زنگ سی‌شارپ – قسمت بیست و چهارم

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

کنترل دسترسی به اعضای کلاس

Encapsulation در سی‌شارپ بدین معناست که اطلاعات یک کلاس در برابر دسترسی‌های غیرمجاز و خراب‌کاری محفوظ نگه داشته شود. کلاس که امکان encapsulation را به شما می‌دهد، دو مزیت عمده دیگر نیز به همراه دارد. اول اینکه داده‌ها را به کدهای درون کلاس متصل می‌کند. دوم اینکه دسترسی به اعضای کلاس را کنترل می‌کند. تا اینجا شما با دو نوع از اعضای کلاس آشنا شده‌اید که یکی public بود و دیگری private.

عضوی که public است می‌تواند آزادانه در خارج از کلاس خودش نیز قابل دسترسی باشد. اعضای private فقط درون همان کلاس قابل دسترسی هستند و البته از طریق یک متد public می‌توانند قابل دسترسی و کنترل باشند.

محدود کردن دسترسی به اعضای یک کلاس یکی از بخش‌های اساسی برنامه‌نویسی شی‌گرا است چراکه از استفاده‌ی نادرست یک شیء جلوگیری می‌کند. کنترل دسترسی از طریق access modifier ها انجام می‌شود که public، private، protected و internal هستند. فعلاً به public و private می‌پردازیم و protected و internal در جای خود توضیح داده می‌شوند. استفاده به‌موقع و به‌جا از public و private یکی از نکات مهم و کلیدی برنامه‌نویسی شی‌گرا است. در زیر به چند نکته در این مورد اشاره شده:

- آن دسته از اعضای کلاس که فقط در همان کلاس استفاده می‌شوند باید private باشند.
- داده‌هایی که باید بین یک محدوده خاص باشند نیز باید private باشند و دسترسی به آن‌ها از طریق یک متد public که بازه‌ی اعداد را بررسی می‌کند، کنترل شود.
- اگر تغییر دادن یکی از اعضا موجب شود که تاثیر این تغییر از آن عضو فراتر رود (و بر قسمت‌های دیگر شیء تاثیر گذارد)، آن عضو باید private باشد و دسترسی به آن باید کنترل شود.
- آن دسته از اعضا که درصورت استفاده نادرست موجب می‌شوند که شیء آسیب ببیند و آن‌طور که باید رفتار نکند، باید private باشند و دسترسی به آن‌ها از طریق متدهای public انجام شود تا از استفاده‌ی نادرست آن‌ها جلوگیری شود.

- متدهایی که مقداری را به اعضای private اختصاص می‌دهند یا مقدار آن‌ها را می‌خوانند، باید public باشند.
- اگر هیچ دلیلی برای private کردن متغیرهای کلاس نیافتید، می‌توانید آن‌ها را public در نظر بگیرید.

البته نکات ظریف دیگری نیز وجود دارندکه در بالا به آن‌ها شاره نکردیم اما در کل اگر همین قوانین را رعایت کنید اشیایی که می‌سازید به‌راحتی خراب نمی‌شوند و مورد سوءاستفاده قرار نمی‌گیرند.

برای این‌که درک بهتری از این موارد داشته باشید و بدانید که چرا و چگونه از این‌ها استفاده می‌کنیم، مثالی از ساختمان داده‌ی stack می‌تواند مفید باشد. یکی از مثال‌های مفید برنامه‌نویسی شی‌گرا کلاسی است که stack را پیاده‌سازی و اجرا می‌کند. همان‌طور که احتمالاً می‌دانید، stack ساختاری است برای ذخیره کردن اطلاعات که در آن اولین مقدار ذخیره شده، آخرین مقداری است که مورد استفاده قرار می‌گیرد. این‌طور تصور کنید که تعدادی ظرف را روی میزی (روی هم) قرار داده‌اید. اولین ظرفی که روی میز قرار دادید، آخرین ظرفی است که می‌توانید آن را بردارید یا برعکس، آخرین ظرفی که گذاشته‌اید، اولین ظرفی است که می‌توانید آن را بردارید. به این مکانیسم در اصطلاح Last-in, First-out می‌گویند. کلاسی که ما برای این منظور تعریف می‌کنیم شامل محلی برای ذخیره‌سازی اطلاعات و متدهایی برای کنترل روی این اطلاعات است بنابراین stack درواقع یک data engine است که این اجبار را به‌وجود می‌آورد تا عملیات Last-in, First-out اجرا شود. با این تعاریف نیاز است که یک‌سری از اعضای کلاس private و یک‌سری دیگر public باشند تا بتوان مکانیسم Last-in, First-out را اجرا کرد.

