زنگ سی‌شارپ – قسمت سی و ششم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبل با Indexer آشنا شدید، در این قسمت با Properties آشنا خواهیم شد. Property یکی دیگر از اعضای کلاس است. برای این‌که با اساس کار Properties آشنا شوید به مثال ساده‌ی زیر توجه کنید:

```
using System;
class MyClass
{
    private int ID;

    public void SetID(int id)
    {
        if (id >= 0 && id <= 10)
        {
            ID = id;
        }
    }
    public int GetID()
    {
        return ID;
    }
}
class GetAndSet
{
    static void Main()
    {
        MyClass ob = new MyClass();
        ob.SetID(5);
        Console.WriteLine("ID: " + ob.GetID());

        ob.SetID(20);
        Console.WriteLine("ID: " + ob.GetID());

        ob.SetID(9);
        Console.WriteLine("ID: " + ob.GetID());
    }
}
```

همان‌طور که می‌بینید یک فیلد به اسم ID داریم که private بوده و دسترسی به آن فقط برای اعضای کلاس خودش مجاز است. به‌منظور دسترسی به این فیلد، دو متد public در نظر گرفته‌ایم. توسط متد GetID() مقدار ID را return کرده‌ایم و توسط متد SetID() یک مقدار کنترل شده را به ID اختصاص داده‌ایم. در واقع تنها مقادیر بین صفر و ده می‌توانند به ID اختصاص داده شوند. همان‌طور که می‌بینید، توسط این دو متد توانستیم کنترل کاملی روی مقداردهی فیلد مورد نظر داشته باشیم.

کاری که Property انجام می‌دهد دقیقاً همین است: کنترل دسترسی و مقداردهی به فیلد. Property مشابه متدهای بالا عمل کرده و کنترل دسترسی و مقداردهی یک فیلد را در دست می‌گیرد. Property مانند Indexer از get accessor و set accessor استفاده می‌کند تا مقداری را در یک متغیر set و یا مقداری را از آن get کند.

فرم کلی یک property به‌شکل زیر است:

```
type name
{
    get {
        // get accessor code
    }
    set {
        // set accessor code
    }
}
```

در این‌جا type مشخص‌کننده‌ی نوع property (مثل int) و name مشخص‌کننده‌ی نام آن است. set accessor به‌طور اتوماتیک یک پارامتر به اسم value دریافت می‌کند که شامل مقداری است که به property اختصاص داده می‌شود. دقت کنید که property ها storage location (محل ذخیره‌سازی) تعریف نکرده و درواقع دسترسی به یک فیلد را مدیریت می‌کنند. یک property خودش فیلد تعریف نمی‌کند و فیلد باید به‌طور جداگانه تعریف شود (به استثنای auto-implemented property که به شرح آن خواهیم پرداخت).

مثال بالا را این‌بار توسط property انجام می‌دهیم:

```
using System;
class MyClass
{
    private int id;

    public int ID
    {
        get
        {
            return id;
        }
        set
        {
            //if (value >= 0 && value <= 10)
            //{
            //    id = value;
            //}

            id = value;
        }
    }
}
class PropDemo
```

```
{
    static void Main()
    {
        MyClass ob = new MyClass();

        ob.ID = 5;
        Console.WriteLine("ID: " + ob.ID);

        ob.ID = 20;
        Console.WriteLine("ID: " + ob.ID);

        ob.ID = 9;
        Console.WriteLine("ID: " + ob.ID);
    }
}
```

به تعریف property و چگونگی استفاده از آن توجه کنید. همان‌طور که می‌بینید به‌شکلی مشابه با فیلد، از property استفاده کرده‌ایم. در تعریف property در قسمت get مقدار id را return کرده‌ایم و در قسمت set پارامتر value (که شامل مقداری است که به property اختصاص داده می‌شود) را به id اختصاص داده‌ایم. قسمتی که در برنامه comment شده است نشان می‌دهد که چگونه می‌توانید بر روی مقداری که قرار است به فیلد اختصاص داده شود کنترل داشته باشید.

