زنگ سی‌شارپ – قسمت سی‌ام

نوشته‌ی مسعود درویشیان

لینک مستقیم این مطلب در وب‌تارگت

در قسمت قبلی زنگ سی‌شارپ در حل تمرین شماره‌ی ۱۴ تا آنجا پیش رفتیم که توانستیم یک هنرمند را ذخیره و هم‌چنین توانستیم لیست هنرمندهای ذخیره شده را مشاهده کنیم. در این قسمت به ادامه‌ی حل تمرین شماره ۱۴ می‌پردازیم.

هنگامی که یک Artist را ذخیره می‌کنید علاوه‌بر ذخیره کردن آن، یک‌سری عملیات دیگر را باید روی آن انجام دهید. در این برنامه عملیات Edit، Delete، Add Album، Add Single Tune، View Single Tunes و View Albums برای هر خواننده مد نظر ما است. بدین معنی که پس از افزودن یک خواننده به جعبه‌ی موسیقی، بتوانیم به آن، آلبوم و تک‌آهنگ اضافه کنیم، آثار آن هنرمند را ویرایش کنیم، ببینیم، بشنویم و یا این‌که آن هنرمند را به‌کلی حذف کنیم.

اگر قصد داشته باشید هنرمندی که ذخیره کردید را حذف کنید، کافی است آن خانه از آرایه که هنرمند مربوطه در آن ذخیره شده است را برابر با null قرار دهید. برای این منظور ما یک متد به نام RemoveArtist() به کلاس MusicBox اضافه می‌کنیم:

```
public void RemoveArtist(int index)
{
    Artists[index] = null;
}
```

کلاس MusicBox آرایه‌ای از Artist دارد. در این متد ما index آرایه‌ای که هنرمند مربوطه در آن ذخیره شده است را مشخص کرده و سپس آن را برابر با null قرار می‌دهیم.

توجه کنید هنگامی‌که هنرمند ذخیره شد، ما باید یک سری عملیات را روی آن انجام دهیم، پس اگر ۴ هنرمند ذخیره کرده باشید باید از بین این ۴ نفر، هنرمند مربوطه انتخاب شود تا عملیات لازم را روی آن انجام دهیم. برای این‌کار باید یک جستجو در آرایه‌ی Artists انجام دهیم.

در مثال زیر نمونه‌ای از جستجو کردن در یک آرایه از جنس int را مشاهده می‌کنید:

```
using System;
class SearchDemo
{
    static void Main()
    {
        bool found = false;
        int[] myArray = { 2, 6, 4, 12, 20 };

        int input = Convert.ToInt32(GetInput("Enter a number: "));

        for (int i = 0; i < myArray.Length; i++)
        {
            if (input.Equals(myArray[i]))
            {
                found = true;
                Console.WriteLine("This number is exist.");
                break;
            }
        }

        if (!found)
            Console.WriteLine("This number doesn't exist!");
    }
    static string GetInput(string message)
    {
        Console.Write(message);
        return Console.ReadLine();
    }
}
```

همان‌طور که می‌بینید یک عدد از کاربر گرفته شده و توسط یک حلقه‌ی for، موجود بودن عدد در آرایه بررسی شده است. در این میان می‌بینید که به‌جای استفاده از == از متد Equals() به منظور مقایسه استفاده کردیم. این متد یک مقدار بولین را return می‌کند. در آرایه Artists جستجو را به همین منوال انجام می‌دهیم با این تفاوت که آرایه از جنس int نیست.

یکی دیگر از کارهایی که باید انجام دهیم افزودن آلبوم برای هر هنرمند است. هر آلبوم یک سری مشخصات و یک سری آهنگ دارد بنابراین برای ساخت یک آلبوم بایستی این اطلاعات را برای آلبوم فراهم کنیم. مجدداً به کلاس Album نگاهی بیندازید:

```
class Album
{
    // Fields
    private string AlbumName;
    private string AlbumOwner;
    private string AlbumGenre;
    private ushort AlbumYear;
    private Tune[] Tunes;

    // Constructor
    public Album(string albumName, Artist artist,
        string albumGenre, ushort albumYear, Tune[] tunes)
```

```
    {
        AlbumName = albumName;
        AlbumOwner = artist.GetArtistNameAndFamily();
        AlbumGenre = albumGenre;
        AlbumYear = albumYear;
        Tunes = tunes;
    }
}
```

