دانشکده مهندسی کامپیوتر

جزوه درس

برنامه‌سازی پیشرفته

استاد درس: سید صالح اعتمادی*

نیم‌سال دوم

سال تحصیلی ۹۸-۹۹

* مطالب این جزوه توسط دانشجویان جمع‌آوری شده است. استاد درس درستی مطالب را بررسی نکرده است.

فهرست مطالب

۲ Functions and Parameters ۸

پریا فصاحت - ۱۳۹۸/۱۱/۱۹

۱.۲ معرفی زبان‌های Managed و Native ۸

۳ Functions ۱۳

نیوشا یقینی - ۱۳۹۸/۱۱/۳۰

۱.۳ تعریف Function ۱۴

۲.۳ انواع Function ۱۵

۳.۳ انواع ورودی ها ۱۵

۴.۳ انواع خروجی ها ۱۷

۴ Method/Function Features ۱۹

محمدمهدی جاوید - ۱۳۹۸/۱۱/۲۶

۱.۴ متد اورلودینگ ۱۹

۲.۴ فرستادن براساس مقدار یا آدرس

Pass by reference VS Pass by value ۲۷

۳.۴ توابع با پارامترهای ورودی متغیر (variadic functions) : ۳۴

۴.۴ توابع و کلاس‌های عمومی (Generic Functions) : ۳۹

۵.۴ قراردادهای نام گذاری (Naming Conventions) : ۴۴

۶.۴ Fold Expression : ۴۹

۷.۴ نحوه صحیح تقسیم کردن دو عدد بر یکدیگر : ۵۳

۸.۴ تمرین کلاسی : ۵۶

۵ Class and Object ۶۰
روزبه غزوی - ۱۳۹۸/۱۱/۲۸

۱.۵ تعریف کلاس (class) . . . ۶۱
۲.۵ ایجاد یک شیء از کلاس . . . ۶۱
۳.۵ بررسی یک نمونه مثال از کلاس . . . ۶۲
۴.۵ سطح دسترسی (Access Modifiers) . . . ۶۳
۵.۵ فیلد (Field) . . . ۶۴
۶.۵ سازنده (Constructor) . . . ۶۴
۷.۵ متد (Method) . . . ۶۵
۸.۵ صفت (Property) . . . ۶۵
۹.۵ صفات Auto-implemented . . . ۶۶
۱۰.۵ فضای نام (Namespace) . . . ۶۷

۶ آرایه‌ها و کلاس ۶۸
نیکی نزاکتی - ۱۳۹۸/۱۲/۳

۱.۶ آرایه‌ها در C++ . . . ۶۸
۲.۶ آرایه‌ها در Java . . . ۷۰
۳.۶ آرایه‌ها در Python . . . ۷۱
۴.۶ آرایه‌ها در C# . . . ۷۲
۵.۶ کلاس در C++ . . . ۷۳

۷ تفاوت بین رفرنس تایپ با ولیو تایپ در سی پلاس پلاس ،جاوا و سی شارپ ۷۶
امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

۱.۷ Value Type و Reference Type . . . ۷۷
۲.۷ Value Type . . . ۷۷
۳.۷ Pass by Value ارسال با مقدار . . . ۷۸
۴.۷ Reference Type . . . ۷۹
۵.۷ Pass by Reference ارسال با ارجاع . . . ۸۰
۶.۷ Heap و Stack . . . ۸۱
۷.۷ Java in Class . . . ۸۵

۸ کار با فایل در سی‌شارپ ۸۶

بابک بهکام کیا - ۱۳۹۸/۱۲/۱۰

۱.۸ آشنایی با فایل ۸۶
۲.۸ برنامه ثبت‌نام دانش‌آموزان ۸۷

۹ شی گرایی و استاتیک ۹۲
محمدجواد مهدی‌تبار - ۱۳۹۸/۱۲/۱۲

۱.۹ اهداف اصلی این جلسه ۹۲
۲.۹ کلمه های کلیدی مهم ۹۲
۳.۹ دستورات مهم فایل ۹۹
۴.۹ کد زده شده درون کلاس ۱۰۰

۱۰ (Directory/File) دیرکتوری و فایل ۱۰۵
علی رهنما علمداری - ۱۳۹۸/۱۲/۱۷

۱.۱۰ Directoryclass ۱۰۵
۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص ۱۰۷
۳.۱۰ منابع ۱۱۰

۱۱ شبیه سازی چیلین وارز ۱۱۱
شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹

۱.۱۱ حل مسئله ۱۱۱
۲.۱۱ Queue ۱۲۳

۱۲ stacks- objects ۱۲۷
بنفشه قلی نژاد - ۱۳۹۸/۱۲/۲۴

۱.۱۲ استک ۱۲۹
۲.۱۲ objects ۱۳۴

۱۳ Destructor ۱۴۱
یاسمین مدنی - ۱۳۹۹/۱/۱۶

۱.۱۳ عناوین کلی جلسه ۱۴۱
۲.۱۳ دیستراکتور و فاینالایزر ۱۴۲
۳.۱۳ Struct ۱۴۵

۱۴ داده نوع های Value Type و Reference Type ۱۴۷
مسعود گلستانه - ۱۳۹۹/۱/۱۸

۱.۱۴ تفاوت میان داده های reference type و value type در سی شارپ ۱۴۸
۲.۱۴ کپی سطحی (shallow copy) ۱۵۱
۳.۱۴ باکسینگ و آنباکسینگ ۱۵۲
۴.۱۴ Nullable ها در سی شارپ ۱۵۴

۱۵ Exceptions ۱۵۶
یاسمن توکلی - ۱۳۹۹/۱/۲۳

۱.۱۵ exception چیست؟ ۱۵۶
۲.۱۵ رفع exception ۱۵۷
۳.۱۵ نکته ۱ ۱۵۹
۴.۱۵ نکته ۲ ۱۶۱
۵.۱۵ try/catch vs if/else ۱۶۱
۶.۱۵ throwing using else/if ۱۶۲
۷.۱۵ Call Stack and Re-Throwing Exceptions ۱۶۳
۸.۱۵ تفاوت throw با throw new exception ۱۶۴

۱۶ Exceptions & Operators ۱۶۶
بیان دیوانی آذر - ۱۳۹۹/۱/۲۵

۱.۱۶ Exceptions ۱۶۶
۲.۱۶ Indexers ۱۷۳

۱۷ Operator Overloading ۱۷۵
نیکی مجیدی فرد - ۱۳۹۸/۱/۳۰

۱.۱۷ Operator Conversion ۱۷۷
۲.۱۷ Pairs Operator ۱۸۰

۱۸ واسط ها ۱۸۴
باوان دیوانی آذر - ۱۳۹۹/۲/۱

۱.۱۸ چالش ۱ ۱۸۴
۲.۱۸ چالش ۲ ۱۸۹
۳.۱۸ نکات ۱۹۲

۴.۱۸ خلاصه بندی ۱۹۲
۵.۱۸ تمرینات اضافی ۱۹۳

۱۹ Interface IEnumerable، IDisposable ۱۹۴
آزاده دارابی مقدم - ۱۳۹۹/۲/۶

۱.۱۹ Generic Interface ۱۹۴
۲.۱۹ Generic Constraints ۱۹۶
۳.۱۹ IDisposable ۱۹۷
۴.۱۹ StreamReader Class ۱۹۸
۵.۱۹ Stopwatch ۱۹۸
۶.۱۹ IEnumerable ۱۹۹

۲۰ چگونگی کارکرد مموری ۲۰۲
سعید شهیب زاده - ۱۳۹۹/۲/۸

۱.۲۰ Memorymanager Example ۲۰۲

۲۱ اشاره گر به تابع ۲۱۱
مهدیه نادرری - ۱۳۹۸/۲/۱۳

۱.۲۱ اشاره گر به تابع در زبان پایتون ۲۱۱
۲.۲۱ اشاره گر به تابع در زبان سی شارپ ۲۱۲

۲۳ Closure - Async Pattern ۲۲۱
پارمیدا مجمع صنایع - ۱۳۹۹/۰۲/۲۰

۱.۲۳ Closure در c++ ۲۲۲
۲.۲۳ Closure در c# ۲۲۷
۳.۲۳ Multi-Threading ۲۲۹

۲۴ Thread و Tasks و Async Pattern ۲۳۷
زهرا مومنی نژاد - ۱۳۹۹/۷/۱۸

۱.۲۴ Await و Async ۲۳۷
۲.۲۴ Tasks ۲۴۱
۳.۲۴ Thread ۲۴۲

۲۵ LINQ : Language Integrated Query ۲۵۲
فاطمه میرجلیلی - ۱۳۹۹/۲/۲۷

۱.۲۵ انواع تعریف `Tuple` . . . ۲۵۵
۲.۲۵ `Deconstruct` کردن یک `Tuple` . . . ۲۵۷
۳.۲۵ `LINQ` . . . ۲۵۷

۲۶ لینک ۲۶۲
محمد حسین رجبی - ۱۳۹۹/۳/۱۶

۱.۲۶ دیزاین پترن (الگوی طراحی) چیست ؟ . . . ۲۶۲
۲.۲۶ شکل فایل بدین صورت است : . . . ۲۶۳
۳.۲۶ Ternary operator . . . ۲۶۵
۴.۲۶ Aggregate . . . ۲۶۵
۵.۲۶ GroupBy . . . ۲۶۵
۶.۲۶ شکل فایل بدین صورت است : . . . ۲۶۶
۷.۲۶ Join . . . ۲۶۷

۲۷ وراثت ۲۶۸
امیرحسین درخشان - ۱۳۹۹/۳/۳

۱.۲۷ کاربرد وراثت . . . ۲۶۹
۲.۲۷ ضرورت استفاده از وراثت . . . ۲۷۰
۳.۲۷ توضیح کلی و نحوه استفاده از وراثت . . . ۲۷۰
۴.۲۷ استفاده از Constructor و توابع درکلاس های فرزند . . . ۲۷۲
۵.۲۷ استفاده از protected . . . ۲۷۴
۶.۲۷ استفاده از abstract . . . ۲۷۴
۷.۲۷ Polymorphism . . . ۲۷۵
۸.۲۷ استفاده از Seald . . . ۲۷۷

۲۹ Patterns Design ۲۷۸
زهرا امیری - ۱۳۹۹/۰۵/۱۴

۱.۲۹ Design Patterns . . . ۲۷۹

۳۰ Design Patterns - State Pattern ۲۹۰

۱.۳۰ کلاس Account ۲۹۰
۲.۳۰ Guard Clause ۲۹۱
۳.۳۰ State Pattern ۲۹۲
۴.۳۰ ماشین حساب ۲۹۹

جلسه ۲

Functions and Parameters

پریا فصاحت - ۱۳۹۸/۱۱/۱۹

جزوه جلسه ۲ام مورخ ۱۳۹۸/۱۱/۱۹درس برنامه‌سازی پیشرفته تهیه شده توسط پریا فصاحت؛ در این جلسه توابع و پارامترها در زبان‌های C++, C# تدریس شدند.امید است جزوه‌ی تهیه شده مفید واقع شود.

۱.۲ معرفی زبان‌های Managed و Native

زبان‌های برنامه‌نویسی در انواع مختلفی کامپایل می‌شوند؛که به تعریف و بررسی آن‌ها می‌پردازیم.

۱-اولین تفاوت این زبان‌ها به شرح زیر است:

- در بعضی زبان‌هایی مانند `C++, C` که `native` نامیده می‌شوند:

کد نوشته‌شده تنها کدی‌ست که اجرا می‌شود.از قسمت `main` شروع می‌شود و تنها هر چه در main نوشته شده اجرا می‌شود.وهیچ چیز اضافه‌ای اجرا نمی‌شود. کد فقط برای `CPU` همان ماشین کامپایل می‌شود. در واقع برای یک نرم‌افزار و یک سخت‌افزار منحصر به فرد.

- و اما در بعضی زبان‌ها مانند `Java, C#` که `managed` نامیده می‌شود:

این زبان‌ها تبدیل به یک زبان میانی می‌شوند. برای مثال در زبان C# تبدیل به `ILCode` * می‌شود. و در زبان Java ؛ `Java Byte Code` کد میانی‌ست. لازم به ذکر است که کدهای میانی به تنهایی قابل اجرا نیستند؛ و نیاز به یک `Run Time Enviroment` نیاز دارند.که برای جاوا `Java Run Time Enviroment` و برای سی‌شارپ `DotNet Run Time Enviroment` نام دارد. شایان ذکر است؛در این زبان‌ها می‌توان کد `Cross Platform` نوشت.

۲- دومین تفاوت زبان‌های native, managed در ویژگی تحت عنوان `Garbage Collection` است:

- به طور کلی در زبان‌های managed نیازی به از دسترس خارج کردن حافظه مصرف شده نیست.چرا که به علت دارا بودن ویژگی مدیریت حافظه † ؛نیازی به این کار نیست.

برای مثال در زبان‌های C#, Java خود کامپایلر وقتی ببیند که Pointer متغیر new یا malloc* شده؛در جایی استفاده نمی‌شود به طور خودکار حافظه‌ی تخصیص یافته را دوباره استفاده می‌کند.

*برای تفهیم تفاوت نوشتاری در استفاده از malloc یا new ؛به دو خط زیر در زبان CPP توجه کنید.

```cpp
int* nums1 = (int*) malloc(SizeOf(int) * 5);
 //Malloc

int* nums2 = new int[5];
 //new
```

نمونه کد ۱: تفاوت نوشتاری malloc و new

لازم به ذکر است؛قسمتی از حافظه‌ی heap به هر دو تخصیص داده می‌شود.

- در این قسمت با ایجاد متغیر مناسب برای داشتن کدی بهینه آشنا می‌شویم:

```cpp
public class Program
{
    int[] nums = ReadFromInput();
    static void Main(string[] args)
    {
        SortNumbers();
    }
}
```

نمونه کد ۲: استفاده از متغیر global

intermediate Language*
Garbage Collection†

در این حالت کد نوشته‌شده قابل اجرا فقط برای همین آرایه است؛ وبرای هر ارایه دیگری باید دوباره نوشته‌شود. بنابراین بهتر است به شکل دیگری بازنویسی شود:

```
public class Program
{
     static void Main(string[] args)
     {
         int[] nums = ReadFromInput();
         SortNumbers(nums);
      }
{
```

نمونه کد ۳: استفاده از متغیر local

این نمونه کد برای هرتعدادی آرایه قابل اجراست و نیازی به بازنویسی آن نیست.

*لازم به ذکر است در زبان C# به تابع static موجود در یک کلاس public؛ تابع Global می‌گویند.

- در ادامه توابع Swap، Sort را در زبان‌های C#, CPP پیاده‌سازی خواهیم کرد.

برای مرتب سازی آرایه از دو حلقه‌ی for استفاده کردیم:

```
static void Sort(int[] nums)
{
   for(int i=0; i<nums.Length;i++)
   {
        for(int j=i+1; j<nums.Length; j++)
        {
           if (nums[i] < nums[j])
           {
                Swap(ref nums[i], ref nums[j]);
           }
        }
    }
}
```

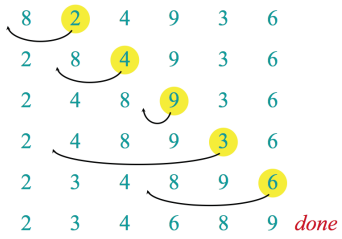
نمونه کد ۴: تابع Sort در زبان C#

در نمونه کد بالا کلیدواژه‌ای تحت عنوان `ref` مشاهده می‌کنیم.با استفاده از این کلید واژه در واقع ما آدرس این متغیر را در دسترس تابع قرار می‌دهیم و تغییر اعمال شده مستقیما روی خود متغیر ایجاد می‌شود، نه صرفا مقدار کپی شده‌ی آن. **[ref]**

- برای درک بهتر،این روش مرتب‌سازی را به وسیله‌ی شکل زیر توضیح می‌دهیم:

دز این روش با استفاده از دو حلقه‌ی for هر عددی در آرایه با سایر اعداد مقایسه می‌شود و اگر کوچک‌تر

بود؛مکان آن‌ها در آرایه با یکدیگر تعویض می‌شود. این روند تا جایی ادامه می‌یابد که تمامی عناصر آرابه باهم مقایسه شده‌باشند و حلقه‌ی اول به پایان برسد. لازم است توجه کنیم که؛ هربار شمارنده‌ی حلقه‌ی اول یک واحد اضافه می‌شود،حلقه‌ی دوم یک بار کامل اجرا می‌شود.



شکل ۱.۲: Sort Function Algorithm

این نمونه کد در زبان CPP به شکل زیر نوشته می‌شود:

```cpp
void sort(std::vector<int>& nums)
{
        for (size_t i = 0; i < nums.size(); i++)
        {
                for (size_t j = i + 1; j < nums.size(); j++)
                {
                        if (nums[i] < nums[j])
                                swap(nums[i], nums[j]);
                }
        }
}
```

نمونه کد ۵: تابع Sort در زبان CPP

در نمونه کد فوقانی `size_t` در واقع برای استفاده از سایز یا شمار [‡] آورده شده‌است.که البته مصارف دیگری [**sizetUsage**] هم دارد.

- و اما شاهد استفاده از `vector` هستیم.

وکتور آرایه‌ای است که نیازی به اندازه ندارد.و در واقع در جایی که اندازه‌ی معینی برای آرایه در نظر نداریم،می‌توانیم از وکتور استفاده کنیم.به همین خاطر به وکتور `Dynamic Array` می‌گویند. هر بار که یک واحد جدید در وکتور ساخته می‌شود، آدرس خانه‌ی بعد را در خود ذخیره می‌کند.و این کار توسط پوینتر صورت می‌گیرد.

‡ count

می‌توان توابع آماده‌ای چون `pop_back()` را برای حذف آخرین خانه؛ و `push_back()` را برای اضافه کردن خانه به انتهای وکتور به کار برد. شایان توجه است برای استفاده از وکتور باید دستور `#include<vector>` را ضمیمه کنیم. [**vector**]

انواع دیگر توابع قابل استفاده برای وکتور در جدول زیر ذکر شده‌اند:

```
size()
assign()
swap()
emplace()
clear()
insert()
erase()
```

- در جلسه‌ی بعد به بررسی بیشتر توابع و پارامترها می‌پردازیم.