در مثال زیر چگونگی استفاده از property را بهتر می‌بینید:

```
using System;
class Car
{
    public Car(string name, string style,
        int seatingCapacity, string color, int price)
    {
        this.name = name;
        this.style = style;
        this.seatingCapacity = seatingCapacity;
        this.color = color;
        this.price = price;
    }

    private string name;
    public string Name
    {
        get
        {
            return name;
        }
    }

    private string style;
    public string Style
    {
        get
        {
            return style;
        }
    }
```

```
    private int seatingCapacity;
    public int SeatingCapacity
    {
        get
        {
            return seatingCapacity;
        }
    }

    private string color;
    public string Color
    {
        get
        {
            return color;
        }
    }

    private int price;
    public int Price
    {
        get
        {
            return price;
        }
    }

    private int currentSpeed;
    public int Speed
    {
        get
        {
            return currentSpeed;
        }
        private set
        {
            if (value >= 0 && value <= 300)
                currentSpeed = value;
            else
                currentSpeed = 0;
        }
    }

    public void Accelerate(int amount)
    {
        if (amount > 0)
            Speed += amount;
    }
    public void Brake()
    {
        Speed = 0;
    }

}
class PropDemo
{
    static void Main()
    {
        Car myCar = new Car("Jaguar F Type 5.0 V8 S",
            "Convertibles", 2, "Red", 16100000);
```

```
        Console.WriteLine("Name:\t\t" + myCar.Name);
        Console.WriteLine("Style:\t\t" + myCar.Style);
        Console.WriteLine("Capacity:\t" + myCar.SeatingCapacity);
        Console.WriteLine("Color:\t\t" + myCar.Color);
        Console.WriteLine("Price:\t\t" + myCar.Price);
        Console.WriteLine("-------------------------");
        Console.WriteLine();

        Console.WriteLine("Current Speed: " + myCar.Speed);
        Console.WriteLine();
        Console.WriteLine("Accelerating...");
        myCar.Accelerate(30);
        Console.WriteLine("Current Speed: " + myCar.Speed);
        myCar.Accelerate(30);
        Console.WriteLine("Current Speed: " + myCar.Speed);
        myCar.Accelerate(50);
        Console.WriteLine("Current Speed: " + myCar.Speed);
        myCar.Accelerate(290);
        Console.WriteLine("Current Speed: " + myCar.Speed);


        Console.WriteLine();
        Console.WriteLine("Brake");
        myCar.Brake();
        Console.WriteLine("Current Speed: " + myCar.Speed);

        // these are read-only properties
        // myCar.Speed = 50; // Impossible!
        // myCar.SeatingCapacity = 5; // Impossible!
        // ...
    }
}
```

همان‌طور که می‌بینید یک سری field در کلاس Car موجود است که private هستند و دسترسی به آن‌ها فقط از طریق property امکان‌پذیر است. اکثر این property ها فقط شامل get accessor (یا getter) هستند و این بدین معنی است که تنها مجاز هستید مقدار آن‌ها را بخوانید و نمی‌توانید به آن‌ها مقداری را اختصاص بدهید.

Auto-Implemented Properties

با آمدن C# 3.0 این امکان به‌وجود آمد که بتوان property های خیلی ساده را تعریف کرد که دیگر نیازی به متغیر ندارند تا property روی آن‌ها مدیریت داشته باشد.

در عوض شما به کامپایلر اجازه می‌دهید که یک متغیر (underlying variable) برای این مورد به‌وجود آورد.

فرم کلی auto-implemented property به‌شکل زیر است:

```
type name { get; set; }
```

در این‌جا، type مشخص‌کننده‌ی نوع و name مشخص‌کننده‌ی نام property است. توجه کنید که get و set بدنه ندارند و مستقیماً بعد از آن‌ها semicolon قرار می‌گیرد. این syntax به کامپایلر می‌فهماند که باید یک storage location (که گاهاً به آن backing field هم گفته می‌شود) برای نگه‌داری مقدار مورد نظر بسازد. این متغیر (backing field) دارای اسم نبوده و مستقیماً برای شما قابل دسترس نیست و تنها می‌توانید از طریق property به آن دسترسی داشته باشید.

به مثال زیر توجه کنید:

```
using System;
class Person
{
    public string Name { get; set; }
    public string Family { get; set; }
    public int Age { get; set; }
    public string Gender { get; set; }

    public Person(string name, string family)
    {
        Name = name;
        Family = family;
    }
}
class PropDemo
{
    static void Main()
    {
        Person a = new Person("Ian", "Somerhalder");
        Console.WriteLine("Name: " + a.Name);
        Console.WriteLine("Family: " + a.Family);

        a.Age = 26;
        a.Gender = "Male";

        Console.WriteLine("Age: " + a.Age);
        Console.WriteLine("Gender: " + a.Gender);
    }
}
```

همان‌طور که می‌بینید، به‌جای تعریف متغیر مستقیماً property تعریف کرده‌ایم. از آن‌جا که property های تعریف شده public بوده و دارای getter و setter هستند، می‌توانید مقادیر را get و set کنید. بر خلاف property های معمولی، auto-implemented properties نمی‌توانند read-only یا write-only باشند و همیشه get و set باید تعریف شوند. با این‌که auto-implemented properties روش جالب و راحتی است، تنها زمانی باید از آن استفاده کنید که نیازی به کنترل کردن backing field نداشته باشید.

به‌طور پیش‌فرض، دسترسی به get و set بر اساس دسترسی خود properties (یا indexer) است. به‌عنوان مثال اگر property را به‌صورت public تعریف کنید، get و set نیز public هستند. با این حال می‌توانید برای get و set دسترسی جداگانه (مثلاً private) در نظر بگیرید.