یک stack دو عملکرد پایه‌ای را انجام می‌دهد که عمل push و pop هستند. Push یک مقدار را به بالای stack اضافه می‌کند و pop یک مقدار را از بالای stack حذف می‌کند. کلاسی که تعریف خواهیم کرد، شامل یک آرایه‌ی private برای ذخیره سازی اطلاعات و دو متد public که عملیات pop و push را انجام می‌دهند و Last-in, First-out را پیاده می‌کنند:

```
// A stack class for characters
class Stack
{
    // These members are private
    char[] stck; // holds the stack
    int tos; // index of the top of the stack

    // Construct an empty Stack given its size.
    public Stack(int size)
    {
        stck = new char[size]; // allocate memory for stack
        tos = 0;
    }
```

```
    // Push characters onto the stack
    public void Push(char ch)
    {
        if (tos == stck.Length)
        {
            Console.WriteLine("Stack is full!");
            return;
        }
        stck[tos] = ch;
        tos++;
    }

    // Pop a character from the stack.
    public char Pop()
    {
        if (tos == 0)
        {
            Console.WriteLine("Stack is empty!");
            return (char)0;
        }
        tos--;
        return stck[tos];
    }

    // Return true if the stack is full.
    public bool IsFull()
    {
        return tos == stck.Length;
    }

    // Return true if the stack is empty.
    public bool IsEmpty()
    {
        return tos == 0;
    }

    // Return total capacity of the stack.
    public int Capacity() { return stck.Length; }

    // Return number of objects currently on the stack.
    public int GetNum() { return tos; }
}
```

بهتر است به این کلاس دقیق‌تر نگاه کنیم. کلاس Stack با تعریف این دو instance variables شروع می‌شود:

```
// These members are private
char[] stck; // holds the stack
int tos; // index of the top of the stack
```

آرایه‌ی stck محلی برای ذخیره‌سازی اطلاعات در stack فراهم می‌کند که این آرایه کاراکتر را در خود نگه می‌دارد. تخصیص آرایه توسط constructor انجام می‌گیرد و متغیر tos ایندکس (شماره) بالاترین خانه‌ی stack را در خود نگه می‌دارد.

هر دو عضو tos و stck به‌صورت private هستند و این باعث می‌شود که مکانیسم Last-in, First-out اجرا شود. اگر دسترسی این دو به‌صورت public تعریف شود آن‌گاه عناصر stack می‌توانند خارج از ترتیب قابل دسترسی باشند. همچنین از آن‌جا که متغیر tos ایندکس بالاترین عنصر stack را نگهداری می‌کند باید دسترسی به آن محدود شود تا توسط کدهای خارج از کلاس در دسترس و قابل دستکاری و خراب‌کاری نباشد. البته دسترسی به tos و stck از طریق متدهای public امکان‌پذیر است.

در خط بعد constructor را می‌بینید:

```
// Construct an empty Stack given its size.
public Stack(int size)
{
    stck = new char[size]; // allocate memory for stack
    tos = 0;
}
```

توسط constructor اندازه دل‌خواه stack مشخص شده و آرایه مورد نظر ساخته می‌شود. همچنین به متغیر tos مقدار صفر اختصاص می‌یابد. بنابراین مقدار صفر برای tos مشخص می‌کند که stack خالی است.

متد Push() که public تعریف شده، یک عنصر را به stack اضافه می‌کند:

```
// Push characters onto the stack
public void Push(char ch)
{
    if (tos == stck.Length)
    {
        Console.WriteLine("Stack is full!");
        return;
    }
    stck[tos] = ch;
    tos++;
}
```

عنصری که قرار است به stack افزوده شود از طریق پارامتر ch وارد stack می‌شود. البته قبل از این‌که عنصری به stack اضافه شود ابتدا بررسی می‌شود که stack هنوز خانه‌ی خالی داشته باشد. این‌کار با بررسی این‌که tos از طول stck فراتر نرفته باشد انجام می‌شود. سپس کاراکتر در ایندکسی که tos مشخص می‌کند ذخیره می‌شود و در نهایت tos یک واحد افزایش می‌یابد. بنابراین tos همیشه ایندکس خانه‌ی بعدی stck را در خود نگه می‌دارد.