با آرزوی سعادتمندی

جلسه ۳

Functions

نیوشا یقینی - ۱۳۹۸/۱۱/۳۰

یک اصطلاحی به نام Top-down-approach داریم که به نوعی به یک سبک برنامه نویسی گفته می‌شود. روش آن بصورت مرحله به مرحله فکر کردن و اجرا کردن است که در آن با مشخص کردن قطعات پیچیده و سپس تقسیم آنها به قطعات پی در پی کوچکتر آغاز می شود. به عنوان مثال برای پخت کیک ابتدا باید به دستورالعمل مراجعه کرد که مراحل را عنوان کرده مانند مخلوط کردن، در قالب ریختن، ... سپس برای هر مرحله توضیحات جداگانه داده می‌شود. در واقع این همان کاری ست که function ها انجام می‌دهند؛ آن ها ورودی و خروجی های تعریف شده ای دارند و عملکرد های خاصی را دنبال می‌کنند. اجرا و نوشتن Top-down-approach را در هر مرحله Pseudocode می‌گوییم.

۱.۳ تعریف Function

یک Function برای تعریف شدن از الگوی خاصی پیروی می‌کند، به عنوان مثال نمونه ای از متدها را در زبان های مختلف را در زیر میبینید:

```
def func():
    print (world" "Hello)
```

نمونه کد ۶: مثالی در زبان

```
public void func()
{
    System.Console.WriteLine(world" "Hello);
}
```

نمونه کد ۷: مثالی در زبان c#

```
#include <iostream>
int func()
{
    std::cout << World!" "Hello;
}
```

نمونه کد ۸: مثالی در زبان c++

```
public class Sample {
    public void func() {
        System.out.println(world" "Hello);
    }
}
```

نمونه کد ۹: مثالی در زبان

۲.۳ انواع Function

۴ نوع function قابل تعریف و استفاده کردن است:

- متد بدون ورودی و بدون خروجی
- متد بدون ورودی و دارای خروجی
- متد با ورودی و بدون خروجی
- متد با ورودی و با خروجی

۳.۳ انواع ورودی ها

برای استفاده از توابع باید صدا زده شوند، برای صدا زدن آنها اگر تابع نیاز به ورودی اولیه داشت(ورودی ها می‌توانند از هر نوع integer، list، string، array، vector object و ... باشند) به ۲ صورت میتوان این ورودی ها را داد، البته باید به این نکته اشاره شود که برخی زبان های از جمله پایتون هستند که قابلیت تعریف نوع ورودی برای آنها امکان پذیر نیست و صرفا از حالت خاصی پیروی می‌کنند.

* vector قابلیتی مشابه آرایه دارد با این تفاوت که سایز آنها بصورت dynamically قابل تغیر هم هست. این قابلیت با درج کتابخانه اش قابل استفاده است و کاربرد های گسترده ای دارد.

- بصورت reference by pass
- بصورت value by pass

* **reference by pass** : در این حالت آدرس متغیر مورد نظر به تابع فرستاده می‌شود و هر تغیری روی متغیر مستقیما رویش اثر می‌کند.

* اگر به تابع در این حالت بخواهیم آرایه بگیریم در واقع تابع آدرس اولین شی موجود در آرایه را دریافت کرده است.

مثال: زبان های C، cpp، csharp، java ... قابلیت استفاده از این نوع را دارند.

```c
void swap(int* x, int* y)
{
   int temp;
   temp = *x;     */ x address at value the save /*
   *x = *y;       */ x into y put /*
   *y = temp;     */ y into temp put /*
}
int a = 1;
int b = 2;
swap(&a, &b);
```

نمونه کد ۱۰: مثالی در زبان `c`

```cpp
void swap(int& x, int& y)
{
   int temp;
   temp = x;  */ x address at value the save /*
   x = y;     */ x into y put /*
   y = temp;  */ y into x put /*
}
int a = 1;
int b = 2;
swap(a, b);
```

نمونه کد ۱۱: مثالی در زبان `c++`

```csharp
void swap(ref x, ref y)
{
    int temp = x;
    x = y;
    y = temp;
}
int a = 1;
int b = 2;
swap(ref a, ref b);
```

نمونه کد ۱۲: مثالی در زبان `csharp`

* اگر بخواهیم متغیری را به یک تابع پاس کنیم که مقدار اولیه به آن نداده باشیم با ارور مواجه می‌شویم. در این حالت می‌توان از out استفاده کرد.

* اگر بخواهیم دقیقا نوع متغیر ورودی را مشخص نکنیم می‌توان از معادل های کلی استفاده کرد؛

در `c#` => var یا در `c++` => auto یا ...

*** value by pass :**

در این حالت مقدار متغیر (یک کپی از متغیر مورد نظر) به تابع فرستاده می‌شود و هر تغیری روی متغیر مستقیما رویش اثر نمی‌کند.

در این حالت نیازی به استفاده از پوینتر یا مورد خاصی نیست.

۴.۳ انواع خروجی ها

خروجی ها که return های یک function هستند مقداری ست که تابع پس از انجام کار خود پاسخ می‌دهد. مقدار برگشتی می‌تواند هر یک از ۴ نوع متغیر باشد: لیست یا آرایه ، object integer، یا string .

تقریبا در تمام زبان ها به جز پایتون هنگام تعریف تابع نوع برگشتی آن نیز باید مشخص شود.

session this for sites useful some

- https://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/
- https://www.tutorialspoint.com/cprogramming/c_function_call_by_reference.htm
- https://www.tutlane.com/tutorial/csharp/csharp-pass-by-reference-ref-with-examples
- https://stackoverflow.com/questions/42039600/call-by-reference-doesnt-work-in-c-sharp-when-
- https://www.webopedia.com/TERM/F/function.html

جلسه ۴

Method/Function Features

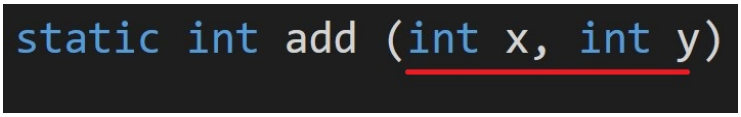
محمدمهدی جاوید - ۱۳۹۸/۱۱/۲۶

۱.۴ متد اورلودینگ

متد اورلودینگ: هرگاه چند تابع نام‌های یکسانی داشته باشند. اما سیگنیچرهای متفاوتی داشته باشند. یعنی اینکه تعداد پارامترهای ورودی آن‌ها متفاوت باشد.
*** پارامترهای ورودی آن‌ها متفاوت باشد. یعنی هم می‌توانند از نظر تعداد یا نوع تفاوت داشته باشند.
*** اگر پارامترهای ورودی تابع نوع داده‌ای متفاوت داشته باشند. با تغییر جای آن‌ها باز هم متد اورلودینگ خواهیم داشت.
*** نوع خروجی تابع (ریترن تایپ) تاثیری در متد اورلودینگ ندارد.

۱.۱.۴ سیگنیچر تابع

معنای سیگنیچر یک تابع وابسته به این است که از کلمه سیگنیچر در کجا استفاده خواهیم کرد.
اما در متد اورلودینگ سیگنیچر یک تابع پارامترهای ورودی یک تابع هستند.
قسمت‌هایی از شکل که زیر آن خط قرمز کشیده شده است. پارامترهای تابع هستند.



شکل ۱.۴: سیگنیچر یک تابع در متد اورلودینگ

۲.۱.۴ متد اورلودینگ با تعداد پارامترهای ورودی متفاوت

تابع اول ۴ پارامتر ورودی دارد.

تابع دوم ۳ پارامتر ورودی دارد.

تابع سوم ۲ پارامتر ورودی دارد.

هر سه تابع سیگنیچرهای متفاوتی دارند. و همانطورکه می‌بینید هیچ اروری در برنامه وجود ندارد. و خود برنامه با توجه به تعداد ورودی توابع موقع صدا زدن آن‌ها متوجه خواهد شد. که ما از کدام تابع در حال استفاده کردن هستیم. و آن تابع را صدا خواهد زد.

```
0 references
static int add (int x, int y, int z, int i) => x+y+z+i;
0 references
static int add (int x, int y, int z) => x + y + z; 3 paramteres
0 references
static int add (int x, int y) => x + y; 2 parameters
```

شکل ۲.۴: توابع با تعداد پارامترهای ورودی متفاوت

مثال زیر را حل کنید:

۱- در مثال زیر کدام تابع صدا خواهد شد؟

۲- خروجی برنامه چه خواهد بود؟

```
using System;

namespace MethodOverLoading
{
    class Program
    {
        static int Multiply (int x, int y, int z, int i, int j) => x * y * z * i * j; function-1 //
        static int Multiply (int x, int y, int z, int i) => x * y * z * i; function-2 //
        static int Multiply (int x, int y, int z) => x * y * z; function-3 //
        static int Multiply (int x, int y) => x * y; function-4 //
        static void Main(string[] args)
        {
            Console.WriteLine(Multiply(1, 2, 3));
        }
    }
}
```

نمونه کد ۱۳: سوال توابع با تعداد ورودی‌های متفاوت

جواب سوال :

خروجی برنامه : ۶

تابع شماره ۳ صدا زده خواهد شد. زیرا تعداد پارامترهای ورودی آن سه تا است.

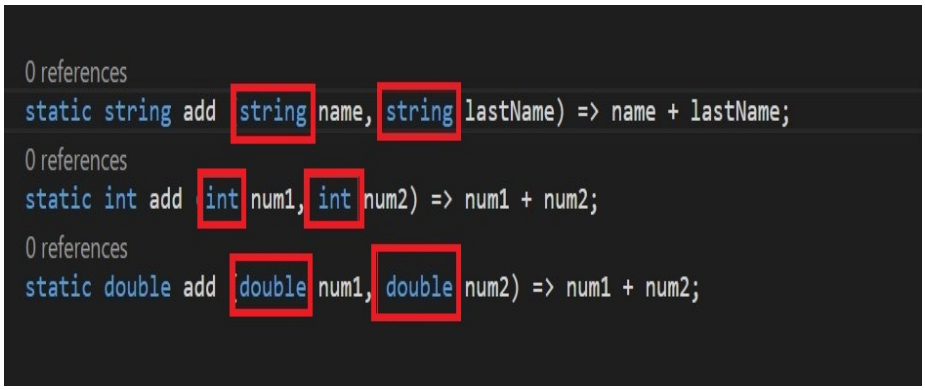
۳.۱.۴ متد اورلودینگ با نوع (تایپ) داده‌ای متفاوت :

یعنی اینکه اگر دو تابع نام‌های یکسانی داشته باشند. اما نوع (تایپ) هر یک از پارامترهای آن دو تابع متفاوت باشد.

*** وجود یک پارامتر با نوع داده‌ای متفاوت در بین دو تابع برای متد اورلودینگ کافی است.

مثال از چند نوع یا تایپ داده‌ای متفاوت :

int - double - string - bool



شکل ۳.۴: توابع با تعداد ورودی یکسان و نوع داده‌ای متفاوت

مثال زیر را حل کنید :

۱- کدام تابع صدا زده خواهد شد؟

۲- خروجی برنامه چه خواهد بود؟

```
using System;

namespace MethodOverLoading
{
    class Program
    {

        static string add (string name, string lastName) => name + lastName; 1 function //
        static int add (int num1, int num2) => num1 + num2; 2 function //
        static double add (double num1, double num2) => num1 + num2; 3 function //
        static void Main(string[] args)
        {
            Console.WriteLine(add("Ali", "Reza"));
        }
    }
}
```

نمونه کد ۱۴: سوال توابع با تعداد پارامترهای یکسان و نوع داده‌ای متفاوت

جواب سوال :

۱- تابع شماره یک صدا زده خواهد شد.

۲- خروجی برنامه = AliReza خواهد بود.

سوال :

چگونه می‌تواند متد اورلودینگ رخ دهد. در صورتی که نوع و تعداد پارامترهای ورودی یکسانی دو تابع داشته باشند؟

جواب سوال :

```
0 references
static void printStudent (int id, string studentName)
=> Console.WriteLine(studentName + id);
0 references
static void printStudent (string studentName, int id)
=> Console.WriteLine(studentName + id);
```

شکل ۴.۴: دو تابع که پارامترهای ورودی آن‌ها از نظر تعداد و نوع یکسان هستند. اما سیگنیچر متفاوتی دارند.

۴.۱.۴ متد اورلودینگ در بقیه زبان‌ها:

در تمامی زبان‌های برنامه‌نویسی (جاوا - سی‌پلاس‌پلاس - سی‌شارپ) متد اورلودینگ به طور مشابه طبق توضیحات بالا رخ می‌دهد. اما در پایتون اینطور نیست.

علت چیست ؟

در پایتون توابع آبجکت (شیء) ازکلاس تابع(فانکشن) هستند. و متغیرها در پایتون همانند پوینتر عمل می‌کنند. که به یک شیء اشاره می‌کنند. اگر ما تابعی با نام یکسان در پایتون بنویسیم. اخرین تابع نوشته شده. یا آخرین آبجکت(شیء) ساخته شده. استفاده خواهد شد. و تعاریف بالا از تابع بدون استفاده خواهند ماند. برای فهم بیشتر این موضوع کد زیر را نگاه کنید.

```
۱  def add (x) :
۲      return x
۳  def add (x, y) :
۴      return x + y
۵  def add (x, y, z) :
۶      return x + y + z
۷
۸  print(add(1))
۹  print(add(1, 2))
۱۰ print(add(1, 2, 3))
```

نمونه کد ۱۵: متداورلودینگ در پایتون

به نظر شما خروجی برنامه بالا چه خواهد بود؟

برنامه بالا اکسپشن خواهد داد

اما در چه خطی ؟

در خط هشت برنامه بالا اکسپشن خواهد داد. زیرا انتظار دارد. که موقع صدا زدن تابع add سه ورودی به آن تابع داده شود. اما یک پارامتر ورودی بیشتر به این تابع داده نشده.

همانطور که در کد بالا می‌بینید. آخرین تعریف و پیاده‌سازی برای تابع لحاظ می‌شود.

پس تنها راه صدا زدن این تابع دادن سه ورودی به آن است.

۲.۴ فرستادن براساس مقدار یا آدرس
Pass by reference VS Pass by value

۱.۲.۴ کلمه کلیدی ref
ref keyword

هنگامی که توابع را صدا می‌زنیم. می‌توانیم به‌جای پارامترهای ورودی آن‌ها مقدار بنویسیم. برای مثال عدد دو یا هر عدد دیگری را به عنوان ورودی به تابع بدهیم.
یا می‌توانیم. که عدد دو را در متغیری ذخیره کنیم. و آن متغیر را به تابع بدهیم.
وابسته به اینکه نوع داده‌ای (type) متغیری که به تابع موقع صدا زدن می‌دهیم. تابع رفتارهای متفاوتی خواهد داشت.

نوع داده‌هایی که به طور خودکار به صورت آدرس (reference types) به تابع داده خواهد شد:
۱- آبجکت(شیء) هایی که از کلاس‌ها ساخته می‌شوند.
۲- رشته‌ها .(string)
۳- لیست و آرایه.

نوع داده‌هایی که به صورت مقدار (value types) به تابع داده می‌شود:
۱- اعداد (دابل - فلوت - اینتیجر و...)
۲- بولین‌ها (مقادیر صحیح یا غلط)
۳- کارکتر (char)
۴- استراکت‌ها
۵- اینام‌ها (enums)

لیست بالا کامل نیست اما تعدادی از نوع داده‌های پرکاربرد را در خود جا می‌دهند.

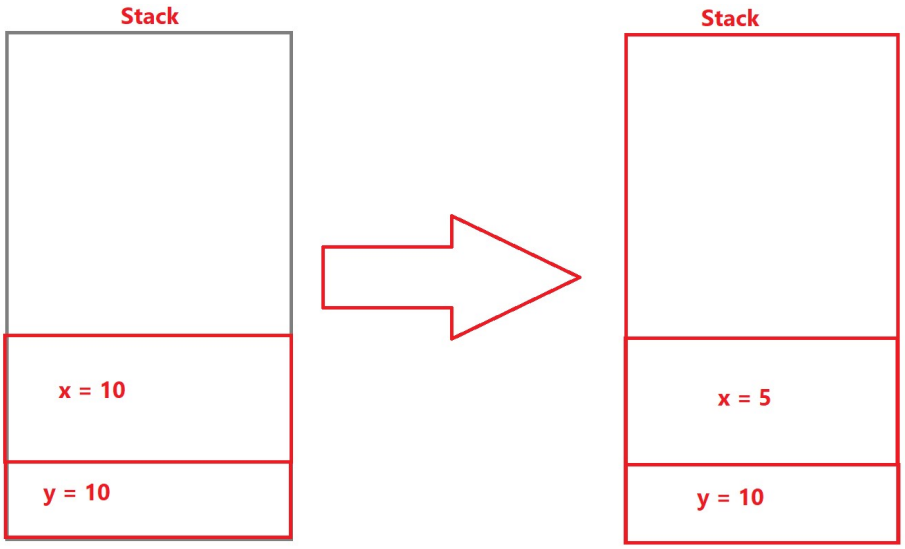
اگر ما هنگام صدا زدن تابع به آن نوع داده‌ای مقدار (value Type) بدهیم. همانند این است. که مقداری که در آدرس متغیر ذخیره شده. را در متغیر ورودی تابع ذخیر خواهد کرد. برای فهم بیشتر این موضوع به توضیحات زیر نگاه کنید.

```
static void NewGrade (int x)
{
    x = 5;
    Console.WriteLine(x);
}

static void Main(string[] args)
{
    int y = 10;
    NewGrade(y);
    Console.WriteLine(y);
}
```

نمونه کد ۱۶: value by pass

در مثال بالا در کنسول ابتدا عدد پنج و سپس عدد ده چاپ خواهد شد.
با اینکه ما متغیر y را به تابع NewGrade دادیم اما همانطور که می‌بینیم. مقدار درون متغیر تغییر نکرده است. هنگامی که ما تابع NewGrade را صدا می‌زنیم. برنامه مقدار درون متغیر y را نگاه می‌کند. و آن مقدار را کپی کرده. و در متغیر x که پارامتر ورودی تابع NewGrade است. قرار می‌دهد. اگر ما مقدار درون متغیر x عوض کنیم. هیچ تاثیری بر متغیر y نخواهد داشت. به همین علت به آن فرستادن بر اساس مقدار یا (pass by value) می‌گویند.