به constructor این کلاس توجه کنید، می‌بینید که باید نام آلبوم، اسم خواننده، سبک آلبوم، سال و آرایه‌ای از آهنگ‌ها را هنگام ساخت شیء از این کلاس تعریف کنیم بنابراین تمام این اطلاعات را از کاربر دریافت کرده، آرایه‌ای از Tune به‌وجود می‌آوریم و یک شیء از کلاس Album را تولید می‌کنیم. هنگامی‌که این شیء به‌وجود آمد، توسط متد زیر آن را در آرایه‌ی Albums کلاس Artist ذخیره خواهیم کرد:

```
public bool AddAlbum(Album album)
{
    if (Counter < Albums.Length)
    {
        Albums[Counter] = album;
        Counter++;
        return true;
    }
    return false;
}
```

این متد یک شیء از جنس Album دریافت و آن را در آرایه Albums ذخیره می‌کند. متغیر Counter را در constructor کلاس Artist برابر با صفر قرار می‌دهیم و بعد افزودن هر آلبوم، یک واحد به این متغیر می‌افزاییم. دلیل قرار دادن دستور if در این متد این است که هنگام افزودن آلبوم جدید از index آرایه‌ی Albums خارج نشویم چراکه در غیر این‌صورت برنامه با خطا مواجه می‌شود. این متد هنگامی‌که با موفقیت آلبوم را به آرایه افزود مقدار true و در صورت عدم موفقیت مقدار false را return می‌کند.

برای پاک کردن یک آلبوم از متد زیر استفاده می‌کنیم:

```
public bool RemoveAlbum(int index)
{
    if (Albums[index] != null)
    {
        Albums[index] = null;
        return true;
    }
    return false;
}
```

این متد نیز قبل از پاک کردن آلبوم ابتدا بررسی می‌کند که index مشخص شده null نباشد، سپس آن را برابر با null قرار می‌دهد.

نکته‌ی قابل توجه دیگر در این برنامه، متد ToLower() است. این متد یک پارامتر string می‌گیرد و تمام حروف آن را تبدیل به حروف کوچک (lowercase) و سپس این string جدید را return می‌کند. متد ToUpper() بر خلاف ToLower() عمل می‌کنید. به نمونه‌ی زیر توجه کنید:

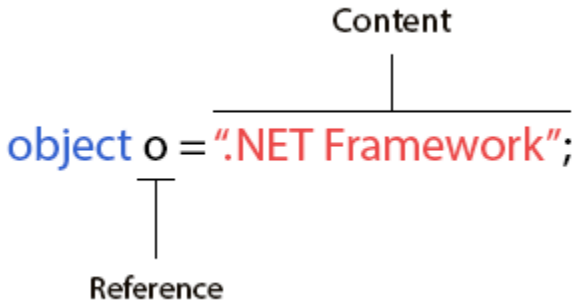
```
using System;
class ToUpperToLowerDemo
{
    static void Main()
    {
        string s1 = "FOR this OnE We'll use TOLOWER() METHOD.";
        string s2 = "for this one we'll use ToUpper() method.";

        Console.WriteLine(s1.ToLower());
        Console.WriteLine(s2.ToUpper());
    }
}
```

همان‌طور که ذکر شد ما از متد Equals() به‌جای == استفاده کردیم. مطمئناً سوالی که برای‌تان به‌وجود می‌آید این است که این‌ها چه تفاوتی با هم دارند؟ البته در این تمرین استفاده از هردو نتیجه‌ی یکسانی را در بر دارد اما دانستن تفاوت این دو خالی از لطف نیست. هنگامی‌که یک شیء می‌سازید، شیء شما شامل دو بخش است. یک بخش شامل محتوای شیء (content) و بخش دیگر شامل آدرسی (reference) است که به محتوا اشاره دارد.

برای مثال اگر شما به‌صورت زیر یک شیء بسازید:

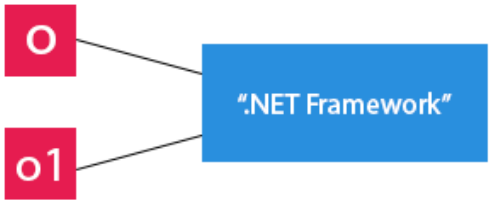
```
object o = ".NET Framework";
```

در این‌جا ".NET Framework" محتوا (content) است و o به محتوا اشاره دارد (reference).



تفاوت در این‌جاست که Reference == ها را باهم مقایسه می‌کند، در صورتی‌که متد Equals() محتوا (Content) را مورد مقایسه قرار می‌دهد. در مورد نوع object در سی‌شارپ همین‌قدر بدانید که همه‌ی کلاس‌ها از object مشتق می‌شوند. در مورد ارث‌بری بعداً مفصل صحبت خواهیم کرد.