برای پاک کردن یک عنصر از stack از متد Pop() استفاده می‌کنیم:

```
// Pop a character from the stack.
public char Pop()
{
    if (tos == 0)
    {
        Console.WriteLine("Stack is empty!");
        return (char)0;
```

```
    }
    tos--;
    return stck[tos];
}
```

در این‌جا مقدار tos بررسی می‌شود، اگر برابر با صفر بود stack خالی است در غیر این‌صورت tos یک واحد کاهش می‌یابد و عنصری که tos به آن اشاره می‌کند return می‌شود.

اگرچه Push() و Pop() تنها متدهایی هستند که برای اجرای یک stack مورد نیاز است اما تعریف چند متد دیگر نیز می‌تواند مفید باشد. ما در این کلاس ۴ متد بیشتر تعریف کرده‌ایم که IsFull()، IsEmpty()، Capacity() و GetNum() نام دارند و اطلاعاتی را از وضعیت stack به ما می‌دهند:

```
// Return true if the stack is full.
public bool IsFull()
{
    return tos == stck.Length;
}

// Return true if the stack is empty.
public bool IsEmpty()
{
    return tos == 0;
}

// Return total capacity of the stack.
public int Capacity() { return stck.Length; }

// Return number of objects currently on the stack.
public int GetNum() { return tos; }
```

متد IsFull() هنگامی که stack پر است true برمی‌گرداند و متد IsEmpty() هنگامی که stack خالی است true و هنگامی که stack خالی نیست false برمی‌گرداند. برای بدست آوردن ظرفیت کلی stack متد Capacity() فراخوانی می‌شود و برای به‌دست آوردن تعداد عناصری که در stack ذخیره شده‌اند نیز از متد GetNum() استفاده می‌شود. دلیل اهمیت این متدها این است که اطلاعات مورد نیاز برای دسترسی به tos را به ما می‌دهند (با این‌که private است). همچنین این متدها نمونه‌های خوبی هستند که بدانید چگونه از طریق متدهای public به اعضای private دسترسی و کنترل داشته باشید.

در مرحله‌ی بعد از این کلاس استفاده می‌کنیم:

```
using System;
class MyClass
{
    static void Main()
    {
        Stack stk1 = new Stack(10);
        Stack stk2 = new Stack(10);
        Stack stk3 = new Stack(10);
        char ch;
        int i;

        // Put some characters into stk1.
        Console.WriteLine("Push A through J onto stk1.");
        for (i = 0; !stk1.IsFull(); i++)
```

```
            stk1.Push((char)('A' + i));

        if (stk1.IsFull()) Console.WriteLine("stk1 is full.");

        // Display the contents of stk1.
        Console.Write("Contents of stk1: ");
        while (!stk1.IsEmpty())
        {
            ch = stk1.Pop();
            Console.Write(ch);
        }

        Console.WriteLine();

        if (stk1.IsEmpty()) Console.WriteLine("stk1 is empty.\n");

        // Put more characters into stk1.
        Console.WriteLine("Again push A through J onto stk1.");
        for (i = 0; !stk1.IsFull(); i++)
            stk1.Push((char)('A' + i));

        // Now, pop from stk1 and push the element in stk2.
        // This causes stk2 to hold the elements in reverse order.
        Console.WriteLine("Now, pop chars from stk1 and push " +
        "them onto stk2.");
        while (!stk1.IsEmpty())
        {
            ch = stk1.Pop();
            stk2.Push(ch);
        }

        Console.Write("Contents of stk2: ");
        while (!stk2.IsEmpty())
        {
            ch = stk2.Pop();
            Console.Write(ch);
        }
        Console.WriteLine("\n");

        // Put 5 characters into stack.
        Console.WriteLine("Put 5 characters on stk3.");
        for (i = 0; i < 5; i++)
            stk3.Push((char)('A' + i));

        Console.WriteLine("Capacity of stk3: " + stk3.Capacity());
        Console.WriteLine("Number of objects in stk3: " +
        stk3.GetNum());
    }
}
```

خروجی:

```
Push A through J onto stk1.
stk1 is full.
Contents of stk1: JIHGFEDCBA
stk1 is empty.
Again push A through J onto stk1.
Now, pop chars from stk1 and push them onto stk2.
Contents of stk2: ABCDEFGHIJ
Put 5 characters on stk3.
Capacity of stk3: 10
Number of objects in stk3: 5
```