برای مفهوم شدن مطالب بالا به عکس زیر هم نگاه کنید:



شکل ۵.۴: pass by value

سوال: حال اگر بخواهیم که متغیرهای x و y آدرس یکسانی بر روی استک داشته باشند چکار باید کنیم؟
*** اگر دو متغیر آدرس یکسان استک داشته باشند. با تغییر یکی، دیگری نیز تغییر خواهد کرد. حتی اگر این دو متغیر در دو اسکوپ متفاوت باشند.

*** نکته: تنها داده‌هایی که از نوع مقدار (value types) هستند. در توابع متفاوت مستقل از هم هستند. اگر نوع داده‌ای ما از نوع (reference types) باشد. هر تغییری که در یکی از توابع بدهیم. در همه جا اعمال می‌شود. زیرا تنها چیزی را که به ما تابع دیگری می‌دهیم. پوینتر به آدرس هیپ آن است. و اگر ما تغییری در آن آدرس هیپ بدهیم. در همه جاهایی که به آن اشاره می‌کنند. تغییر خواهد کرد.

اگر بخواهیم که داده‌هایی که از نوع مقدار (value types) هستند. آدرس استک آن‌ها را موقع صدا زدن آن تابع به آن‌ها بدهیم. کافی است. که از کلمه کلیدی ref در سیگنیچر تابع و در هنگام دادن آن متغیر به تابع استفاده کنیم.

کلمه کلیدی ref (ref keyword) :

هرگاه که ما تابعی را صدا بزنیم. و بخواهیم که آدرس متغیر داده شده به تابع با آدرس پارامتر ورودی تابع یکسان باشد. از این کلمه کلیدی استفاده خواهیم کرد.

اگر آدرس هر دو متغیر یکسان باشند. پس با تغییر هر یک از آن‌ها دیگری نیز تغییر خواهد کرد.

سوال: خروجی برنامه زیر چه خواهد بود؟

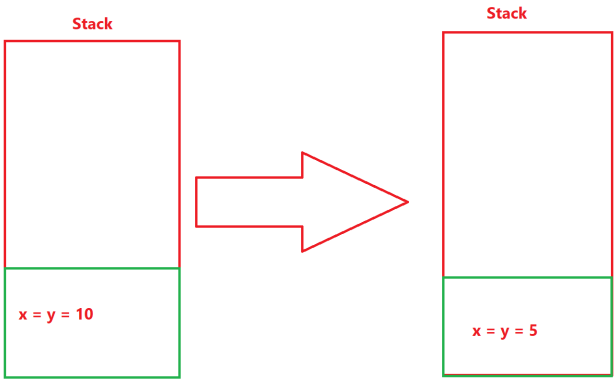
```
۱  static void NewGrade (ref int x)
۲  {
۳      x = 5;
۴      Console.WriteLine(x);
۵  }
۶
۷  static void Main(string[] args)
۸  {
۹      int y = 10;
۱۰     NewGrade(ref y);
۱۱     Console.WriteLine(y);
۱۲ }
```

نمونه کد ۱۷: keyword ref

*** نکته‌ای که در این سوال وجود دارد. این است که چون دو متغیر x و y قبل از آن‌ها از کلمه کلیدی ref استفاده شده. پس آدرس استک آن‌ها یکسان خواهد بود. و با تغییر یکی دیگری نیز تغییر خواهد کرد.

پس خروجی برنامه در کنسول عدد ۵ و ۵ خواهد بود. که نشان دهنده این است که هر دو متغیر آدرس یکسانی دارند. و با تغییر یکی دیگری نیز تغییر می‌کند.

برای تفهیم بیشتر این موضوع به عکس زیر نگاه کنید.



شکل ۶.۴: ref keyword

*** تمامی داده‌هایی که بر روی استک قرار می‌گیرند. اگر بخواهیم که از آن‌ها استفاده کنیم. در زمان کامپایل شدن برنامه باید اندازه آن‌ها مشخص باشد. و مقدار دهی شوند. یا Initialize شوند. پس نمی‌توانیم که داده‌ای داشته باشیم. که برروی استک باشد. و از آن استفاده کنیم. و مقدار دهی اولیه نشده باشد. زیرا با ارور Use of unassigned local variable روبرو خواهیم شد.

چگونه می‌توانیم. که از یک تابع چند خروجی داشته باشیم؟ به طور کلی امکان داشتن چند خروجی از یک تابع میسر نیست. اما می‌توان از روش‌های دیگری استفاده کرد. یکی از راه‌های آن استفاده از کلمه کلیدی out است. اگر ما قبل از متغیری از کلمه out استفاده کنیم. می‌توانیم از آن متغیر استفاده کنیم. در صورتی که آن متغیر مقدار دهی اولیه نشده باشد.
*** در صورتی که از کلمه کلیدی out استفاده می‌کنیم. حتما باید که در تابع دوم مقداردهی شود.

کلمه کلیدی out (out keyword) :

هر گاه بخواهیم. که متغیری را به عنوان پارامتر ورودی به تابع دیگری بدهیم. و آن را در تابع دوم که صدا زده شده مقدار دهی اولیه کنیم. می‌توانیم از این کلمه کلیدی استفاده کنیم.
*** در تابع اول که این متغیر تعریف می‌شود. به هیچ وجه نباید مقدار دهی اولیه شود.
*** هنگام استفاده از این کلمه کلیدی آدرس استک هر دو متغیر یکسان خواهد شد.
*** یکی از راه‌هایی که بتوانیم چند خروجی از یک تابع داشته باشیم.

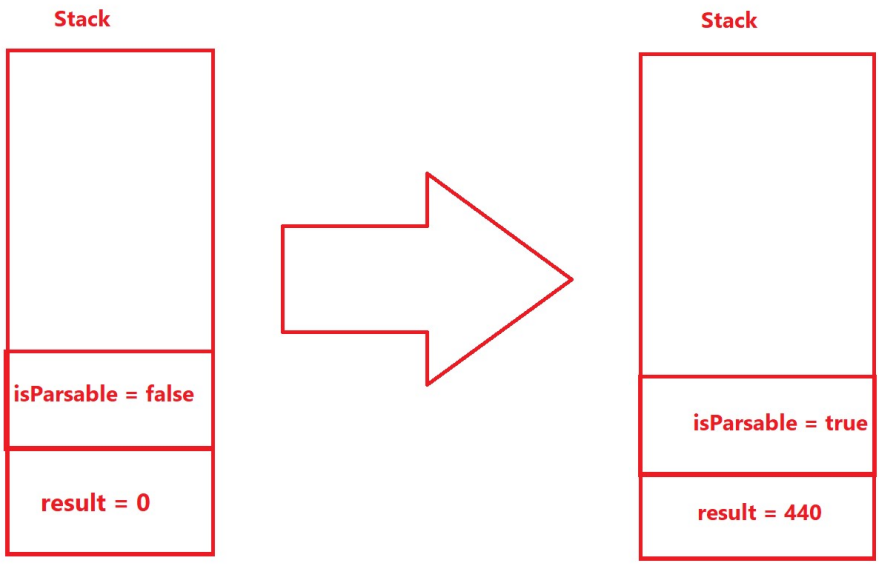
برای تفهیم بیشتر این مطلب به کد زیر نگاه کنید.

```
۱  static void Main(string[] args)
۲  {
۳      int result;
۴      bool isParsable = int.TryParse("440", out result);
۵      Console.WriteLine(result + "\t" + isParsable);
۶  }
```

نمونه کد ۱۸: out keyword

همانطور که می‌بینید در کد بالا ما از متغیر result استفاده کردیم. بدون آنکه به آن را مقدار دهی اولیه کنیم. که این تنها با استفاده از کلمه کلیدی out امکان پذیر است. الان تابع int.TryParse() دو خروجی به ما خواهد داد. که یکی از آن‌ها در متغیر result و دیگری در متغیر isParsable ذخیره خواهد شد. خروجی این برنامه این است که مقدار متغیر result برابر است با ۴۴۰ و مقدار متغیر isParsable برابر است با true که معنای آن این است. که این رشته به عدد صحیح قابل تبدیل است.

برای تفهیم بیشتر این مطلب به عکس زیر نگاه کنید :



شکل ۷.۴: out keyword

جرا مقدار اولیه متغیر result مساوی با صفر است؟
داده‌هایی که روی استک تعریف می‌شوند. می‌توانند مقدار پیشفرض بگیرند. این اتفاق در فیلدهای کلاس‌ها یا ساخت آرایه‌ای از نوع اعداد صحیح و... می‌تواند رخ دهد.
مقدار پیشفرض بعضی از داده‌ها را در پایین می‌آوریم :

the first constant in the enum : enum

value of zero : int

false : bool

'\0' : char

فرق کلمه کلیدی ref با out :

۱- با استفاده از هر دو کلمه آدرس استک دو متغیر یکسان خواهد شد. پس یعنی با تغییر یکی دیگری نیز تغییر خواهد کرد.
۲- تنها زمانی می‌توان از کلمه کلیدی out استفاده کرد. که در تابع دوم که صدا زده شده. متغیر مقدار بگیرد. اما در استفاده از کلمه کلیدی ref حتما باید که متغیر ما در تابع اولیه مقدار دهی اولیه شده باشد. تا بتوان از آن استفاده کرد.

نکته : آیا می‌توان از کلمه کلیدی ref برای نوع داده‌ای از نوع رفرنس نیز استفاده کرد؟
بله اما بودن یا نبودن آن تاثیری نخواهد داشت.

۳.۴ توابع با پارامترهای ورودی متغیر (variadic functions) :

۱.۳.۴ C# :

اگر بخواهیم تابعی بنویسیم. که از تعداد پارامترهای ورودی آن‌ها اطلاعی نداشته باشیم. از توابع متغیر (variadic functions) استفاده می‌کنیم.

برای ساخت توابع متغیر یا (variadic functions) کافی است که نوع داده‌ای که تعدادش نامشخص است. قبل از آن از کلمه کلیدی params استفاده کنیم. و نوع داده‌ای آن را به آرایه‌ای از آن نوع تغییر دهیم. *** نکته بسیار مهم این است که همیشه باید نوع داده‌ای با تعداد متغیر به عنوان آخرین پارامتر ورودی به تابع داده شود.

برای تفهیم بیشتر مطالب به کدهای زیر نگاه کنید.

```
۱  public static int Sum (params int[] numbers)
۲  {
۳      int result = 0;
۴      foreach (int number in numbers)
۵      result += number;
۶      return result;
۷  }
۸  static void Main(string[] args)
۹  {
۱۰     Console.WriteLine(Sum(1, 2, 3, 4, 5, 6));
۱۱     int[] numbers = {3, 4, 5, 6};
۱۲     Console.WriteLine(Sum(numbers));
۱۳ }
```

نمونه کد ۱۹: variadic functions

خروجی برنامه بالا ابتدا عدد ۲۱ و سپس عدد ۱۸ خواهد بود.
همانطور که می‌بینید تعداد ورودی های تابع Sum می‌توانند هر چقدر باشند. و محدودیتی ایجاد نمی‌کنند.

پارامتر ورودی تابع Sum می‌تواند هم به صورت آرایه و هم به صورت اعداد جدا شده با کاما باشد. و فرقی با یکدیگر ندارند.

اشتباه رایج :

به کد زیر نگاه کنید. و اشتباه آن را بیابید.

```
public static int HowManyMatches (params int[] numbers, int matchNumber)
{
    int totalMatch = 0;
    foreach (int number in numbers)
        if (number == matchNumber)
            totalMatch ++;
    return totalMatch;
}
static void Main(string[] args)
{
    Console.WriteLine(HowManyMatches(1, 2, 3, 4, 5, 2, 3, 1, 2));
}
```

نمونه کد ۲۰: params Keyword

مشکل برنامه بالا در کجاست؟

همانطور که در بالاتر گفته شد. اگر قرار است. که توابع تعداد پارامترهای ورودی متفاوتی داشته باشند. الزاما باید به عنوان آخرین پارامتر به تابع داده شوند.

پس کد زیر نحوه صحیح کد بالا را نمایش می‌دهد.

```
public static int HowManyMatches (int matchNumber, params int[] numbers)
{
    int totalMatch = 0;
    foreach (int number in numbers)
        if (number == matchNumber)
            totalMatch ++;
    return totalMatch;
}
static void Main(string[] args)
{
    int[] numbers = {1, 2, 3, 4, 5, 2, 3, 1,};
    int match = 2;

    Console.WriteLine(HowManyMatches(match, numbers));
    Console.WriteLine(HowManyMatches(2, 1, 2, 3, 4, 5, 2, 3, 1));
}
```

نمونه کد ۲۱: params keyword

برنامه بالا تعداد عدد ۲ (match) را در اعداد (numbers)

پیدا می‌کند. که همانطور که می‌بینید. دو بار عدد ۲ در اعداد بالا تکرار شده است.
و خروجی برنامه عدد ۲ خواهد بود.

Python : ۲.۳.۴

در پایتون کافی است. که قبل از آن پارامتر ورودی که قرار است. تعداد متفاوتی از مقادیر را بگیرد. یک ستاره (*) (asterisk) قرار دهیم.
۱- پارامتر ورودی که چند مقدار می‌گیرد. از نوع داده‌ای توپل (tuple) خواهد بود.
۲- در پایتون می‌توانیم. که علاوه بر اینکه پارامترهای ورودی تابع مقادیر مختلفی می‌گیرند. بتوانند که کلمه کلیدی نیز بگیرند. که نوع داده‌ای آن‌ها به دیکشنری تغییر خواهد کرد. برای این کار کافی است. که از دو تا ستاره (**) قبل از آن پارامتر ورودی تابع استفاده کنیم.

خلاصه‌ای از توپل (Tuple) : مجموعه‌ای از داده‌ها که قابل تغییر نیستند. و به وسیله کاما از یکدیگر جدا می‌شوند. و ترتیب آن‌ها نیز اهمیت دارند. با استفاده از ایندکس می‌توان از المان‌های درون توپل استفاده کرد.

```
languages_and_their_grades = ("Python", "Java", "C++", "C#", 10, 9, 8, 7)
item1 = languages_and_their_grades[0]
print(item1) "Python" #
```

نمونه کد ۲۲: Python Tuples

خلاصه‌ای از دیکشنری (Dictionary) : مجموعه‌ای از داده‌ها که ترتیب آن‌ها اهمیتی ندارد. و المان‌های درون آن به صورت جفت‌جفت کلید و مقدار (key-value pairs) ذخیره می‌شوند. المان‌های درون دیکشنری قابل تغییر هستند. با استفاده از کلیدهای درون دیکشنری می‌توان از آن‌ها استفاده کرد.

```
languages_and_their_grades = { "Python" : 100 ,
                               "C++" : 99 ,
                               "C#" : 80 ,
                               "Java" : 90 }
grade1 = languages_and_their_grades["Python"]
print(grade1) #100
```

نمونه کد ۲۳: Python Dictionary

برای تفهیم بیشتر توابع با پارامترهای ورودی متفاوت در پایتون به کد زیر نگاه کنید:

```python
def sum_of_numbers (*numbers):
    result = 0
    for number in numbers :
        result += number
    return result

print(sum_of_numbers(1, 2, 3, 4, 5, 6))
```

نمونه کد ۲۴: Python Variable-length Arguments

خروجی برنامه بالا عدد ۲۱ است. تمامی اعداد در توپلی به نام numbers قرار می‌گیرند.

$$\text{numbers} = (1, 2, 3, 4, 5, 6)$$

```python
def print_max_grade_with_course (**languages_with_grades):
    max_grade = 0
    course_with_max_grade = None

    for course, grade in languages_with_grades.items() :
        if grade > max_grade :
            max_grade = grade
            course_with_max_grade = course

    print(course_with_max_grade)
    print(max_grade)

print_max_grade_with_course( Python = 100 ,
                             Cpp = 99     ,
                             JAVA = 95    ,
                             Cs = 90      )
```

نمونه کد ۲۵: Python Variable-length KeywordArguments

خروجی برنامه بالا Python و 100 خواهد بود.

برای تفهیم بیشتر سوال بالا ما دیشکنری به نام languages_with_grades داریم. که کلیدهای آن نام دروس و مقدارهای آن نمرات آن‌ها هستند.

Java : ۳.۳.۴

در جاوا به جای استفاده از کلمه کلیدی params کافی است. که بعد از نوع داده‌ای آن سه نقطه(...) بگذاریم.
که به آن Ellipses می‌گویند.
*** نباید بیش از یک پارامتر ورودی با طول داده‌ای متفاوت در سیگنیچر تابع باشد.
*** می‌توان از آرایه نیز استفاده کرد.

برای تفهیم بیشتر مطالب به کد زیر نگاه کنید:

```java
public static int Sum (int... numbers) {
    int result = 0;
    for (int number : numbers)
        result += number;
    return result;
}
public static void main(String[] args) {
    int[] numbers = {1, 2, 3, 4, 5};
    System.out.println(Sum(1, 2, 3, 4, 5));
    System.out.println(Sum(numbers));
}
```

نمونه کد ۲۶: Java Variable-length Arguments

خروجی کد بالا عدد ۱۵ خواهد بود.
در تابع Sum آرگومان numbers آرایه‌ای از اعداد صحیح است.

۴.۴ توابع و کلاس‌های عمومی (Generic Functions) :

اگر بخواهیم. تابع یا کلاسی بنویسیم. که برای نوع داده‌ای(تایپ) متفاوت کار کند. برای اینکه نیاز به کپی کردن آن کد برای نوع داده‌های متفاوت نباشد. می‌توان کاری کرد. که کار یکسانی را این توابع و کلاس‌ها برای برای نوع‌های داده‌ای متفاوت انجام دهند. حال این موضوع را در زبان‌های مختلف بررسی می‌کنیم.

Csharp :

برای نوشتن توابع یا کلاس‌های عمومی و یا اینترفیس‌هایی که از نوع‌های داده‌ای متفاوت پیروی کنند. کافی است. که با علامت‌های کوچک‌تر و بزرگتر نامی را برای آن نوع داده‌ای انتخاب کنیم.

برای تفهیم بیشتر مطالب به کدهای زیر نگاه کنید.

توابع عمومی :

```
public static void print <_Type> (params _Type[] collection)
{
    foreach (_Type element in collection)
        Console.WriteLine(element);
}
static void Main(string[] args)
{
    print<string>("Python", "C++", "C#", "Java");
    print("Python", "C++", "C#", "Java");
    print<object>(Reza" "Ali, 22, true, 'M');
}
```

نمونه کد ۲۷: Generic Functions

در برنامه بالا، ما می‌توانیم. که هر داده‌ای را در کنسول نمایش دهیم. فارغ از اینکه نوع آن داده چه باشد.

