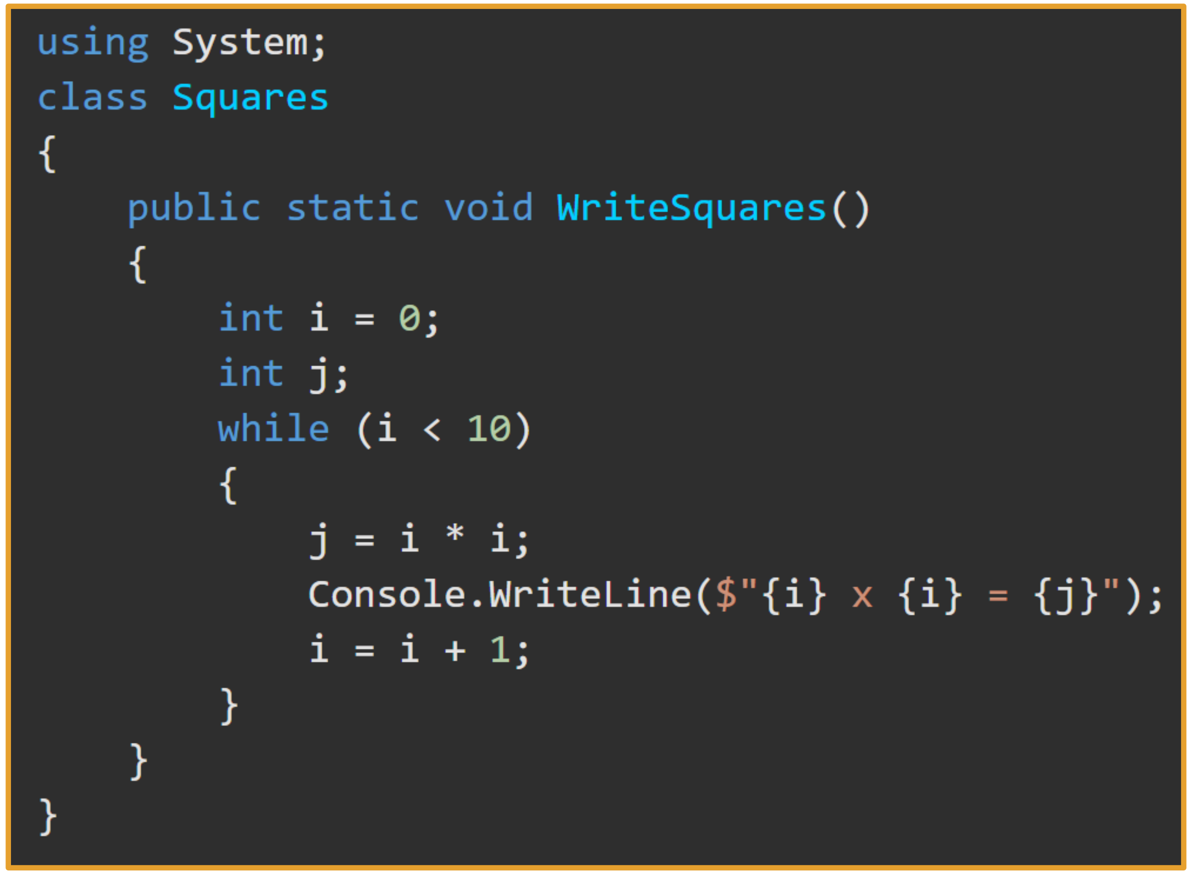
- private - This class only.
- protected - accessed by code in the same class, or in a derived class.
- private protected - accessed by the same class or derived classes only when in the same assembly.
- internal – accessed by any code in the same assembly (.exe, .dll).
- protected internal - accessed by any code in the same assembly or from within a derived class in a different assembly.
- public - Access isn't limited.

Class – Local Variables

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects#method-body-and-local-variables

Local variables are declared inside the body of a method. They must have a ***type*** name and a variable name. All variables must be given a default value.