به مثال زیر توجه کنید:

```
using System;
class Properties
{
    public int ID { get; private set; }
    public Properties()
    {
        ID = 180;
    }
}
class PropDemo
{
    static void Main()
    {
        Properties ob = new Properties();
        Console.WriteLine(ob.ID);

        // ob.ID = 550; // Illegal!
    }
}
```

در این مثال، ID در کلاس خودش هم می‌تواند get و هم می‌تواند set شود اما خارج از کلاس فقط قابل get شدن است. همان‌طور که ذکر شد auto-implemented property نمی‌تواند read-only یا write-only باشد (نمی‌تواند فقط get یا set داشته باشد) اما با در نظر گرفتن get یا set به‌صورت private می‌توانید دسترسی را محدود کنید.

نکته‌ی دیگر این است که می‌توانید از properties در object initializers نیز استفاده کنید.

به مثال زیر توجه کنید:

```
using System;
class Properties
{
    public string Name { get; set; }
    public int ID { get; set; }
}
class PropDemo
{
    static void Main()
    {
        Properties ob = new Properties() { Name = "Joe", ID = 180 };
        Console.WriteLine(ob.Name + ", " + ob.ID);
    }
}
```

همان‌طور که می‌بینید، Name و ID توسط object initializer مقداردهی شده‌اند. همان‌طور که قبلاً ذکر شد، از object initializers بیشتر در LINQ استفاده می‌شود.

Properties نیز تعدادی محدودیت دارند. یک، از آن‌جا که property ها storage location تعریف نمی‌کنند نمی‌توانند به‌عنوان پارامتر ref و out به متد فرستاده شوند. دو، propery ها overload نمی‌شوند. محدودیت آخر این است که property در هنگام استفاده از get نباید backing field را تغییر دهد. هرچند کامپایلر این اجبار را به‌وجود نمی‌آورد، با این حال تغییر دادن backing field با استفاده از get از لحاظ منطقی صحیح نمی‌باشد.

در مثال زیر، برنامه‌ی دفترچه تلفن ساده‌ای را می‌بینید که از properties استفاده کرده است:

```
using System;
class Person
{
    public string Name { get; set; }
    public int Number { get; set; }
    public string Email { get; set; }

    public Person(string name, int number)
    {
        Name = name;
        Number = number;
        Email = "";
    }
    public Person(string name, int number, string email)
        : this(name, number)
    {
        Email = email;
    }
}
class PhoneBook
{
    public int Counter { get; private set; }
    Person[] persons;
    public PhoneBook()
    {
        persons = new Person[10];
        Counter = 0;
    }

    public bool AddContact(Person p)
    {
        if (Counter < persons.Length)
        {
            persons[Counter] = p;
            Counter++;
            return true;
        }
        return false;
    }
    public Person GetContact(int index)
    {
```

```
        return persons[index];
    }
}
class UI
{
    PhoneBook phoneBook;
    public UI()
    {
        phoneBook = new PhoneBook();
    }
    public string ShowMenu()
    {
        Console.Clear();
        Console.WriteLine("Simple PhoneBook");
        Console.WriteLine("----------------");
        Console.WriteLine("1. Add");
        Console.WriteLine("2. Show Contacts");
        Console.WriteLine("3. Exit");
        Console.WriteLine();
        Console.Write("Choose a number: ");
        return Console.ReadLine();
    }
    public void Process(string choice)
    {
        switch (choice)
        {
            case "1":
                Console.Clear();
                Person person = new Person(
                    GetInput("Enter Name: "),
                    Convert.ToInt32(GetInput("Enter Number: ")),
                    GetInput("Enter Email Address: ")
                    );

                if (phoneBook.AddContact(person))
                    Console.WriteLine("The contact has been added successfully.");
                else
                    Console.WriteLine("Faild! Something's wrong...");
                break;
            case "2":
                Console.Clear();
                for (int i = 0; i < phoneBook.Counter; i++)
                {
                    Console.WriteLine("Name: {0}", phoneBook.GetContact(i).Name);
                    Console.WriteLine("Number: {0}", phoneBook.GetContact(i).Number);
                    Console.WriteLine("Email: {0}", phoneBook.GetContact(i).Email);
                    Console.WriteLine();
                }
                break;
            case "3":
                Environment.Exit(0);
                break;
            default:
                Console.WriteLine("Invalid Choice!");
                break;
        }
    }
    public string GetInput(string message)
    {
        Console.Write(message);
        return Console.ReadLine();
```

```
    }
}
class PhoneBookDemo
{
    static void Main()
    {
        UI ui = new UI();
        while (true)
        {
            ui.Process(ui.ShowMenu());
            Console.ReadLine();
        }
    }
}
```

در مثال بالا، به‌جای استفاده از field از auto-implemented property استفاده شده است. هرچند می‌توانستیم از property های معمولی نیز استفاده کنیم ولی به‌دلیل اینکه در این‌جا نیازی به کنترل روی backing field نیست، استفاده از auto-implemented property مناسب‌تر است.