مثال زیر را هم برای تفهیم بیشتر نگاه کنید.

```
static void Swap <_Type> (ref _Type element1, ref _Type element2)
{
    _Type hold = element1;
    element1 = element2;
    element2 = hold;
}
static void Main(string[] args)
{
    int num1 = 1, num2 = 2;
    Swap(ref num1, ref num2);
    Console.WriteLine(num1);
    Console.WriteLine(num2);

    double dnum1 = 5.1, dnum2 = 5.3;
    Swap(ref dnum1, ref dnum2);
    Console.WriteLine(dnum1);
    Console.WriteLine(dnum2);
}
```

نمونه کد ۲۸: Generic Functions

خروجی برنامه بالا چه خواهد بود؟

جواب :

2

1

3.5

1.5

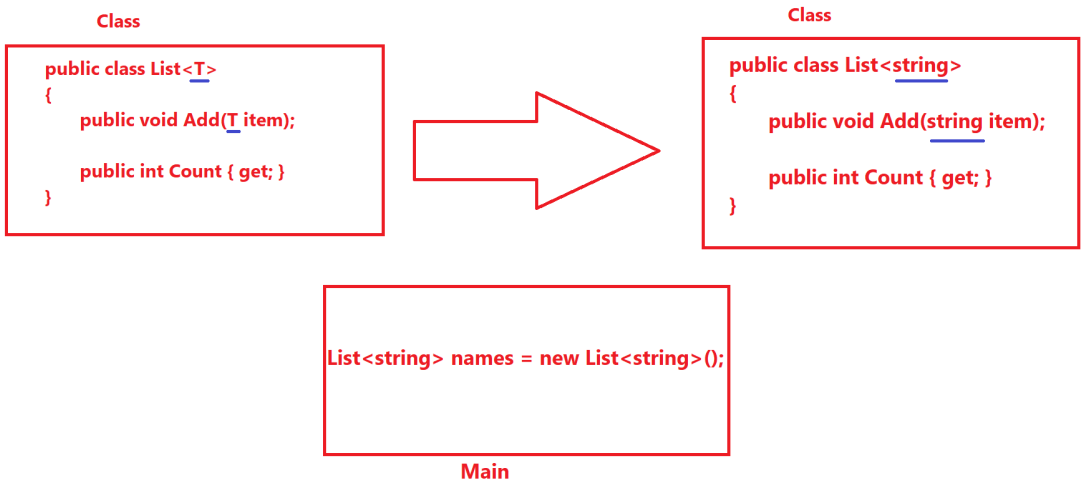
کلاس‌ها یا اینترفیس‌های عمومی (Generic classes and interfaces) :

مثال خوبی که از کلاس‌های عمومی وجود دارد. لیست است. که در System.Collections.Generic قرار دارد.

```
۱  public class List<T>
۲  {
۳      public void Add(T item);
۴      public int Count { get; }
۵  }
۶  static void Main(string[] args)
۷  {
۸      List<string> names = new List<string>();
۹      List<int> grades = new List<int>();
۱۰ }
```

نمونه کد ۲۹: Generic Class

کلاس بالا را در عکس زیر تشریح شده
به جای T می‌توان از کلمه string استفاده کرد.



شکل ۸.۴: generic class

اینترفیس نیز وضعیتی مشابه کلاس دارد که در زیر نمونه کدی برای آن آورده شده.

```
۱ public interface IEnumerable<_Type>
```

نمونه کد ۳۰: Generic Interface

Java :

برای توابع عمومی در زبان جاوا نیز کافی است. که نامی برای آن نوع داده‌ای انتخاب کنیم. و آن را قبل از نوع داده‌ای خروجی تابع بنویسیم.
به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید.

```
۱ public static <_Type> void print (_Type... collection) {
۲     for (_Type element : collection)
۳         System.out.println(element);
۴ }
۵ public static void main(String[] args) {
۶     print("Reza", true, 123, 'M');
۷     print(140, 70, 2);
۸ }
```

نمونه کد ۳۱: Generic Method

خروجی برنامه بالا :

Reza

true

123

M

140

70

2

Python :

پایتون زبانی با نوع داده‌ای پویا dynamically typed است. یعنی اینکه نوع داده‌ای متغیرها توسط مترجم زبان پایتون زمانی مشخص می‌شود. که برنامه در حال اجرا شدن است. و نوع داده‌ای متغیرها در طول برنامه مجاز به تغییر هستند.

پس از آن‌جا که نوع داده‌ای متغیرها قابل تغییر هستند. و نیازی به از پیش تعیین شدن آن‌ها نیست. پس نیازی به توابع و کلاس‌های عمومی نیست.

۵.۴ قراردادهای نام گذاری (Naming Conventions) :

نکته بسیار مهم : از به کار بردن کلمات کلیدی هر زبان به جای نام متغیرها جدا خودداری کنید.
نام چند قرارداد نام گذاری معروف را در زیر می‌آوریم:

Famous Naming Conventions :

Pascal Casing : هر گاه که نام یک معرف یا معین کننده هویت (Identifier) از چند کلمه تشکیل شده باشد. و تمامی کلمات حرف اول آن‌ها به صورت بزرگ Upper-case نوشته شود. به آن پاسکال کیس می‌گوییم.
به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

1- FirstName
2- LastName
3- StudentId
4- NameOfVariable

Camel Casing : هر گاه که نام یک معرف یا معین کننده هویت Identifier از چند کلمه تشکیل شده باشد. و کلمه اول آن. حرف اول آن به صورت کوچک Lower-case نوشته شود. و کلمات بعدی حروف اول آن‌ها بزرگ نوشته شوند. به آن کمل کیس می‌گوییم.
به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

1- firstName
2- lastName
3- studentId
4- nameOfVariable

Hungarian notation : هر گاه که نام یک معرف یا معین کننده هویت (Identifier) به این صورت نوشته شود که کلمه اول نام متغیر بیانگر نوع داده‌ای آن باشد. و ادامه کلمات نام متغیر به صورت پاسکال کیس نوشته شوند.

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

1- strFirstName (string)

2- strLastName (string)

3- iStudentId (integer)

4- bNameOfVariable (boolean)

Screaming Caps : هر گاه که نام یک معرف یا معین کننده هویت (Identifier) همگی حروف آن به صورت بزرگ نوشته شوند. Upper-case یا به طور دیگر تمامی حروف با Caps Lock نوشته شده باشند. طوری که All Caps نام بگیرند.

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

1- FIRSTNAME

2- LASTNAME

3- STUDENTID

4- NAMEOFVARIABLE

Snake case : هر گاه که نام یک معرف یا معین کننده هویت (Identifier) بین کلمات آن از under-line (underscore) استفاده شود. و حروف کلمات به صورت کوچک نوشته شده باشند.

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

1- first_name

2- last_name

3- student_id

4- name_of_variable

Csharp :

Methods and Classes : در زبان سی‌شارپ بهتر آن است. که نام متدها و کلاس‌ها را به صورت پاسکال کیس نوشته شوند.

به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید:

```
public class ClientActivity
{
    public void ClearStatistics()
    {
       ......
    }
}
```

نمونه کد ۳۲: Method and Classes Naming convention

Method Arguments and Local Variables : در زبان سی شارپ بهتر آن است. که پارامترهای ورودی توابع و متغیرهای که مخصوص یک تابع هستند. و در اسکوپ تابع هستند. به صورت کمل کیس نوشته شوند.

به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید:

```
public void Func (int studentId)
{
    string name = "Ali";
}
```

نمونه کد ۳۳: Method Arguments and Local Variables

Constants or Readonly Variables : در زبان سی‌شارپ بهتر است. که متغیرهایی که مقدار ثابت در طول برنامه دارند. یا فقط در سازنده مقدار دهی اولیه می‌شوند. با پاسکال کیس نوشته شوند.
به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید :

```
public static const Id = 98521000;
```

نمونه کد ۳۴: Constants or Readonly Variables

Interface : در زبان سی‌شارپ بهتر است. که نام اینترفیس‌ها با حرف بزرگ I شروع شود. به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید :

```
1 public interface IComparable
2 {
3 }
```

نمونه کد ۳۵: Interface

vertically align curly brackets : در زبان سی‌شارپ بهتر است. که آکولادهای باز و بسته در یک خط و بعد از سیگنیچر توابع باشند.
به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید :

```
1 static void Main(string[] args)
2 {
3     ......
4 }
```

نمونه کد ۳۶: Curly brackets

Fields and Properties : در زبان سی‌شارپ، در کلاس‌ها بهتر است. که نام فیلدها را به صورت کمل کیس و نام پراپرتی‌ها(ویژگی) را به صورت پاسکال کیس بنویسیم.
می‌توانیم برای متغیرهایی که پرایوت(خصوصی) هستند. ابتدای نام آن‌ها از آندرلاین استفاده کنیم.
به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید :

```
1 public class Student
2 {
3     private string _name;
4     public String Name
5     {
6         get => _name;
7         set => _name = value;
8     }
9 }
```

نمونه کد ۳۷: Fields and Properties

Python :

Classes : در زبان پایتون، نام کلاس‌ها باید پاسکال کیس باشند. اگرچه که معمولا کلاس‌های خود پایتون با حروف کوچک نام گذاری شده‌اند. کلاس‌های اکسپشن نیز باید با کلمه Error ختم بشوند.

```
۱ class Student:
```

نمونه کد ۳۸: Classes in Python

Function and Variable Names : در زبان پایتون، نام متغیرها و توابع باید به صورت حروف کوچک باشد. و کلمات با آندرلاین از یکدیگر جدا بشوند.

```
۱ def add_two_numbers (number1, number2):
۲     return number1 + number2
```

نمونه کد ۳۹: Functions and Variables

Java :

Classes and Interfaces : در زبان جاوا نام کلاس‌ها و اینترفیس‌ها به صورت پاسکال کیس است.

```
۱ class ImageSprite;
۲ interface Sport
```

نمونه کد ۴۰: Classes and Interfaces

Methods : در زبان جاوا، نام متدها باید به صورت کمل کیس باشد.

```
۱ public static int addNumbers (int num1, int num2)
```

نمونه کد ۴۱: Methods

Variables : در زبان جاوا، نام متغیرها نیز به صورت کمل کیس است.

```
۱ int studentId;
```

نمونه کد ۴۲: Methods

۶.۴ Fold Expression :

از بی‌سوادی‌ام (محمدمهدی جاوید) در زبان سی‌پلاس‌پلاس عذر خواهی می‌کنم.

چگونه می‌توانیم در سی‌پلاس‌پلاس تابعی داشته باشیم. که کار مشخصی را روی هر نوع داده‌ای پارامتر ورودی انجام دهد ؟
کافی است. که در سگنیچر متد کلمه تمپلیت را درون دو علامت بزرگتر و کوچکتر قرار داده. و ابتدا کلمه تایپ نیم را نوشته و سپس نامی برای آن نوع داده‌ای انتخاب کنیم.
نکته مهم : شما می توانید به جای کلمه typename از کلمه class نیز استفاده کنید.
به مثال زیر نگاه کنید:

```
template <typename T>
auto add (T num1, T num2)
{
    return num1 + num2;
}
int main(int argc, char const *argv[])
{
    cout << add(1, 2) << endl;
    cout << add(1.1, 6.1) << endl;}
    cout << add(true, true) << endl;
}
```

نمونه کد ۴۳: Fold Expression

خروجی برنامه بالا به صورت زیر است:

```
3
7.2
2
```

حال چگونه تابعی بنویسیم. که علاوه بر نامشخص بودن نوع داده‌های ورودی آن تعداد زیادی ورودی بتواند بگیرد؟

```cpp
template <typename ..._Types>
auto sum (_Types ...numbers)
{
    return (numbers + ...);
}
int main(int argc, char const *argv[])
{
    cout << sum(1, 2, 67.5, true, 0001.0) << endl;
}
```

نمونه کد ۴۴: Fold Expression

خروجی برنامه بالا 71.5001 است.

توضیح : پارامتر پک بالا به صورت زیر باز می‌شوند.

$$((\text{pack1} + \text{pack2}) + ...) + \text{packN}$$

برای تفهیم بیشتر به مثال زیر نیز نگاه کنید :

```cpp
template <typename T>
void print(T arg)
{
    cout << arg << endl;
}
template <typename T, typename ..._Types>
void print(T first, _Types ...args)
{
    cout << first << endl;
    print(args...);
}
int main(int argc, char const *argv[])
{
    print("Mahdi", 17, 023.0, true, 'J');
}
```

نمونه کد ۴۵: Fold Expression

خروجی برنامه به صورت زیر خواهد بود :

Mahdi

17

0.023

1

J

توضیحات : در ابتدا رشته مهدی و پارامتر پک را به تابع پرینت با دو ورودی می‌دهد. سپس هربار پارامتر اول که در ابتدا مهدی است. را پرینت می‌کند. سپس پارامتر پک را باز می‌کند. و به عنوان ورودی اول به تابع پرینت با دو ورودی می‌دهد. در انتها که پارامتر پک نداریم. و فقط یک ورودی داریم. تابع پرینت با یک ورودی را صدا می‌زند. پس اگر تابع پرینت با یک ورودی نباشد. قطعا با ارور مواجه می‌شویم.

در زیر راهی برای ذخیره ورودی‌های تابع در یک ساختمان داده را بررسی می‌کنیم.
*** از آن‌جا که ساختمان‌داده فقط یک نوع می‌تواند داشته باشد. پس حتما تمامی ورودی‌ها باید نوع مشابه داشته باشند.
در اینجا فقط توضیح کوتاهی داده شده. جهت مشاهده کاربرد این مطلب به بخش تمرین‌های سرکلاسی آن جلسه مراجعه کنید.

```
۱  template <typename T, typename... _Types>
۲  T Sum(_Types ...args)
۳  {
۴      vector<T> numbers = {args ...};
۵      T result = 0.0;
۶      for(T number : numbers)
۷          result += number;
۸      return result;
۹  }
۱۰ int main(int argc, char const *argv[])
۱۱ {
۱۲     cout << Sum<int>(1, 3, 10) << endl;
۱۳     cout << Sum<double>(1.1, 2.001, 0.3) << endl;
۱۴ }
```

نمونه کد ۴۶: Fold Expression

خروجی برنامه به صورت زیر خواهد بود :

14

3.401

در خط ۴ ما تمامی پارامتر پک را درونی ساختمان داده‌ای به نام وکتور ریخته‌ایم. و می‌توانیم از آن

استفاده کنیم.

به کد زیر نگاه کنید :

```
vector<T> numbers = {args ...};
```

۷.۴ نحوه صحیح تقسیم کردن دو عدد بر یکدیگر :

اگر بخواهیم دو عدد را بر یکدیگر تقسیم کنیم. و جواب دقیقی را به دست آوریم. الزاما باید یکی از اعداد از نوع داده‌ای دابل یا فلوت باشند. که نتیجه نیز به صورت اعشاری به ما داده شود.

نکته :

حاصل تقسیم دو عدد صحیح بر یکدیگر به صورت عادی محاسبه می‌شود. با این تفاوت که بخش اعشاری آن جدا می‌شود.

نکته بسیار مهم :

اگر دو عدد صحیح بر یکدیگر تقسیم شوند. مانند Floor Division عمل نخواهد کرد. و عدد به کوچکترین عدد نزدیکش گرد نخواهد شد. الزاما بخش اعشاری جدا خواهد شد.

برای تفهیم بیشتر مطالب به مثال‌های زیر نگاه کنید :

```
۱  int main(int argc, char const *argv[])
۲  {
۳      int num1 = 7;
۴      int num2 = 5;
۵      cout << num1/num2 << endl;
۶      cout << (double)num1/num2 << endl;
۷      cout << (float)num1/num2 << endl;^^I
۸  }
```

نمونه کد ۴۷: Division

خروجی برنامه بالا به ترتیب :

```
1
1.4
1.4
```

خواهد بود. در ظاهر شاید به نظر برسد. که به عدد کوچکترش گرد شده. اما در واقعیت اینطور نیست. و بخش اعشاری جدا شده. برای تفهیم بیشتر به مثال زیر نگاه کنید:

```cpp
int main(int argc, char const *argv[])
{
    int num1 = -7;
    int num2 = 5;
    cout << num1/num2 << endl;
    cout << (float)num1/num2 << endl;
}
```

نمونه کد ۴۸: Division

اگر فرض کنیم. که به عدد کوچکترش گرد می‌شود. خروجی اول باید عدد 2- باشد. اما در صورتی که عدد 1- است.

خروجی برنامه بالا : -1

-1.4

خواهد بود.

برتری پایتون در عمل تقسیم :

در زبان پایتون اگر شما دو عدد صحیح را نیز بر یکدیگر تقسیم کنید. باز هم خروجی به طور صحیح نشان داده خواهد شد.

```
num1 = int(2)
num2 = int(3)
print(type(num1))
print(type(num2))

result = num1/num2
print(type(result))
print(result)
print(num1/num2)^^I
```

نمونه کد ۴۹: Division

خروجی برنامه بالا به صورت زیر است :

<class 'int'>

<class 'int'>

<class 'float'>

0.6666666666666666

0.6666666666666666

۸.۴ تمرین کلاسی :

سوال : برنامه‌ای بنویسید که هر تعداد و نوع داده ورودی را بتواند. برعکس کند. و سپس تابع دیگری بنویسید. که آن داده‌ها را نمایش دهد.