- int == 0;
- string == “”;



```
using System;
class Squares
{
    public static void WriteSquares()
    {
        int i = 0;
        int j;
        while (i < 10)
        {
            j = i * i;
            Console.WriteLine($"{i} x {i} = {j}");
            i = i + 1;
        }
    }
}
```

Interface

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/interfaces
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

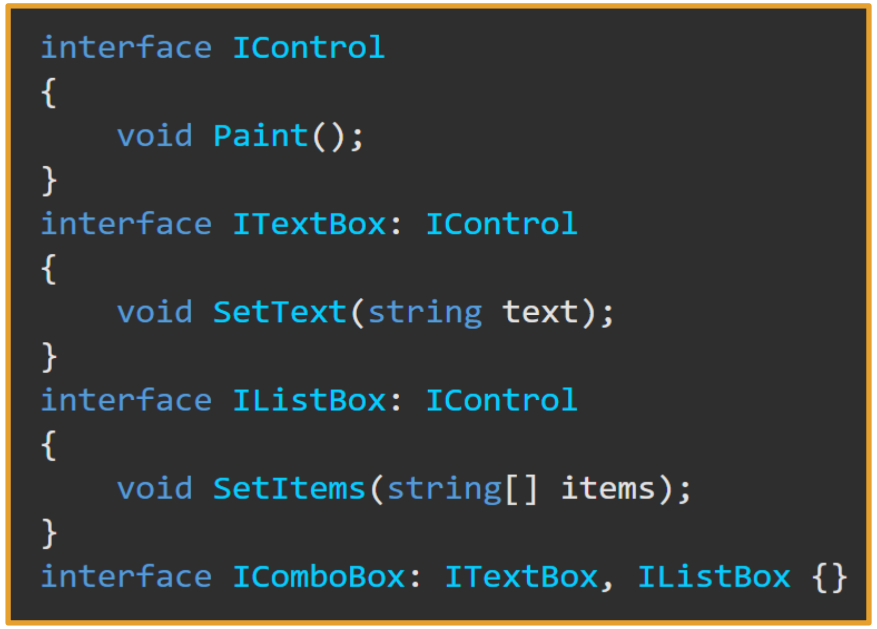
- An *interface* defines a *contract* that must be implemented by classes and structs that use the interface.
- An *interface* can contain methods, properties, events.
- An interface specifies members that must be implemented by classes or structs that implement the interface.
- *Interface* implementation is NOT inheritance.
- An *Interface* is intended to express a "can do" relationship between an interface and its implementing class.
- *Interfaces* are used to simulate *multiple inheritance*.
- An *interface* CAN have *default interface methods*. These methods have implementations and are not required to be implemented by the implementing class. They are useful for when methods are added to an interface after other classes have implemented it already.

```
interface IControl
{
    void Paint();
}

interface ITextBox : IControl
{
    void SetText(string text);
}

interface IListBox : IControl
{
    void SetItems(string[] items);
}

interface IComboBox : ITextBox, IListBox { }
```

Interface

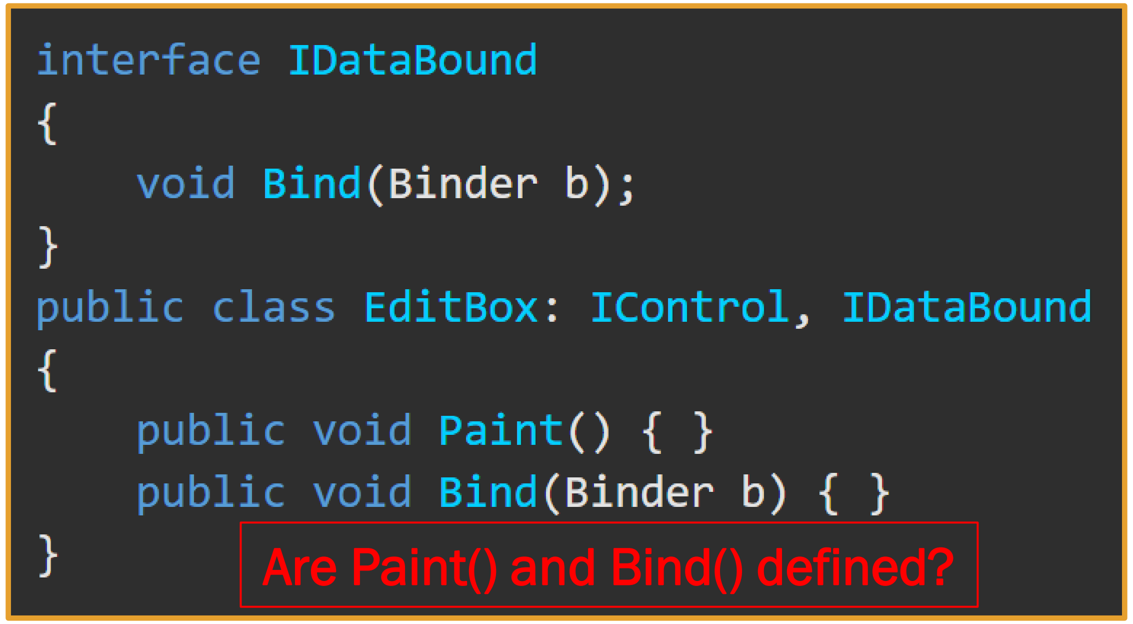
https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/interfaces