بنابراین اگر شما کد زیر را اجرا کنید، == و متد Equals() هردو مقدار true را نمایش می‌دهند به این دلیل که Content و Reference ها یکی هستند:

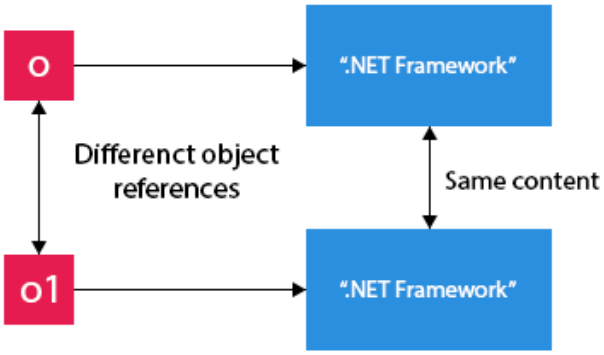



```
object o = ".NET Framework";
object o1 = o;
Console.WriteLine(o == o1);
Console.WriteLine(o.Equals(o1));
```

خروجی:

True
True

اکنون به کد زیر توجه کنید، در کد زیر می‌بینید که ما دو content یکسان داریم اما برای هرکدام reference جداگانه‌ای در نظر گرفته شده است. بنابراین اگر کد زیر را اجرا کنید == مقدار False و متد Equals() مقدار True را return می‌کند:



```
object o = ".NET Framework";
object o1 = new string(".NET Framework".ToCharArray());
Console.WriteLine(o == o1);
Console.WriteLine(o.Equals(o1));
```

خروجی:

False
True

در خط دوم این برنامه یک شیء string جدید توسط new و متد ToCharArray() ساخته شده است که محتوای آن با قبلی یکسان است بنابراین دو شیء و دو reference جداگانه داریم.

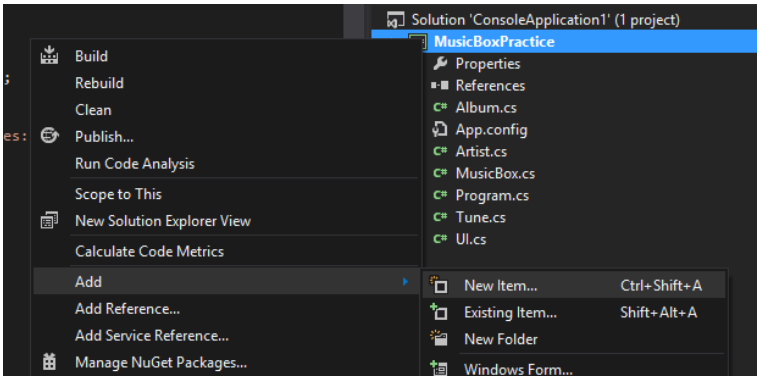
هنگامی که از string data type استفاده می‌کنید، content ها با هم مقایسه می‌شوند. به عبارت دیگر، چه از == استفاده کنید و چه از Equals()، مقایسه همیشه روی content صورت می‌گیرد.

به ادامه حل تمرین برمی‌گردیم. برای نمایش آلبوم‌های ذخیره شده از متد زیر (که در آلبوم Artist قرار دارد) استفاده می‌کنیم:

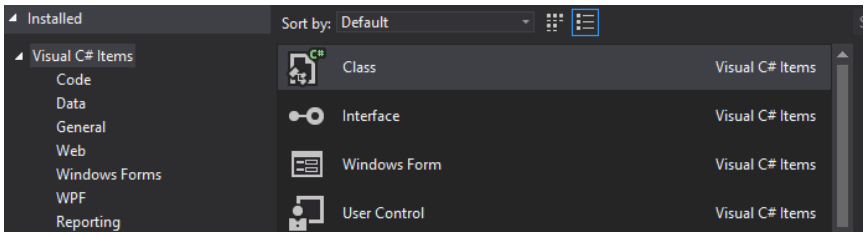
```
public void ViewAlbums()
{
    for (int i = 0; i < Albums.Length; i++)
    {
        if (Albums[i] == null) continue;
        Console.WriteLine(Albums[i].GetAlbumName());
    }
}
```

هنگامی که برنامه شما کلاس‌های زیادی دارد دیگر نباید کلاس‌ها را پشت سر هم ردیف و از آن‌ها استفاده کنید بلکه باید هر کلاس در فایل جداگانه‌ای قرار داشته باشد. برای ساختن فایل جدا برای هر کلاس به ترتیب زیر عمل می‌کنید.