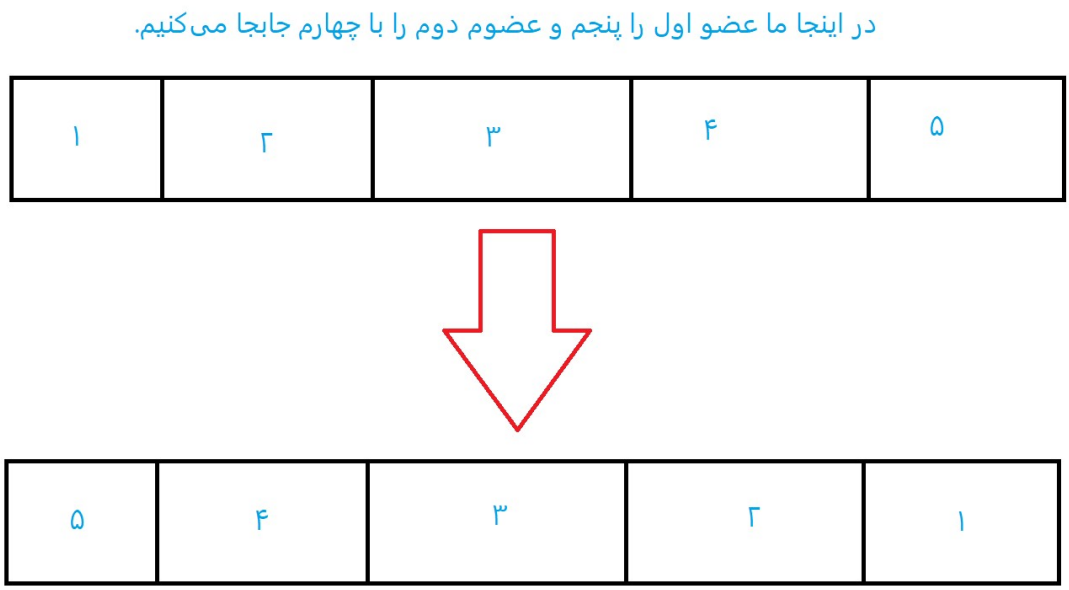
۱.۸.۴ Algorithm

اگر ما از اولین پارامتر ورودی شروع کنیم. تا به وسط آن‌ها برسیم. و آن‌ها را با نظر خودشان از ته جابه‌جا کنیم. این کار انجام خواهد شد.

برای تفهیم بیشتر مثالی کوچک را برای خود میزنیم.

فرض کنید تعداد ورودی‌های ما ۵ تا است. اگر ما اولی را پنجمی و دومی را با چهارمی و سومی را دست نزنیم. این کار انجام خواهد شد.

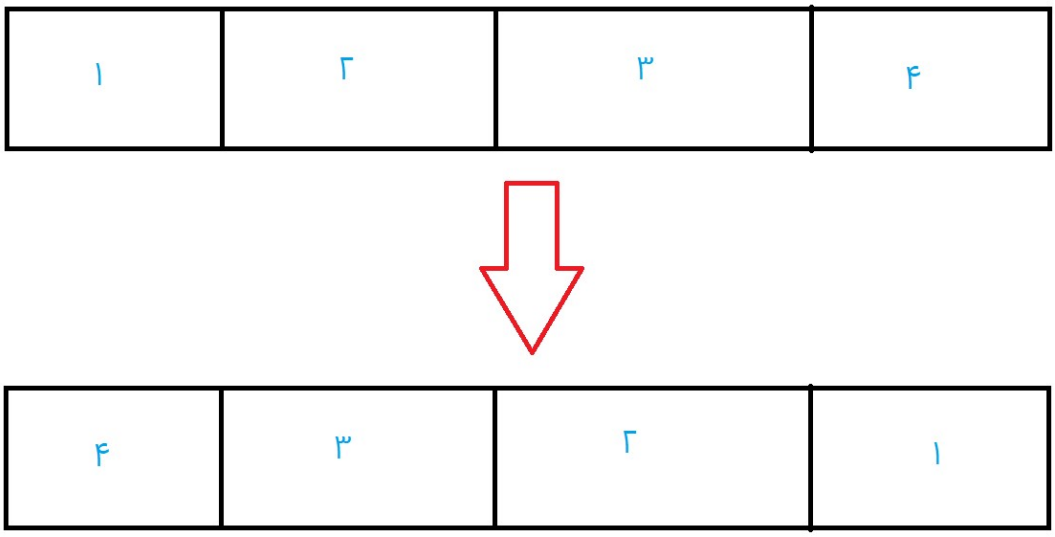
ما با ۵ ورودی این کار را دو بار انجام دادیم.



شکل ۹.۴: Reverse with Odd Number of elements

فرض کنید اکنون تعداد ورودی‌های ما ۴ تا است. اگر ما عضو اول را چهارم و عضو دوم را با عضو سوم جابجا کنیم. جواب را به دست خواهیم آورد.
در اینجا ما با چهار ورودی باید این کار را دو بار انجام دهیم.

کافی است. که چهارمی را با اولی و سومی را با دومی جابجا کنیم.



تمامی عضوها برعکس شده‌اند.

شکل ۱۰.۴: Reverse with Even Number of elements

پس با تست کردن عدد های کوچک می‌فهمیم. که اگر ما هر چه تعداد ورودی‌هایمان باشد. را بر ۲ تقسیم کرده. و به عدد قبلش گرد کنیم. تعداد بارهای جابجا کردن لازم را به ما خواهد داد.

Python ۲.۸.۴

```python
def reverse(l1):
    for idx in range(int(len(l1)/2)):
        l1[idx], l1[len(l1)-idx-1] = l1[len(l1)-idx-1], l1[idx]

def output_reverse(*args):
    args = list(args)
    Reverse(args)
    print(args)
```

نمونه کد ۵۰: Reverse and Print

int(len(l1)/2) باعث می‌شود. که اگر ۵ ورودی داشته باشیم. این کار دو بار انجام شده. و اگر ۴ ورودی داشته باشیم. نیز این کار دو بار انجام شود.

چرا ورودی‌های تابع output_reverse را به لیست تبدیل کردیم؟ زیرا tuple ها غیرقابل تغییر هستند.

: C# ۳.۸.۴

```csharp
static void Reverse<_Type>(_Type[] list)
{
    for (int i=0; i<(list.Count()/2); i++)
        (list[i], list[list.Count()-1-i]) = (list[list.Count()-1-i], list[i]);

}
static void OutputReverse<_Type>(params _Type[] args)
{
    Reverse(args);
    System.Console.WriteLine(string.Join(",", args));
```

نمونه کد ۵۱: Reverse and Print

توضیحات :

list.Count() اندازه آرایه را برگردانده که خروجی آن عددی از نوع عدد صحیح است. و با تقسیم بر ۲ اعداد مورد نظر را به ما می‌دهد.

۴.۸.۴ C++ :

```cpp
template<typename _Type>
void Reverse(vector<_Type>& v)
{
    for(int i=0; i<v.size()/2; i++)
    {
        auto hold = v[i];
        v[i] = v[v.size()-i-1];
        v[v.size()-i-1] = hold;
    }
}

template<typename _T, typename... _Type>
void OutputReverse(_Type... args)
{
    vector<_T> v1 = {args ...};
    Reverse(v1);
    for(string n : v1)
        cout << n << " ";
    cout << endl;
}
```

نمونه کد ۵۲: Reverse and Print

جلسه ۵

Class and Object

روزبه غزوی - ۱۳۹۸/۱۱/۲۸

یک کلاس مانند نقشه ای کامل از یک شی مشخص است. در جهان واقعی هر شی ایی دارای یک سری خصوصیات مانند رنگ ، شکل و نوع عملکرد است. برای مثال شما یک اتومبیل فراری را درنظر بگیرید. فراری یک شی از نوع اتومبیل است و اتومبیل در اینجا نقش کلاس را برای ما بازی میکند. یک کلاس اتومبیل میتواند دارای خصوصیات معینی مانند سرعت ، رنگ و شکل باشد.

بنابراین هر شرکت خودرو سازی که یک اتومبیل را با ویژگی های مورد نظرش تولید کند، شی ایی از یک اتومبیل را ساخته است. با این اوصاف اتومبیل های فراری ، لامبورگینی و کادیلاک همگی شی ایی از کلاس اتومبیل هستند.

در دنیای برنامه نویسی شی گرا ، یک کلاس دارای تعدادی مشخص فیلد ، صفت ، رویداد و متد است. یک کلاس انواع داده و عملکرد هایی که اشیا دارند را مشخص میکند. در یک کلاس میتوانید نوع مورد نظر خود را از طریق گروه بندی متغیر ها و دیگر انواع ایجاد کنید.

۱.۵ تعریف کلاس (class)

در زبان برنامه نویسی (سی شارپ) یک کلاس میتواند با استفاده از کلمه ی رزرو شده ی `class` تعریف شود:

```
1  public class Circle
2  {
3      // Fields & Properties & Methods & Events go here //
4  }
```

نمونه کد ۵۳: نمونه ای از یک کلاس

در نمونه مثال فوق قبل از کلمه ی class از کلمات رزرو شده ی سطوح دسترسی استفاده شده است. و چون در این مورد از کلمه ی public استفاده شده ، هر کسی میتواند شی ایی از این کلاس را ایجاد کند. به دنبال کلمه ی class نام دلخواه کلاس (Circle) قرار گرفته است. باقی مانده ی تعریف یک کلاس بدنه ی آن است که داده ها و رفتار های کلاس در آن تعریف میشود. فیلد ها ، صفات ، متد ها و رویداد ها در مجموع اعضای کلاس را تشکیل میدهند.

۲.۵ ایجاد یک شیء از کلاس

اگرچه یک شی (object) و کلاس در مواقعی به عنوان جایگزینی برای هم دیگر استفاده میشوند ، در واقع آنها دو چیز متفاوت هستند. یک کلاس نوع یک شی را مشخص میکند. گاهی اوقات از شی به عنوان نمونه ایی از یک کلاس یاد میشود. اشیا میتوانند با استفاده از کلمه ی رزرو شده ی new که به دنبال آن نام کلاس می آید تعریف شوند :

```
1  Circle object1 = new Circle();
2  Circle object2 = new Circle();
3  Circle object3 = new Circle();
```

نمونه کد ۵۴: نمونه ای از یک شیء

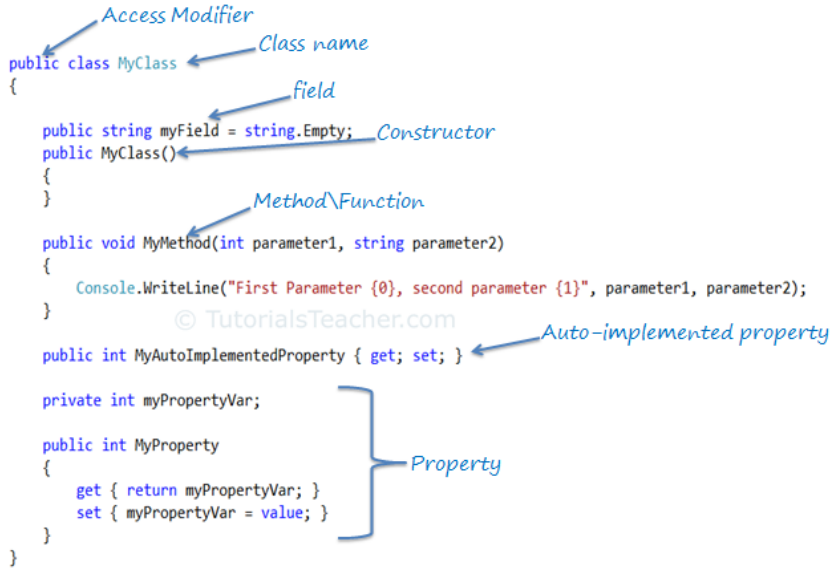
وقتی نمونه ای از یک کلاس ایجاد میشود ، ارجاع آن به یک شیء توسط برنامه نویس انجام میشود. در نمونه مثال قبل object۱ با مقداردهی به یک شیء از نوع Circle ارجاع پیدا کرده است.

۳.۵ بررسی یک نمونه مثال از کلاس

در نمونه مثال زیر کلاسی به نام `MyClass` ایجاد شده است که دارای فیلد ، صفت و متد است.

```
۱  public class MyClass
۲  {
۳      public string  myField = string.Empty;
۴
۵      public MyClass()
۶      {
۷      }
۸
۹      public void MyMethod(int parameter1, string parameter2)
۱۰     {
۱۱         Console.WriteLine(" First Parameter {0} , Second Parameter {1} ",
۱۲         parameter1, parameter2);
۱۳     }
۱۴
۱۵     public int MyAutoImplementedProperty { get; set; }
۱۶
۱۷     private int myPropertyVar;
۱۸
۱۹     public int MyProperty
۲۰     {
۲۱         get { return myPropertyVar; }
۲۲         set { myPropertyVar = value; }
۲۳     }
۲۴ }
```

نمونه کد ۵۵: مثالی از یک کلاس



در زیر به طور جداگانه به بررسی هر کدام از قسمت های مهم مثال فوق خواهیم پرداخت.

۴.۵ سطح دسترسی (Access Modifiers)

سطوح دسترسی ،کلمات رزرو شده ای هستند که بر روی اعلان یک کلاس ، متد ، صفت ، فیلد و دیگر اعضای یک کلاس میتوانند اعمال شوند.

کلمات رزرو شده برای سطوح دسترسی در زبان سی شارپ عبارت اند از :

- Public
- Private
- Protected
- Internal

این کلمات، چگونگی و سطح دسترسی یک کلاس و یا اعضای آن را در برنامه مشخص میکنند. برای آشنایی بیشتر با آنها به جدول زیر رجوع کنید.

کلمه ی کلیدی	عملکرد
public	کلمه ی public به هر قسمت از برنامه در همان اسمبلی و یا اسمبلی دیگر اجازه میدهد که به نوع و اعضای آن دسترسی پیدا کند.
private	کلمه ی private دسترسی قسمت های دیگر برنامه را به نوع و اعضای خود محدود میکند. تنها کد های همان کلاس و یا struct میتوانند به آن دسترسی پیدا کنند.
internal	کلمه ی internal به هر قسمت از برنامه در همان اسمبلی اجازه میدهد که به نوع و اعضای آن دسترسی پیدا کند
protected	کلمه ی protected به کد های برنامه در همان کلاس و یا کلاس هایی که از آن کلاس مشتق شده اند اجازه دسترسی به نوع و اعضای خود را میدهد.

۵.۵ فیلد (Field)

متغیری که در سطح یک کلاس تعریف میشود فیلد نامیده میشود. فیلد ها میتوانند مقادیری از یک نوع مشخص را در خود نگه دارند. عموما فیلد ها در کلاس دارای سطح دسترسی private (فقط قابل دسترسی در محدوده ی همان کلاس) هستند و در صفت ها (property) استفاده میشوند.

۶.۵ سازنده (Constructor)

یک کلاس میتواند دارای سازنده های پارامتر دار و یا بدون پارامتر باشد. سازنده ها در هنگام تعریف یک شی از یک کلاس فراخوانی میشوند. سازنده ها به وسیله ی یک کلمه ی سطح دسترسی و کلمه ای که همنام با نام کلاس باشند تعریف میشوند :

```
class MyClass
{
    public MyClass()
    {

    }
}
```

نمونه کد ۵۶: مثالی از کانستراکتر بدون پارامتر

```
class Product
{
        public ID Id;
        public string Name;
        public int Price;
        public double Rate;
        public Product(ID id , string name, int price, double rate)
        {

                this.Id=id;
                this.Name=name;
                this.Price=price;
                this.Rate=rate;
        }
}
```

نمونه کد ۵۷: مثالی از کانستراکتر پارامتر دار

در بدنه سازنده قصد داریم مقدار متغیری که از پارامتر ورودی سازنده دریافت کرده ایم درون متغیر نمونه کلاس بریزیم! به همین دلیل متغیر نمونه کلاس را با کلمه کلیدی this صدا میزنیم.

۷.۵ متد (Method)

یک متد در زبان برنامه نویسی سی شارپ میتواند به شکل الگوی زیر تعریف شود:

```
<access modifier> <return type> MethodName(param Type param Name)
```

```
1  public void MyMethod(int parameter1, string parameter2)
2  {
3       //  write your method code here .. //
4
5  }
```

نمونه کد ۵۸: مثالی از یک متد در کلاس

۸.۵ صفت (Property)

یک صفت میتواند با استفاده از کلمات رزرو شده ی get و set مانند نمونه کد زیر ایجاد شود :

```
1  private int myPropertyVar;
2
3  public int MyProperty
4  {
5      get { return myPropertyVar; }
6      set { myPropertyVar = value; }
7  }
```

نمونه کد ۵۹: مثالی از یک صفت

دقت داشته باشید که در یک صفت از یک فیلد استفاده میشود. در نمونه مثال بالا با توجه به تعریف صفت MyProperty ، هر گاه بخواهیم مقدار این صفت را بدست آوریم مقدار فیلد myPropertyVar به ما نشان داده میشود و هرگاه این صفت را مقدار دهی کنیم این مقدار در فیلد myPropertyVar قرار میگیرد.

عموما صفات در زبان سی شارپ برای سطح دسترسی public (قابل دسترسی در خارج از محدوده ی کلاس) هستند. به عبارت دیگر فیلد myPropertyVar در خارج از کلاس به طور غیر مستقیم از طریق صفت MyProperty قابل دسترسی است.

نکته : الزامی برای وجود هر دو کلمه ی رزرو شده ی get و set در تعریف یک صفت وجود ندارد. برای مثال اگر صفتی فقط دارای قسمت get باشد آن صفت فقط خواندنی است. حتی میتوان منطقی خاص را در قسمت های get و set برای یک صفت به کار برد.

```
private int myPropertyVar;

public int MyProperty
{
    get {
        return myPropertyVar / 2;
    }

    set {

        if (value > 100)
            myPropertyVar = 100;
        else
            myPropertyVar = value; ;
    }
}
```

نمونه کد ۶۰: مثالی از یک صفت

در نمونه مثال فوق هنگام خوانده شدن مقدار صفت، همیشه نیمی از فیلد مورد نظر، نشان داده میشود و در هنگام مقدار دهی نیز مقادیر بزرگتر از ۱۰۰ در فیلد مربوطه قرار نمیگیرد.

۹.۵ صفات Auto-implemented

از زمان انتشار سی شارپ نسخه ۳.۰ تعاریف صفات ساده تر شد. این برای زمانی است که نیاز به اعمال منطق خاصی در صفت خود نداریم.

نمونه مثال زیر یک صفت Auto-implemented را نشان میدهد :

```
public int MyAutoImplementedProperty { get; set; }
```

دقت داشته باشید که هیچ فیلدی برای این صفت تعریف نشده است. یک فیلد به صورت ضمنی بعدا توسط کامپایلر ایجاد شده و این نوع صفات را مدیریت میکند.

۱۰.۵ فضای نام (Namespace)

یک فضای نام مکانی برای قرار گیری کلاس ها و یا مجموعه ای از فضای نام هاست. فضای نام را میتوان نام منحصر به فردی دانست که کلاس های داخل خود را از دیگر کلاس ها متمایز میکند. در زبان سی شارپ ، فضای نام میتواند با استفاده از کلمه ی رزرو شده ی namespace تعریف شود :

```
1  namespace CSharpTutorials
2  {
3      class MyClass
4      {
5
6      }
7  }
```

نمونه کد ۶۱: نمونه ای از فضای نام

در نمونه مثال بالا نام کامل کلاس MyClass به این شکل است : .CSharpTutorials.MyClass

جلسه ۶

آرایه‌ها و کلاس

نیکی نزاکتی - ۱۳۹۸/۱۲/۳

جزوه جلسه ۶ام مورخ ۱۳۹۸/۱۲/۳ درس برنامه‌سازی پیشرفته تهیه شده توسط نیکی نزاکتی.