در مثال بعد قصد داریم برنامه دفترچه تلفن سابق را بیشتر شی‌گرا کنیم. در این دفترچه تلفن یک کلاس برای مخاطب داریم و یک کلاس برای دفترچه تلفن. در کلاس مخاطب، متغیرهای نام، آدرس و شماره تلفن قرار دارد. در کلاس دفترچه تلفن، متدهایی برای جستجو، افزودن مخاطب جدید و نمایش مخاطبان ذخیره شده وجود دارد:

```
using System;
class MainClass
{
    static void Main()
    {
        Phonebook myPhonebook = new Phonebook(5);
        while (true)
        {
            Console.Clear();
            Console.WriteLine("1. Add");
            Console.WriteLine("2. Search By Name");
            Console.WriteLine("3. Show all");
            Console.WriteLine("4. Exit");
            Console.WriteLine();
            Console.Write("Choose a number: ");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    Console.Clear();
                    Person contact = new Person(
                        GetString("Please enter your name: "),
                        GetNum("Please enter your number: "),
                        GetString("Please enter your address: ")
                        );

                    if(myPhonebook.Add(contact))
                        Console.WriteLine("Your contact added successfully.");
                    else
                        Console.WriteLine("Fail!");

                    break;
                case "2":
                    myPhonebook.SearchByName(GetString("Please enter the name: "));
                    break;
                case "3":
                    Console.Clear();
                    myPhonebook.ShowContacts();
                    break;
                case "4":
                    Environment.Exit(0);
                    break;
                default:
                    Console.WriteLine("Invalid Input!");
                    break;
            }
            Console.ReadLine();
        }
    }
    static string GetString(string message)
    {
        Console.Write(message);
        return Console.ReadLine();
```

```
    }
    static int GetNum(string message)
    {
        Console.Write(message);
        return Convert.ToInt32(Console.ReadLine());
    }
}
class Person
{
    string Name;
    int Number;
    string Address;

    public Person(string name, int number, string address)
    {
        Name = name;
        Number = number;
        Address = address;
    }

    public string GetName() { return Name; }
    public int GetNumber() { return Number; }
    public string GetAddress() { return Address; }
}
class Phonebook
{
    Person[] Persons;
    int Current;
    bool Found;

    public Phonebook(int size)
    {
        Persons = new Person[size];
        Current = 0;
    }

    public bool Add(Person person)
    {
        if (Current < Persons.Length)
        {
            Persons[Current] = person;
            Current++;
            return true;
        }
        return false;
    }
    public void SearchByName(string search)
    {
        Found = false;
        for (int i = 0; i < Persons.Length; i++)
        {
            if (Persons[i] == null) break;
            if (search == Persons[i].GetName())
            {
                Found = true;
                Console.WriteLine();
                Console.WriteLine(Persons[i].GetName() + "\n" + Persons[i].GetNumber() +
                "\n" + Persons[i].GetAddress());
            }
        }
        if(!Found)
            Console.WriteLine("Not Found!");
```

```
    }
    public void ShowContacts()
    {
        for (int i = 0; i < Persons.Length; i++)
        {
            if (Persons[i] == null) return;
            Console.WriteLine(Persons[i].GetName() + "\n" + Persons[i].GetNumber() +
                "\n" + Persons[i].GetAddress() + "\n");
        }
    }
}
```

همان‌طور که می‌بینید نسبت به قبل از اشیاء و کلاس‌های بیشتری استفاده کردیم و این برنامه را بیشتر به سمت شی‌گرایی سوق دادیم. در کلاس Person سه instance variable تعریف کرده‌ایم که هر سه private و شامل نام، آدرس و شماره تلفن هستند. همچنین از constructor برای مقداردهی به این سه متغیر استفاده کرده و از طریق متدهای public این متغیرها را Read-only کردیم. در کلاس Phonebook آرایه‌ای از جنس کلاس Person داریم. با این‌کار در هر خانه‌ی آرایه، یک شیء از جنس Person ذخیره می‌شود که شامل نام، آدرس و شماره تلفن است. از طریق constructor این کلاس، آرایه ساخته و مقدار دهی می‌شود. کلاس Phonebook همچنین شامل سه متد برای جستجو، افزودن مخاطب جدید و نمایش مخاطبان ذخیره شده دارد. همان‌طور که می‌بینید، متد Add() یک شیء از جنس Person دریافت می‌کند و آن را در آرایه‌ای که از جنس Person ساخته بودیم ذخیره می‌کند و در نهایت اگر با موفقیت مخاطب جدید را ذخیره کرد، مقدار true را باز می‌گرداند. در متد Main() برای افزودن مخاطب جدید، یک شیء از کلاس Person ساخته و argument های لازم را به آن داده‌ایم. سپس متد Add() را صدا زده‌ایم تا مخاطب جدید ذخیره شود.