Interfaces may employ multiple inheritance.



```
interface IControl
{
    void Paint();
}
interface ITextBox: IControl
{
    void SetText(string text);
}
interface IListBox: IControl
{
    void SetItems(string[] items);
}
interface IComboBox: ITextBox, IListBox {}
```

Classes and structs can implement multiple interfaces.



```
interface IDataBound
{
    void Bind(Binder b);
}
public class EditBox: IControl, IDataBound
{
    public void Paint() { }
    public void Bind(Binder b) { }
}
```

Are Paint() and Bind() defined?

Class – Type Parameters

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects#type-parameters

Class *Type* Parameters:

- are used to define a ***generic*** class type.
- follow the class name and are inside **< >.**
- are used to define the ***types*** that the members of the class act on.

```
public class Pair<TFirst,TSecond>
{
    public TFirst First;
    public TSecond Second;
}
```

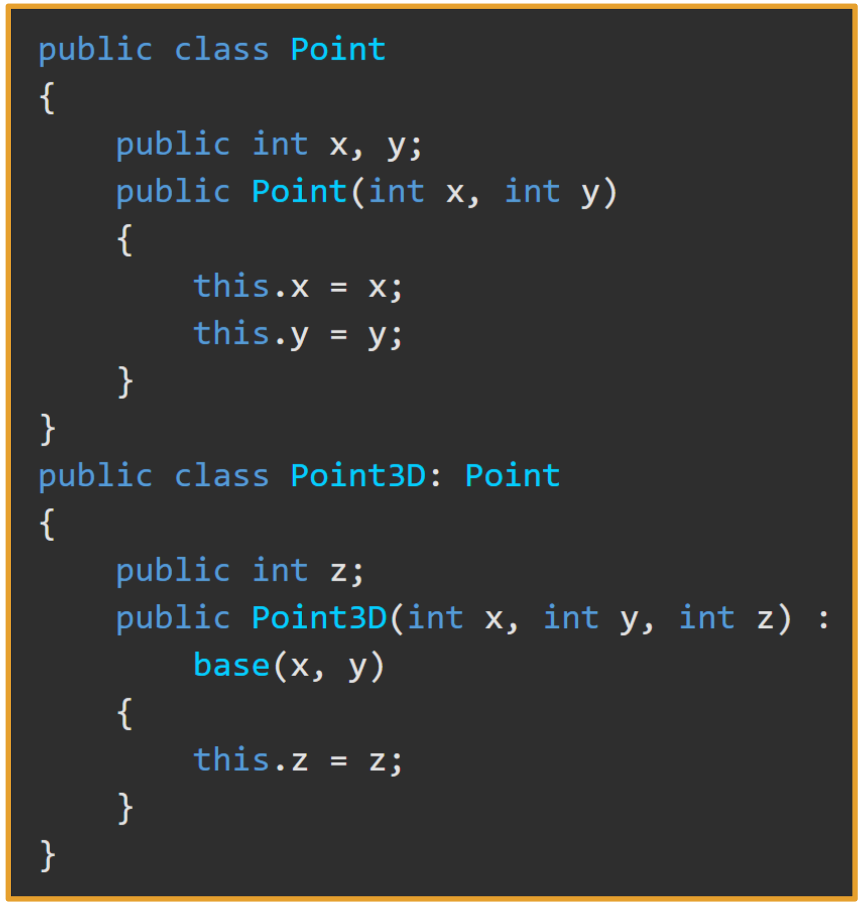
```
Pair<int,string> pair = new Pair<int,string> { First = 1, Second = "two" };
int i = pair.First;     // TFirst is int
string s = pair.Second; // TSecond is string
```

Class – Base (inherited) Classes

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects#base-classes

A class declaration specifies ***inheritance*** of a ***base*** class by following the class name (and type parameters) with…

: <baseClassName>



```
public class Point
{
    public int x, y;
    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
public class Point3D: Point
{
    public int z;
    public Point3D(int x, int y, int z) :
        base(x, y)
    {
        this.z = z;
    }
}
```