۱.۶ آرایه‌ها در C++

می‌دانیم که برای تعریف یک متغیر از نوع Integer به صورت زیر عمل میکنیم.

```
int grade=0;
```

با این کار به اندازه‌ی ۴ بایت از فضا برای متغیر grade از حافظه اشغال می‌شود.
حال اگر بخواهیم تعداد بیشتری متغیر از نوع دلخواه داشته باشیم و در عین حال لازم نباشد کد بالا را چندین بار تکرار کنیم از آرایه استفاده می‌کنیم.

- Static Array

```
int grades[20];
```

در آرایه‌های static تعداد متغیرهای آرایه هنگام compile شدن باید مشخص باشد و این تعداد در ادامه قابل تغییر نخواهد بود.

- Dynamic and Static Array

```
int count=0;
std::cin»count;
int* pGrades=new int[count];
```

این نوع تعریف آرایه از جهتی static و از جهتی dynamic است.از این جهت dynamic است که اندازه‌ی دلخواه توسط ورودی برای آن در نظر گرفته شده است که مقدار آن از قبل مشخص نبود و از این جهت static است که این اندازه پس از تعیین شدن قابل تغییر نیست.
پس از اینکه استفاده‌ی ما از این آرایه به اتمام رسید با دستور زیر حافظه اشغال شده توسط آرایه را آزاد می‌کنیم تا از memory leak جلوگیری شود:

```
delete[] pGrades;
```

واضح است که با توجه به غیر قابل تغییر بودن اندازه‌ی آرایه pGrades اگر به عنوان مثال اندازه‌ی آن ۲۰ باشد،عضو۲۱ به آرایه قابل اضافه کردن نخواهد بود.

- Vector

```
#include<vector>
std::Vector<int> vGrades;
```

کلاس vector از کتابخانه stl است.در اینجا اندازه‌ای برای vGrades در نظر گرفته شده است که مقدار آن مشخص نیست و نمی‌توان به عنوان مثال به خانه ۵ام آن دسترسی داشت. برای اینکار باید آن را مقدار دهی کرد.

```
vGrades.push_back(1);
vGrades.push_back(3);
vGrades.push_back(2);
```

یا در هنگام تعریف اولیه به آن مقدار دهی کرد:

std::Vector<int> vGrades={1,3,2};

بعد از مقدار دهی می‌توان به خانه‌های آن دسترسی داشت.

vGrades[2]=5;

int a=vGrades[1];

int b=vGrades.at(3);

اگر اندازه‌ی یک vector به عنوان مثل ۲۰ باشد، هنگامی که ۲۱امین عضو به آن اضافه می‌شود، یک vector جدید به اندازه ۲۰x۲ ساخته می‌شود که ۲۰ عضو vector قبلی را کپی کرده و در آن می‌ریزد،vector قبلی را پاک کرده و عضو ۲۱ام را به این vector جدید اضافه می‌کند.

متغیر vGrades روی stack قرار دارد ولی مقادیر آن در Heap ذخیره شده‌است.در نتیجه برای استفاده از آن در متدها باید از refrence آن استفاده کرد:

Void Sort(std::vector<int> &vGrades)

در غیر اینصورت vGrades را کپی کرده و در vector جدیدی می‌ریزد.

اگر بخواهیم در حین استفاده از vector از طریق refrence غیر قابل تغییر باشد از const استفاده می‌کنیم:

```
۱ Void Print(const std::vector<int> &vGrades)
۲ {
۳     for(int n:vGrades)
۴         std::cout<<n;
۵ }
```

نمونه کد ۶۲: تابع چاپ همه اعضای یک vector

۲.۶ آرایه‌ها در Java

در زبان جاوا آرایه‌ها روی stack قرار ندارند و روی allocate heap می‌شوند.

- Dynamic and Static Array

int count=5;

int[] gradeList = new int[count];

آرایه‌ی بالا معادل *int در زبان c++ است.

- ArrayList

```
ArrayList<Double>gradeList = new ArrayList<Double>();
```

```
import java.util.ArrayList;
```

از دستور زیر برای اضافه کردن عضو به ArrayList استفاده می‌کنیم:

```
gradeList.add(5.1);
gradeList.add(6.0);
gradeList.add(7.2);
gradeList.add(4.8);
```

دستور زیر مقدار عدد a را برابر عضوی از gradeList قرار می‌دهد که index آن ۳ است:

```
double a = gradeList.get(3);
```

در نتیجه مقدار a برابر ۴/۸ خواهد بود. دستور زیر عضوی از gradeList را که index آن ۱ است را از gradeList حذف کرده و مقدار آن را بر می‌گرداند:

```
double d = gradeList.remove(1);
```

مقدار d برابر ۶/۰ است.

۳.۶ آرایه‌ها در Python

آرایه‌ها در python از همه نظر dynamic هستند.

```
list=[]
```

تعریف یک آرایه به صورت بالا انجام می‌شود که مشود در ابتدا به آن مقدار داد و یا مقدار دهی به صورت زیر انجام شود:

```
list.append(5)
```

به این ترتیب مقدار ۵ به لیست اضافه می‌شود. دستور زیر اولین عضو list را که مقدار آن ۵ است از list حذف می‌کند:

list.remove(5)

۴.۶ آرایه‌ها در C#

- Dynamic and Static Array

int count=5;

int[] list = new int[count];

- List

using System.collection.Generic;

List<int> myList = new List<int>();

برای اضافه کردن عضو به List از دستور زیر استفاده میکنیم:

myList.Add(5);

با دستور بالا عدد ۵ به انتهای myList اضافه می‌شود. برای دسترسی به اعضای آن می‌توان از index آن‌ها استفاده کرد:

int a=myList[0];

مقدار عدد a برابر با عضوی از myList با index ۰ یعنی عدد ۵ خواهد بود.

- 2 Dimensional Array

int[,] my2dArray = new int [5,3];

و یا:

int[] myjaggedArray = new int[5][];

که برای تعیین کردن بعد دوم myjaddegArray به صورت زیر عمل می‌کنیم:

```
for(int i=0; i<myjaggedArray.Length; i++)
{
    myjaggedArray[i] = new int[2];
}
```

به این ترتیب یک آرایه ۵ عضوی داریم که هر عضو آن یک آرایه ۲ عضوی است.

۵.۶ کلاس در C++

برای تعریف کلاس در c++ در فایل جداگانه گاهی به این صورت عمل می‌کنیم که تعریف کلاس را در فایلی با پسوند h. تعریف می‌کنیم و پیاده‌سازی آن را در فایلی دیگر با پسوند hpp. انجام می‌دهیم.
برای کلاس های کوچک تر مانند این مثال، تعریف و پیاده سازی کلاس هر دو در یک فایل با پسوند h. انجام شده است. نام فایل ها باید با نام کلاس یکی باشد.
برای تعریف کلاس مانند زیر ابتدا class را نوشته و سپس نام آن را می‌نویسیم. کد های مربوط به کلاس داخل {} نوشته می‌شود و در آخر آن ؛ قرار می‌گیرد. در c++ کدها به صورت default به حالت private قرار دارند و بیرون از کلاس نمی‌توان به آن ها دسترسی داشت. برای قابل دسترس بودن کد ها باید در قسمت public نوشته شود:

class Course

}

};

در کلاس متغیر های کلاس و constructor آن را تعریف می‌کنیم:

```
#include<string>
#include "Instructor.hpp"

using namespace std;


class Course
{
    string m_Title;
    string m_InstructorName;
    string m_InstructorDegree
    int m_Credits;

public:
    Course(string title, string instName, string instDegree, int credits)
        : m_Title(title)
        , m_InstructorName(instName)
        , m_InstructorDegree(instDegree)
        , m_Credits(credits)
    {}
};
```

نمونه کد ۶۳: کلاس در c++

constructor مشخص می‌کند که برای تعریف یک کلاس چه مقادیری باید به آن داده شود تا یک object از نوع آن کلاس تعریف و ساخته شود. constructor را می‌توان به صورت زیر با استفاده از <- که

یک pointer است، تعریف کرد:

```
Course(string title, string instName, string instDegree, int credits)
{
    this->m_Title = title;
    this->m_InstructornName = instName;
    this->m_InstructorDegree = instDegree;
    this->m_Credits = credits;
}
```

برای دسترسی به کلاس در فایل های دیگر باید از "#include"ClassName استفاده کرد.
کلاس می‌تواند دارای یک object از یک کلاس دیگر باشد:

```
#include <string>
using namespace std;

class Instructor
{
    string m_Name;
    string m_Degree;
    double m_Rating;

public:
    Instructor(string name, string degree, double rating)
        : m_Name(name)
        , m_Degree(degree)
        , m_Rating(rating)
        {}
};
```

متدهای کلاس مانند دیگر متد ها تعریف می‌شوند:

```
 string GetInfo()
    {
        return m_Name + " " + m_Degree;
    }
```

و برای استفادت از کلاس Instructor در کلاس Course به صورت زیر عمل می‌کنیم:

```cpp
#include<string>
#include "Instructor.hpp"

using namespace std;


class Course
{
    string m_Title;
    Instructor m_Instructor;
    int m_Credits;

public:
    Course(string title, string instName, string instDegree, int credits)
        : m_Title(title)
        , m_Instructor(instName, instDegree, 5.3)
        , m_Credits(credits)
    {}

    string GetCourseInfo()
    {
       string result;
       result += m_Title;
       result += "\n";
       result += m_Instructor.GetInfo();
       return result;
    }
};
```

در اینجا m_Instructor object از کلاس Instructor است.

متدی برای این کلاس به صورت زیر تعریف می‌کنیم که اطلاعات مورد نظر کلاس را به صورت یک string باز گرداند:

```cpp
 string GetCourseInfo()
    {
       string result;
       result += m_Title;
       result += "\n";
       result += m_Instructor.GetInfo();
       return result;
    }
```

جلسه ۷

تفاوت بین رفرنس تایپ با ولیو تایپ در سی پلاس پلاس ،جاوا و سی شارپ

امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

جزوه جلسه ۷ام مورخ ۱۳۹۸/۱۲/۰۵ درس برنامه‌سازی پیشرفته تهیه شده توسط امیرحسین سماوات. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

۱.۷ Reference Type و Value Type

سی شارپ نوع داده ها را بسته به چگونگی ذخیره مقادیرشان در حافظه به دو دسته تقسیم میکند :

- Value Type
- Reference Type

۲.۷ Value Type

به نوعی از داده Value Type گفته میشود که یک مقدار را در فضای حافظه ی خود ذخیره کند. و این به این معناست که متغیر هایی که از نوع این داده نوع تعریف میشوند به طور مستقیم دارای مقداری در خود هستند.

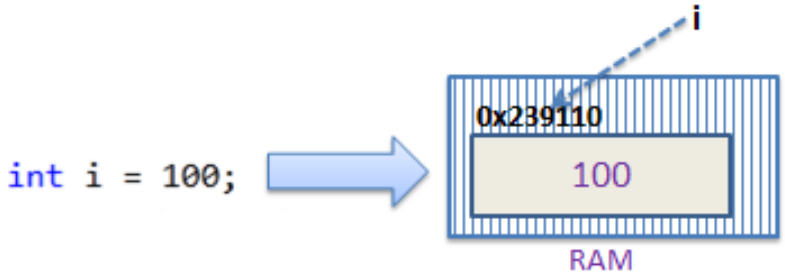
نکته : تمام Type Value ها از فضای نام System.ValueType استفاده میکنند که آن فضای نام هم در فضای نام System.Object قرار دارد.

برای مثال متغیری از نوع int را در نظر بگیرید :

```
۱  int i=100;
```

سیستم مقدار عدد صحیح ۱۰۰ را در فضای حافظه ای که برای متغیر "i" تخصیص داده شده است ، ذخیره می کند.

تصویر زیر نحوه ی ذخیره سازی مقدار ۱۰۰ را در حافظه به آدرس (۰x۲۳۹۱۱۰) برای متغیر "i" نشان میدهد :



همه ی داده نوع هایی که در زیر آورده شده است از نوع value type هستند :

double	decimal	char	byte	bool
sbyte	long	int	float	enum
ushort	ulong	unit	sbyte	short

۳.۷ ارسال با مقدار Pass by Value

وقتی یک متغیر از نوع Value type را به عنوان آرگومان برای یک متد ارسال می کنید ، سیستم یک کپی جداگانه از آن متغیر را ایجاد کرده و آن را برای متد ارسال میکند. بنابراین اگر تغییری در مقدار آن متغیر در متد مربوطه ایجاد شود تاثیری بر مقدار اصلی آن ندارد.

نمونه مثال زیر نحوه ی عملکرد روش ارسال با مقدار را نشان میدهد :

```
۱  static void ValueChange(int x)
۲  {
۳      x =  200;
۴
۵      System.Console.WriteLine(x);
۶  }
۷
۸  static void Main(string[] args)
۹  {
۱۰     int i = 300;
۱۱
۱۲     System.Console.WriteLine(i);
۱۳
۱۴     ValueChange(i);
۱۵
۱۶     System.Console.WriteLine(i);
۱۷ }
```

در نمونه مثال فوق، "i" متغیری است که در متد Main تعریف و مقدار دهی شده است. متغیر "i" را به متدی به نام ()ChangeValue ارسال میکنیم (ارسال با مقدار). با وجود اینکه مقدار این متغیر در متد ()ChangeValue تغییر میکند ولی به خاطر اینکه روش ارسالی ، ارسال با مقدار است یک کپی از متغیر "i" برای متد ارسال شده است و تغییر در آن هیچ تاثیری برای مقدار اولیه ی متغیر "i" ندارد. با چاپ مقدار

متغیر "i" در انتهای برنامه به وضوح میتوان دید هیچ تغییری در مقدار آن ایجاد نشده است :

```
100
200
100
```

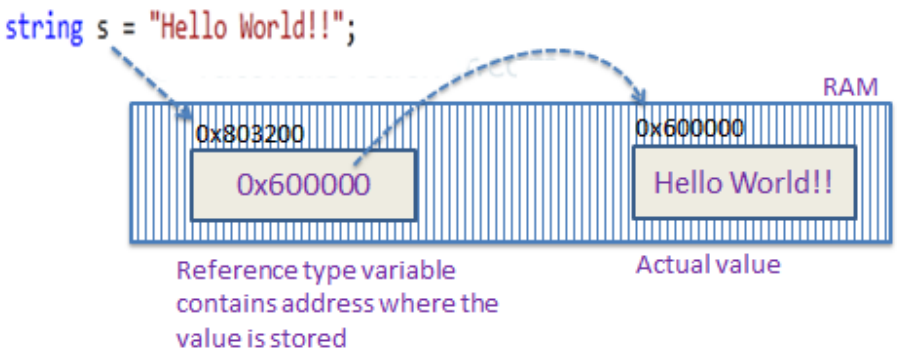
۴.۷ Reference Type

برخلاف Value Type ها ، Reference Type ها مقادیرشان را به صورت مستقیم در خود ذخیره نمی کنند. در عوض آنها آدرس مکانی از حافظه را که مقدار در آن قرارگرفته است، در خود ذخیره میکنند. به عبارت دیگر Reference Type ها شامل یک اشاره گر هستند که به مکانی دیگر از حافظه اشاره میکند که داده یا مقدار در آن ذخیره شده است.

برای این حالت یک متغیر رشته ای را میتوان مثال برد :

```
string s = " Hello Worlds ";
```

تصویر زیر چگونگی تخصیص حافظه را برای متغیر رشته ای بالا نشان میدهد :



همانطور که در تصویر بالا مشاهده می کنید سیستم یک مکان تصادفی در حافظه (۰x۸۰۳۲۰۰) را برای متغیر "S" انتخاب کرده است. مقداری که در متغیر "S" قرار میگیرد ۰x۶۰۰۰۰۰ است که آدرس خانه ای از حافظه است که مقدار اصلی یعنی !! Hello World در آن قرارگرفته است.