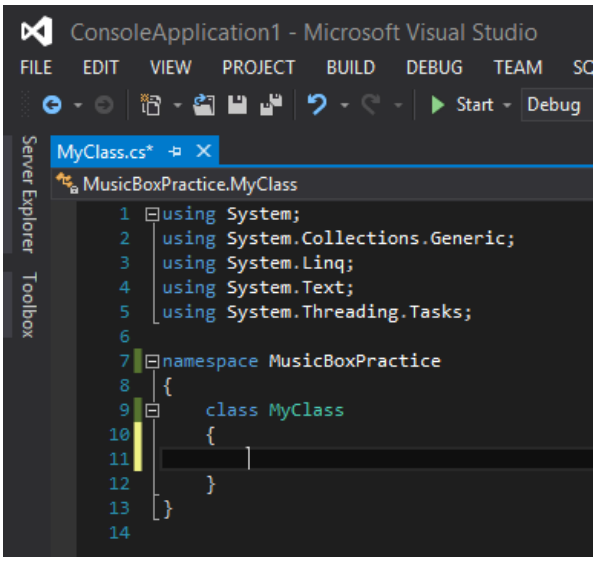
ابتدا در برنامه Visual Studio روی نام پروژه Right Click کرده سپس گزینه‌ی Add، بعد از آن New Item را انتخاب کنید:



سپس از قسمت Visual C# Items گزینه‌ی Class را انتخاب می‌کنید (یک نام دل‌خواه نیز برای آن در نظر بگیرید):



هنگامی‌که کلاس‌تان را اضافه کردید، کدهای زیر را درون آن می‌بینید:



در عکس بالا می‌بینید که یک‌سری using به‌طور پیش‌فرض در کلاسی که ساخته‌اید وجود دارد. می‌توانید همه را به‌جز using System پاک کنید یا اینکه اجازه بدهید در برنامه باشند. این‌ها به شما اجازه می‌دهند به کلاس‌ها و متدهای مختلف و گسترده‌تری از .NET Framework دسترسی داشته باشید. در این‌جا ما تنها به using System نیاز داریم. نکته‌ی قابل توجه در این‌جا namespace است. در واقع، هر برنامه سی‌شارپ از namespace استفاده می‌کند و ما تا این‌جا نیازی به توضیح namespace نداشتیم زیرا سی‌شارپ به‌طور اتوماتیک، یک namespace پیش‌فرض، یک global namespace برای برنامه شما تعریف می‌کند بنابراین برنامه‌هایی که تاکنون می‌نوشتیم از global namespace استفاده می‌کردند اما در برنامه‌های واقعی و کاربردی، برنامه‌نویسان namespace خودشان را تعریف می‌کنند و همچنین با namespace های دیگری نیز در تعامل هستند. یک namespace در واقع ناحیه‌ای را مشخص می‌کند که یک مجموعه‌ی اسم بتوانند از یک موجموعه‌ی اسم دیگر، جدا در نظر گرفته شوند. به عبارت دیگر، یک نام در یک namespace با نام یکسانی دیگر در namespace

دیگری تداخل پیدا نمی‌کند. یکی از namespace های استفاده شده در کتاب‌خانه‌ی .NET Framework (که به آن کتاب‌خانه‌ی سی‌شارپ (C# Library) هم گفته می‌شود) System است. به این دلیل است که ما از عبارت:

```
using System;
```

در بالای هر برنامه استفاده می‌کنیم. namespace های زیادی وجود دارند که وابسته به System هستند و بخش‌های دیگر کتاب‌خانه‌ی سی‌شارپ نگهداری می‌کنند. namespace ها از اهمیت بالایی برخوردار هستند زیرا نام‌های بسیار زیادی در برنامه می‌توانند وجود داشته باشند. نام متغیرها، کلاس‌ها و... که در کتاب‌خانه‌ی سی‌شارپ هستند، نام‌هایی که در فایل‌های dll هستند و وارد برنامه‌تان می‌کنید (در مقالات آینده با dll آشنا خواهید شد) و همچنین اسم‌هایی که در برنامه خودتان تعریف می‌کنید، همه در صورت عدم وجود namespace با هم تداخل پیدا می‌کنند. برای مثال اگر در برنامه‌تان یک کلاس به اسم Accelerate تعریف کنید و در کتاب‌خانه‌هایی که استفاده می‌کنید کلاسی با این نام وجود داشته باشد، بین این دو تداخل و ناسازگاری به‌وجود خواهد آمد. با استفاده از namespace دیگر چنین مشکلاتی به‌وجود نخواهد آمد.

پروژه حل تمرین شماره ۱۴ را می‌توانید از اینجا دانلود کنید. پروژه‌ای که اکنون دانلود می‌کنید شامل مواردی است که تاکنون در مورد این تمرین روی آن‌ها بحث کردیم. این پروژه را به‌صورت کامل در قسمت‌های بعدی (بعد از این‌که روی تمام قسمت‌های حل آن بحث شد) می‌توانید دانلود کنید.