داده نوع های زیر همگی Reference Type هستند :

- String
- تمام آرایه ، حتی اگر مقادیر آنها از نوع Value Type باشد
- Classes
- Delegates

۵.۷ Pass by Reference ارسال با ارجاع

وقتی یک متغیر Reference Type را به عنوان آرگومان از یک متد به متد دیگری میفرستید دیگر کپی ایی از آن ساخته نمیشود. در عوض آدرس آن متغیر به متد مربطه ارسال میشود. نمونه مثال زیر روش ارسال با ارجاع را نشان میدهد :

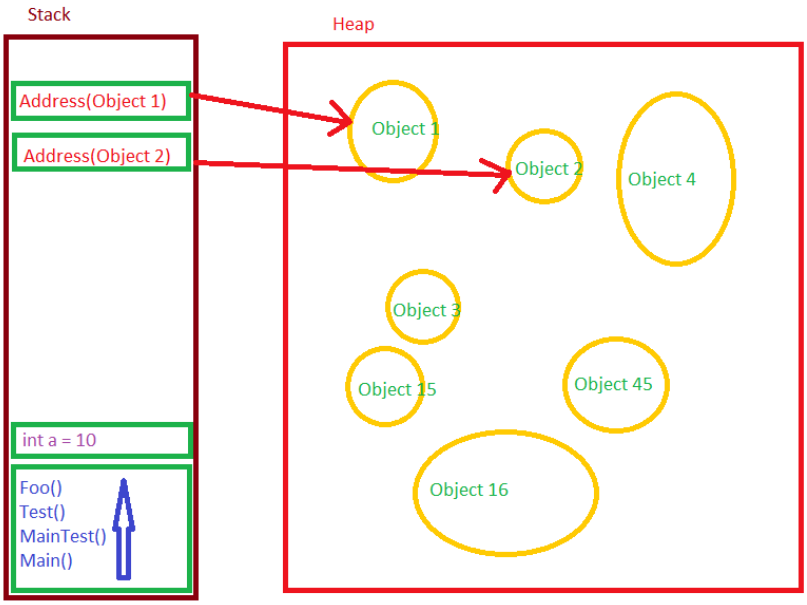
```
۱  static void ChangeReferenceType(Student std2)
۲  {
۳      std2.StudentName = "Steve";
۴  }
۵
۶  static void Main(string[] args)
۷  {
۸      Student std1 = new Student();
۹      std1.StudentName = "Bill";
۱۰
۱۱     ChangeReferenceType(std1);
۱۲
۱۳     Console.WriteLine(std1.StudentName);
۱۴ }
```

در نمونه مثال فوق ، از آنجایی که Student یک کلاس (class) است هنگامی که شی ایی از کلاس Student به نام std۱ را به عنوان آرگومان به متد ChangeReferenceType() ارسال می کنیم ، آن چیزی که در عمل ارسال میشود آدرس حافظه ی شی std۱ است. بنابراین وقتی متد ChangeReferenceType() فیلد StudentName را تغییر میدهد ، مقدار اصلی فیلد StudentName از شی std۱ را تغییر میدهد. به همین دلیل شی std۱ و آرگومان std۲ هر دو به یک آدرس در حافظه اشاره میکنند. بنابراین این خروجی برابر با رشته ی "steve" است

۶.۷ Stack و Heap

وقتی که یک متد را صدا می زنیم، پارامترها و متغیرهای محلی آن،نیاز به حافظه دارند و این حافظه همیشه از `Stack` تامین می شود. سپس وقتی کار به متد تمام شد (یا به خاطر Exception) این حافظه به `Stack` به صورت خودکار بازگردانده میشود. وقتی یک نمونه‌ای از یک کلاس را ایجاد می کنیم (که از کلمه `new` کردن استفاده می کنیم) برای این نیز یک حافظه نیاز است که از `Heap` استفاده میشود. وقتی کار تمام می شود به `Heap` باز گردانده نمی شود.

```
۱  public void Main()
۲  {
۳      Stack //
۴      int x = 2;
۵      Heap //
۶      MyClass ob = new MyClass();
۷  }
```



خب

حالا در عکس بالا میبینید که object ۱ و object ۲ وصل هستن و یه تعدادی اطلاعات هستن داخل `Heap` که به چیزی وصل نیستن. کار `Garbage Collection` این هستش که بیادش این موارد رو پاک کنه تا حافظه شما خالی بشه.

cpp ۱.۶.۷

```cpp
#include <iostream>
using namespace std;
void func(int a)
{
    a++;
}
int main()
{
    int a=2;
    func(a);
    cout<<a;
}
```

خروجی این برنامه ۲ ه چون شما a رو بصورت مقداری یا با value به func فرستادی یعنی a ای که داخل تابع هست با a داخل main فرق می کنه .

ولی حالا اگر بخوایم a ای ک داخل تابع هست همون a داخل main باشه چی ؟!

این جور مواقع به جای فرستادن مقدار a باید ادرس حافظه ای که داخلش a ذخیره شده رو بفرستیم یعنی به این شکل

```cpp
#include <iostream>
using namespace std;
void func(int *a)
{
    (*a)++;
}
int main()
{
    int a=2;
    func(&a);
    cout<<a;
}
```

این جا مقداری که داخل خروجی چاپ میشه ۳ ه یعنی این دفعه خود a رو فرستادیم به تابع نه مقدارش رو ..

به عنوان مثال در

```
int *a=&b
```

یعنی ادرس b رو بزار در a

```
int &b=a
```

یعنی با رفرنس اینجا کار داریم.

پس مثال بالا رو میشه به این شکل هم نوشت بهتر هم هست به همین شکل نوشته بشه چون اشتباهات ناخواسته رو کم می کنه .

```
#include <iostream>
using namespace std;
void func(int& a)
{
    a++;
}
int main()
{
    int a=2;
    func(a);
    cout<<a;
}
```

حالا سوالی که پیش میاد اینه که چرا refrence by Call کنیم اصلا مگه نمیشه کد رو به این شکل هم نوشت ؟ چرا ولی مشکلش اینه که از نظر سرعت اجرا و مصرفه حافظه اصلا بهینه نیست چون موقع ارسال متغیر به تابع یک بار متغیر کپی میشه و همین طور موقع return دَن متغیر باز یک بار دیگه مقدار برگشتی کپی میشه تو a این کپی شدن ممکنه برای int زیاد فرقی نکنه ولی برای یک class که مثلا ۱۰۰ تا فیلد داره خیلی به چشم میاد . از اون طرف موقع ارسال پارامتر به متغیر و زمانی که برنامه داخل تابع هستش ۲ تا کپی از متغیر تو حافظه داریم که اینم یعنی مصرف حافظه ۲ برابر بیشتر از اونی که نیازه !

۲.۶.۷ جمع بندی:

شما وقتی که با `refrence` متغیر رو به تابع می فرستین گاهی اوقات ممکنه چند تا مشکل پیش بیاد : در صورت تغییر دادن متغیر به صورت اشتباه داخل تابع خود متغیر هم عوض میشه (برای حل این مشکل میشه با `const &` هم فرستاد متغیر رو)

در صورتی که شما در حال نوشتن برنامه به شکل موازی باشین و قرار باشه که تابع به شکل موازی با جایی که صدا زده شده اجرا بشه در صورت فرستادن متغیر با `refrence` اگر متغیر از محلی که اون جا صدا زده شده پاک بشه برنامه به مشکل بر می خوره .

بعضی وقت ها نیاز دارین که کپی انجام بشه این جا میشه هم با `refrence` فرستاد هم با value ولی فرستادن با مقدار این جور جا ها سریع تره چون به کامپایلر اجازه optimize کردن کد رو میده .

۷.۷ Java in Class

مفهموم کلاس:

کلاس به مجموعه کدی گویند که برای یک هدف نوشته شده اند و درکنار یکدیگر قرار گرفته اند.

شکل کلی کلاس:

```
Public class Classname{

    protected int id;
    public String text;
    private double spt;


    public Classname(int id,String text) {
        this.id = id;
        this.text=text;
    }

    public String functionName(){
        String data = AP" "Learning;
        return data;
    }
public class Main {
    public static void main(String[] args) {
        Classname c = new Classname(4,"AP");
        c.functionName();
    }
}
}
```

به صورت قراردادی اسامی کلاس‌ها رو با حروف بزرگ شروع می‌کنیم . مثال بالا رو که دقت کرده باشید، یه جاش نوشته `this.id` . کلا داخل هر کلاس، بخوایم به اجزای اون کلاس اشاره کنیم، می‌تونیم از این کلمه کلیدی استفاده کنیم.

مرسی که برای خواندن این جزوه وقت گذاشتید موفق باشید.

جلسه ۸

کار با فایل در سی‌شارپ

بابک بهکام کیا - ۱۳۹۸/۱۲/۱۰

در این جلسه نحوه کار با فایل تدریس شد و در پایان آن تمرین Phone Book داده شد.

۱.۸ آشنایی با فایل

ایتدا با چند ویژگی فایل آشنا می شویم (نمونه کد ۶۴).

```
class Program
{
    static void Main(string[] args)
    {
        string stdid = Console.ReadLine();
        File.WriteAllText("stdid.txt",stdid+"\n")
    }
}
```

نمونه کد ۶۴: مثالی برای `File.WriteAllText`

(نمونه کد ۶۵).

```
class Program
{
    static void Main(string[] args)
    {
        string fileName = "stdid.txt";
        string content = File.ReadAllText(fileName);
        System.Console.WriteLine(content);
    }
}
```

نمونه کد ۶۵: مثالی برای `File.ReadAllText`

۲.۸ برنامه ثبت‌نام دانش‌آموزان

۱.۲.۸ انتخاب اسم فایل

قبل از همه چیز اسم فایلی که قرار است با آن کار کنیم را انتخاب می کنیم. (نمونه کد ۶۶).

```
public const string StorageFileName = "students.csv";
```

نمونه کد ۶۶: انتخاب اسم فایل

۲.۲.۸ پیاده سازی اولیه

حال برنامه ای می نویسیم که بتواند ورودی ار نوع string بگیرد که این ورودی یکی از `add,list,find` باشد که هر کدام متدی را صدا بزند

و اگر کاربر چیز دیگری به عنوان ورودی بدهد در خروجی پیغامی برایش چاپ شود. (نمونه کد ۶۷).

```
class Program
{
    static void Main(string[] args)
    {
        if (args.Length != 1 )
        {
            Usage();
            return;
        }

        if (args[0] == "add")
            AddStudent();
        else if (args[0] == "list" )
            PrintStudents();
        else if (args[0] == "find")
            FindStudent(args);
        else
            Usage();
    }
}
```

نمونه کد ۶۷: پیاده سازی `add,list,find`

(نمونه کد ۶۸).

```
private static void Usage()
{
    System.Console.WriteLine(" students. register to used be can program This ");
    System.Console.WriteLine(" follows: as is syntax usage The ");
    System.Console.WriteLine(" [searchstring] add|list|find cs.exe ");
}
```

نمونه کد ۶۸: پیاده سازی متد `usage()`

۳.۲.۸ AddStudent()

پیاده سازی متد `AddStudent()` . این متد زمانی صدا زده می شود که کلمه `add` به عنوان ورودی وارد شود.

(نمونه کد ۶۹).

```
private static void AddStudent()
{
    Console.Write(" "Id?);
    string id = Console.ReadLine();
    Console.Write(" "Name?);
    string name = Console.ReadLine();
    File.AppendAllText(StorageFileName, " {id} , {name}  \n ");
}
```

نمونه کد ۶۹: پیاده سازی متد `AddStudent()`

۴.۲.۸ PrintStudents()

پیاده سازی متد `PrintStudents()` . این متد زمانی صدا زده می شود که کلمه `list` به عنوان ورودی وارد شود.

(نمونه کد ۷۰).

```
private static void PrintStudents()
{
    var lines = File.ReadAllLines(StorageFileName);
    foreach(var line in lines)
        System.Console.WriteLine(line);
}
```

نمونه کد ۷۰: پیاده سازی متد `PrintStudents()`

حال می خواهیم `find` را پیاده سازی بکنیم به صورتی که کاربر موقع ران کردن علاوه بر تایپ کلید `find` باید اسم دانش‌آموزی را نیز وارد کند. و اگر چنین دانش‌آموزی وجود داشت اسم و شماره دانشجوییش را در خروجی تایپ کند. در صورتی که این دانش‌آموز وجود نداشت `not found` را در خروجی پرینت کند. بنابراین طول `args` باید ۲ باشد و در غیر این صورت متد `usage()` صدا زده شود. پس تغییر کوچکی در کد قبلی باید داشته باشیم.

(نمونه کد ۷۱).

```
class Program
{
    static void Main(string[] args)
    {
        if (args.Length != 1 && args.Length != 2)
        {
            Usage();
            return;
        }

        if (args[0] == "add")
            AddStudent();
        else if (args[0] == "list")
            PrintStudents();
        else if (args[0] == "find")
            FindStudent(args);
        else
            Usage();
    }
}
```

نمونه کد ۷۱:

FindStudent() ۵.۲.۸

در ادامه متد `FindStudent();` را پیاده سازی می کنیم

(نمونه کد ۷۲).

```
private static bool FindStudent(string[] args)
{
    if (args.Length != 2)
        return false;

    string searchKey = args[1].ToLower();

    string[] lines = File.ReadAllLines(StorageFileName);
    bool found = false;
    foreach(string line in lines)
    {
        string[] tokens = line.ToLower().Split(',');
        string id = tokens[0];
        string name = tokens[1];
        if (name == searchKey)
        {
            Console.WriteLine(" {id} : {name} ");
            found = true;
        }
    }

    if (! found)
        Console.WriteLine(" Found! Not ");

    return true;
}
```

نمونه کد ۷۲: پیاده سازی متد `FindStudent();`

در کد بالا سطرهای فایل را داخل آرایه‌ای می ریزیم. در این صورت آرایه ای داریم که هر عضو آن از یک اسم و شماره دانشجویی تشکیل شده است. به کمک `Split()` اسم و شماره دانشجویی هر عضو را از هم جدا می کنیم سپس اگر اسمی با اسمی که کاربر وارد کرده بود یکی باشد آن اسم و شماره دانشجویی متناظرش در خروجی چاپ می شوند و اگر اسمی پیدا نشود در خروجی `Not Found!` چاپ خواهد شد. برای اینکه بزرگی و کوچکی حروف ایرادی در کار ما بوجود نیاورند قبل از مقایسه ۲ string با کمک `ToLower()` حروف یزرگ هر دو آن ها را به حروف کوچک تبدیل می کنیم

جلسه ۹

شی گرایی و استاتیک*

محمدجواد مهدی‌تبار - ۱۳۹۸/۱۲/۱۲

در این جلسه کار با فایل با مبحث شی گرایی ادغام شده است.

۱.۹ اهداف اصلی این جلسه

- تبدیل کردن یک شی از کلاس به رشته و بالعکس [†]
- تابع های استاتیک و غیر استاتیک
- عمومی یا خصوصی بودن اعضای کلاس نظیر متد ها و ویژگی های کلاس [‡]

۲.۹ کلمه های کلیدی مهم

- `args`
- `statics`

object oriented and statics*
deserialize and serialize†
public or private methods or member variable‡

- `private and public`
- `serialize and deserialize`

۱.۲.۹ args

`args` مخفف کلمه arguments می باشد که که آرگمان های خط فرمان [§] را نشان می دهد. که در واقع پارامتر متد اصلی [¶] است. تنها وروی که متد اصلی می‌گیرد `args` می‌باشد و یا می‌توان به آن ورودی نداد.

```
1 static void Main(string[] args)
2 {
3 }
```

نمونه کد ۷۳: متد اصلی

همانطور که در نمونه کد ۷۳ می بینید `args` از نوع آرایه ای از رشته [‡] است . بعد از `build` کردن برنامه و قبل از اجرا شدن برنامه می توان به عنوان پارامتر به متد اصلی داد. در واقع تنها متدی که در طول برنامه اجرا می‌شود متد اصلی است.

نحوه ی استفاده از args

بعد از `build` کردن برنامه با دستور dotnet build می توان با دستور dotnet run و یا فایل `build` شده برنامه با فرمت exe بعد آن‌ها هر کلمه ای نوشته بشود جزء `args` محسوب می‌شود. و کاراکتر `space` جداکننده عنصر های داخل آرایه است. برای مثال

```
dotnet build
dotnet run first second
```

args[0] = first , args[1] = second

برای واضح تر شدن مطلب می‌توانید از مستند ها استفاده کنید .

- microsoft doc 1

§ command line
¶ Main
‡ string array

- microsoft doc 2
- dotnet perl

۲.۲.۹ statics

`static` در `csharp` به معنی این است که به `type` متعلق است و نه به یک شی خاص .
`static` را در موارد زیر می‌توان به کار برد .

- `class`
- `variable`
- `methods`
- `constructor`
- `struct`

وقتی از فیلد `static` برای کلاس استفاده می‌کنیم یعنی دیگر نمی‌توانیم از آن کلاس شی بسازیم . در واقع برای دسترسی به متد ها و متغیر های آن کلاس باید از دستور،
`<class-name.variable > or <class-name.method >` استفاده‌کنیم. در واقع با عملگر `dot` (.) این کار ممکن است.
اگر از فیلد `static` برای کلاس استفاده کنیم تمام اعضای آن کلاس هم باید `static` باشند.

```
static class Example
{
    static int Id;
    static void Main(string[] args)
    {
        Example ex = new Example();
        above line is incorrect , static class cannot be instantiated //
        int X = Example.Id;
    }
}
```

نمونه کد ۷۴: کلاس استاتیک

```
public class Example
{
    public static int X ;
    public int Y ;
    method static //
    public static void print()
    {
        Console.WriteLine( Example.X);
    }

}
public class Program
{
    static void Main(string[] args)
    {
        Example.X = 5 ;
        Example.print();
   }
}
```

نمونه کد ۷۵: متد استاتیک

با توجه به کد نمونه کد ۷۵ از `Y` فقط در صورتی می‌توان استفاده کرد که یک شی از نوع کلاس `Example` داشته باشیم. در واقع در متد `static` ، `this.` وجود ندارد. برای دسترسی به آن متد باید `<class-name.method >` را استفاده کرد .

مواقعی که از `static` برای متد ها استفاده می‌کنیم یعنی آن متد به متعلق به شئ نیست و برای `class` می‌باشد.

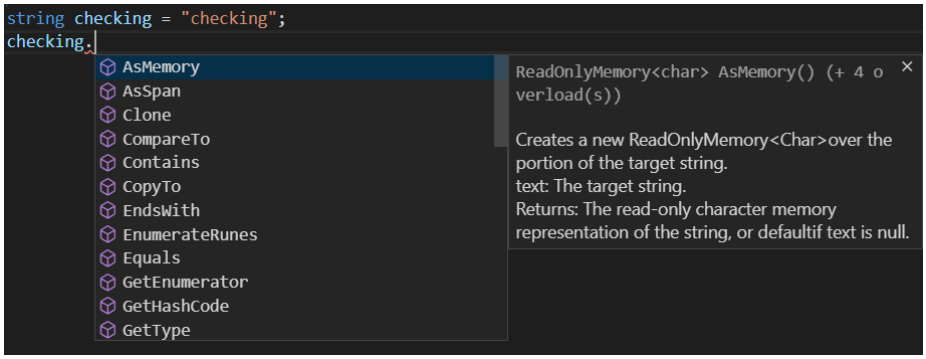
static and non-static method

static method

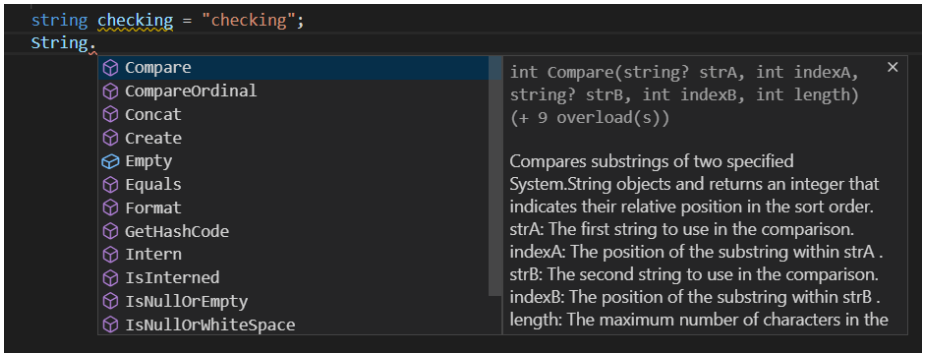
درکل هر وقت از دستور `<class-name.>` استفاده‌کنیم پیشنهاد هایی که از `auto complete` می‌آید همه ی آن‌ها متد های `static` هستند.

non-static method

هر وقت یک شی از یک کلاس درست می‌کنیم و برای آن شی از `dot` استفاده می‌کنیم پیشنهاد هایی که از `auto complete` می آیند `non-static` هستند.



شکل ۱.۹: String class non-static methods



شکل ۲.۹: String class static methods

برای درک بیشتر مفهوم `static` و کارایی آن وبسایت های زیر مفید هستند.

- microsoft doc
- GeeksforGeeks
- tutorials teacher

۳.۲.۹ public or private

`public` به معنی قابل دسترس بودن آن `method` یا `variable` در تمام فضای برنامه است.
`private` تنها در فضای `class` یا `struct` قابل دسترس است و خارج از این‌ها نمی‌توان از آن `method` یا `variable` استفاده کرد.

مواقعی که باید از `public` استفاده کرد

از `public` مواقعی استفاده می‌کنیم که از همه جای برنامه بخواهیم به آن `method` یا `variable` دسترسی داشته‌باشیم. و تنها استفاده مان از آنها فقط در داخل `class` نباشد.
** باید توجه داشته باشیم با استفاده از فیلد `public` کاربر نیز می‌تواند به آن دسترسی داشته باشد و به راحتی آن را تغییر دهد.

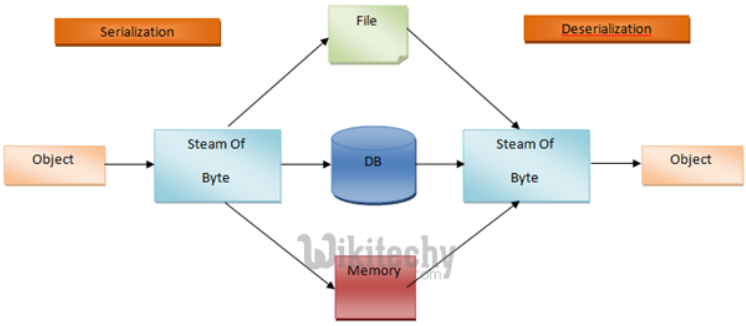
مواقعی که باید از `private` استفاده کرد

در واقع در طول برنامه باید از `private` استفاده کرد مگر در موارد ذکر شده بالا . مزایای `private` نسبت به `public` در این است که فقط نویسنده کد به آنها دسترسی دارد و در `class` های دیگر قابل دسترسی نیست . اگر از `method` یا `variable` فقط بخواهیم درون کلاس استفاده کنیم و جاهای دیگر کارایی ندارد بهتر است آن را `private` کنیم.
پس فرق بین `public or private` در امنیت ، دسترسی و نیاز به آن‌ها است . برای درک بیشتر public or private و کارایی آن وبسایت های زیر مفید هستند

- microsoft doc 1
- microsoft doc 2
- dotnetperl

۴.۲.۹ serialize and deserialize

به عمل تبدیل کردن یک شی به بایت برای ذخیره یا انتقال آن شی در فایل یا حافظه `serialize` است.
هدف اصلی `serialize` ذخیره کردن آن شی است که هر موقعی که نیاز شد دوباره بتوان آن شی را تولید کرد. به عمل تبدیل کردن بایت های ذخیره شده به شی `deserialize` می‌گویند.
برای `serialize` معمولا شی های یک کلاس رو به صورت رشته ** در یک فایل می‌ریزند.
برای `deserialize` معمولا هر خط فایل تشکیل دهنده‌ی یک شی است و با زدن حلقه ی for روی هر خط آن می‌توان شی ها را بدست آورد. و معمولا آنها را در آرایه یا لیستی از نوع آن شی می‌ریزند.



شکل ۳.۹: serialization

```
class Student
{
    private int Id;
    private string Name;
    public string Serialize()
    {
        return this.Id + " " + this.Name;
    }
    public void AddtoFile()
    {
    File.AppendAllText(string path , this.Serialize());
    }
}
```

نمونه کد ۷۶: serialize

** string

```
private List<Student> Students;
private void desrialize(string file)
{
    string[] lines = File.ReadAllLines(file);
    foreach(string line in lines)
    {
        Student s = Student.Parse(line);
        Students.Add(s);
    }
}
public static Student Parse(string line)
    {
        string[] tokens = line.Split(',');
        int id = int.Parse(tokens[0]);
        string name = tokens[1];
        return new Student(id, name);
    }
}
```

نمونه کد ۷۷: deserialize

همانطور که در نمونه کد ۷۷ مشاهده می‌کنید متد `Parse` از نوع `static` است و برای صدا زدن آن باید از دستور `<classname.method>` استفاده کرد . برای درک بیشتر مفهوم، `serialize and deserialize` و کارایی آن وبسایت های زیر مفید هستند.

- microsoft doc
- csharpcorner

۳.۹ دستورات مهم فایل

File.ReadAllText(string path): محتویات درون فایل را می‌خواند و خروجی آن از نوع رشته است.
File.ReadAllLines(string path): محتویات درون فایل را می‌خواند و خروجی آن از نوع آرایه ای از رشته است که اندیس i ام آن خط i ام فایل است.
File.WriteAllText(string path, string content): یک فایل جدید درست کرده و محتوای داده شده (ورودی دوم) را در فایل می‌ریزد.
File.WriteAllLines(string path, string[] content): یک فایل جدید درست کرده و محتوای داده

شده (ورودی دوم) را در فایل می‌ریزد به طوری که اندیس i ام آن خط i ام فایل را تشکیل دهد.
File.AppendAllText(string path, string content): به فایل موجود در `path` محتوای داده شده (ورودی دوم) را اضافه می‌کند. و اگر فایل مورد نظر موجود نباشد آن را ایجاد می‌کند.
File.AppendAllLines(string path, string[] content): به فایل موجود در `path` محتوای داده شده (ورودی دوم) را اضافه می‌کند به طوری که اندیس i ام آن خط i ام فایل را تشکیل دهد و اگر فایل مورد نظر موجود نباشد آن را ایجاد می‌کند.

۴.۹ کد زده شده درون کلاس

```
۱ private static void Usage()
۲ {
۳     Console.WriteLine(" this program can be used to reigster students ");
۴     Console.WriteLine(" the usage syntax is as follows: ");
۵     Console.WriteLine(" cs.exe find|list|add [seachstring] ");
۶ }
```

نمونه کد ۷۸: نحوه ی استفاده از برنامه

همانطور که در نمونه کد ۷۸ مشاهده می‌کنید، در این جلسه مثال دانش‌جو و واحد درسی انتخاب شده است. در این برنامه کاربر با استفاده از `args` می‌تواند با دستورات

add ، list ، find [wanted] ، دانشجویی را یا ثبت نام بکند یا لیست دانشجو ها را بگیرد و یا دانشجوی خاصی را پیدا کند. پس طبیعتا دو کلاس داریم یکی از نوع دانش‌جو [††] و دیگری از نوع واحد درسی [‡‡] و یک کلاس از نوع برنامه [§§] که متد اصلی در آن است.

[††] Student
[‡‡] Course
[§§] Program

```
public class Student
{
    private int Id;
    public string Name;
    public Student(int id, string name)
    {
        this.Id = id;
        this.Name = name;
    }
}
```

نمونه کد ۷۹: Student member variables and constructor

```
public class Course
{
    private string Name;
    private int Code;
    private string StorageFileName;
    private List<Student> Students;
    public Course(string name, int code, string storageFileName)
    {
        this.Name = name;
        this.Code = code;
        this.StorageFileName = storageFileName;
        this.Students = new List<Student>();
        LoadFile(storageFileName);
    }
    private void LoadFile(string file)
    {
        string[] lines = File.ReadAllLines(file);
        foreach(string line in lines)
        {
            Student s = Student.Parse(line);
            this.RegisterStudent(s);
        }
    }
    public void RegisterStudent(Student s)
    {
        Students.Add(s);
    }
}
```

نمونه کد ۸۰: Course member variables and constructor

در نمونه کد ۸۰ هر بار که یک شی جدید ساخته می‌شود در سازنده‌ی[¶¶] آن متد `LoadFile` صدا زده می‌شود تا شی های ساخته شده در فایل در لیست دانش‌جو ها ریخته شود. که همان مفهوم `deserialize` که در نمونه کد ۷۷ به آن اشاره شده است.

¶¶ Constructor

دستور add

```
Program class //
Course math = new Course(name: "Math", code: 101, "Students.csv");
if (args[0] == "add")
    {
    Student s = Student.GetStudentFromConsole();
    math.RegisterStudent(s);
    math.StoreInFile();
    }
Student class //
public static Student GetStudentFromConsole()
    {
        System.Console.Write(" "Id?);
        int id = int.Parse(Console.ReadLine());
        System.Console.Write(" "Name?);
        string name = Console.ReadLine();
        return new Student(id, name);
    }
Course class //
internal void StoreInFile()
{
    List<string> lines = new List<string>();
    foreach(Student s in this.Students)
    {
        lines.Add(s.Serialize());
    }
    File.WriteAllLines(StorageFileName, lines);
}
```

نمونه کد ۸۱: add student

در نمونه کد ۸۱ دستور add دانش‌جو را از ورودی می‌گیرد که متد آن از نوع `static` و با دستور `Student.GetStudentFromConsole();` می‌توان این کار را انجام داد و او را ثبت نام و در فایل ذخیره می‌کند که همان مفهوم `serialize` که در نمونه کد ۷۶ اشاره شده می‌باشد.

دستور find

```
Program class //
else if (args[0] == "find" && args.Length == 2)
{
    string searchKey = args[1];
    string result = Found" "Not;
    Student s = math.FindStudent(searchKey);
    if (null != s)
        result = s.Serialize();
    Console.WriteLine(result);
}
Course class //
public Student FindStudent(string searchKey)
{
    Student foundStudent = null;
    searchKey = searchKey.ToLower();
    foreach(Student s in Students)
    {
        if (s.Name.ToLower() == searchKey)
        {
            foundStudent = s;
        }
    }
    return foundStudent;
}
```

نمونه کد ۸۲: find student

دستور find در لیست دانشجو ها می‌گردد و اگر دانش‌جوی مورد نظر را پیدا کند آن دانش‌جو را برمی‌گرداند و در غیر این صورت `null` برمی‌گرداند.

دستور list

```
Program class //
 else if (args[0] == "list" && args.Length == 1)
    math.PrintStudents();
Course class //
internal void PrintStudents()
{
    foreach(Student s in this.Students)
    {
        Console.WriteLine(s.serialize());
    }
}
```

نمونه کد ۸۳: list student

دستور list تمام دانش‌جو های آن فایل را چاپ می‌کند.

وبسایت های `microsoft doc` [**csharp**] `dotnet perl` [**dotnetperls**] `GeeksforGeeks` [**GeeksforGeeks**] کمک شایانی در یادگیری `csharp` می‌کنند.

جلسه ۱۰

(Directory/File) دیرکتوری و فایل

علی رهنما علمداری - ۱۳۹۸/۱۲/۱۷

جزوه جلسه ۱۰ام مورخ ۱۳۹۸/۱۲/۱۷درس برنامه‌سازی پیشرفته تهیه شده توسط علی رهنما علمداری. در زبان سی شارپ می توان به وسیله کلاس `Directory` با پوشه ها کارکرد همانند کلاس `File` کلاس ، `Directory` یکسری متدهای `static` دارد که به ما اجازه انجام عملیات های مختلف بر روی پوشه ها را می دهند

۱.۱۰ Directoryclass

۱.۱.۱۰ ایجاد پوشه

به وسیله متد `CreateDirectory` نمونه کد ۸۴ می توان پوشه جدید در مسیر مشخص شده ایجاد کرد

```
۱ Directory.CreateDirectory(@"D:\TestFolder\Test");
```

نمونه کد ۸۴: ساخت پوشه در سی‌شارپ

این دستور از بالاترین سطح شروع به ایجاد پوشه می کند . برای مثال در کد بالا در صورت عدم وجود پوشه `TestFolder` این پوشه ایجاد شده و سپس پوشه `Test` داخل ایجاد می شود.

۲.۱.۱۰ بررسی وجود یک پوشه

بوسیله متد `Exists` نمونه کد ۸۵ می توان بررسی کرد که یک پوشه وجود دارد یا خیر:

```
if (!Directory.Exists("D:\\Test"))
{
    Directory.CreateDirectory("D:\\Test");
}
```

نمونه کد ۸۵: بررسی وجود یک پوشه در سی‌شارپ

۳.۱.۱۰ دریافت لیست فایل های موجود در یک پوشه

بوسیله دستور `GetFiles` نمونه کد ۸۶ می توان لیست فایل های داخل یک پوشه را بدست آورد. این دستور آرایه ای از رشته ها را بر میگرداند که شامل مسیر و نام فایل های داخل پوشه است:

```
var files = Directory.GetFiles("D:\\MyFolder");

foreach (var file in files)
{
    Console.WriteLine(files);
}
```

نمونه کد ۸۶: فایل های یک پوشه در سی‌شارپ

۴.۱.۱۰ بدست آوردن زیر پوشه های داخل یک پوشه

بوسیله دستور `GetDirectories` نمونه کد ۸۹ می توان لیست پوشه های داخل یک پوشه را بدست آورد:

```
var subDirectories = Directory.GetDirectories("D:\\MyFolder");

foreach (var directory in subDirectories)
{
    Console.WriteLine(directory);
}
```

نمونه کد ۸۷: زیر پوشه های یک پوشه در سی‌شارپ

حال با دانستن کلاس و متدهای بالا قصد نوشتن کدی را داریم نمونه کد ۸۸ که با گرفتن آدرس یک پوشه، فایل های موجود در آن و همچنین فایل های موجود در تمام زیر دایرکتوری های آن را بنویسد

- نکته:برای این کار لازم است با مفهوم تابع بازگشتی * آشنا باشیم

*RecursiveFunction

```
static void Main(string[] args)
        {

            printfileInDir(@"c:\test\...");

        }

        private static void printfileInDir(string v)
        {
            var dir=Directory.GetDirectories(v);
           printfile(v);
           foreach(var d in dir )
           printfileInDir(d);
        }

        private static void printfile(string v)
        {
            var files =Directory.GetFiles(v);
            foreach(var f in files)System.Console.WriteLine(f);
        }
```

نمونه کد ۸۸: تمام فایل های پوشه در سی‌شارپ

۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص

در این مرحله می‌خواهیم برنامه‌ای بنویسیم که با گرفتن یک رشته از ورودی تمام فایل هایی که در یک پوشه یا زیر پوشه های آن ،شامل آن رشته باشد را چاپ کند

۱.۲.۱۰ منطق اصلی برنامه

```
static void Main(string[] args)
        {
            DirectoryIndex dirIdx = new DirectoryIndex(dir: @"C:\test\...");
            dirIdx.CreateIndex("*.cs");
            while(true)
            {
                System.Console.Write(" "Query?);
                string q = Console.ReadLine();
                if (q == "exit")
                    break;

                List<string> result = dirIdx.Query(q);
                System.Console.WriteLine(" for found "Files + q);
                foreach(var f in result)
                    System.Console.WriteLine(f);
            }
```

نمونه کد ۸۹: منطق برنامه

* کلاس `DirectoryIndex` نمونه کد ۹۰ محلی برای ایندکس(ذخیره کردن)تمام اطلاعات درون

فایل های یک پوشه میباشد

* متد `CreateIndex` نمونه کد ۹۱ در کلاس `DirectoryIndex` وظیفه به وجود آوردن ایندکس با یک فیلتر به عنوان پارامتر را دارد
* متد `Query` نمونه کد ۹۴ در کلاس `DirectoryIndex` وظیفه تطابق رشته ورودی با اطلاعات داخل فایل ها را بر عهده دارد

DirectoryIndex ۲.۲.۱۰

```
class DirectoryIndex
    {
        private string dir;
        private Dictionary<string, List<string>> Index;

        public DirectoryIndex(string dir)
        {
            this.dir = dir;
            this.Index = new Dictionary<string, List<string>>();
        }
    }
```

نمونه کد ۹۰: DirectoryIndex

- دیکشنری `Index` شامل کلید `string` و مقدار لیستی از `string` است که کلید ها تمام کلمات موجود در فایل ها را در بر دارند ومقادیر شامل تمام فایل هایی است که در خود کلید متناظر را دارند.

CreateIndex ۳.۲.۱۰

```
internal void CreateIndex(string filter)
        {
            List<string> allFiles = new List<string>();
            GetAllFiles(dir, filter, ref allFiles);

            foreach(var file in allFiles)
            {
                AddToIndex(file);
            }
        }
```

نمونه کد ۹۱: CreateIndex

* متد `GetAllFiles` نمونه کد ۹۲ با گرفتن آدرس یک پوشه و یک فیلتراز نوع `string` تمام فایل های آن پوشه و زیر پوشه های آن را که شامل فیلتر ورودی شوند را در لیست ذخیره میکند این متد عملکردی شبیه نمونه کد ۸۸ دارد

* متد `AddToIndex` نمونه کد ۹۳ تمام کلمات درون یک فایل را به عنوان `key` به دیکشنری تعریف شده در کلاس `DirectoryIndex` داده ولیستی از تمام فایل هایی که آن کلمه در آن وجود دارد را به عنوان `value` در نظر میگیرد.

- GetAllFailes

```
 private void GetAllFiles(string subdir, string filter, ref List<string> allFiles)
        {
            var files = Directory.GetFiles(subdir, filter);
            foreach(var file in files)
                allFiles.Add(file);

            foreach(var d in Directory.GetDirectories(subdir))
                GetAllFiles(d, filter, ref allFiles);
        }
```

نمونه کد ۹۲: GetAllFailes

- AddToIndex

```
private void AddToIndex(string file)
{
    string [] tokens =
    File.ReadAllText(file).Split(' ', ',', '.', '(', ')', '\n', '\r', ';');

    foreach(var tok in uniqueTokens)
    {
        if (!string.IsNullOrWhiteSpace(tok))
        {
            if (!this.Index.ContainsKey(tok))
                this.Index.Add(tok, new List<string>());

            this.Index[tok].Add(file);
        }
    }
}
```

نمونه کد ۹۳: AddToIndex

Query ۴.۲.۱۰

```
public List<string> Query(string q)
{
    if (this.Index.ContainsKey(q))
        return this.Index[q];

    return new List<string>();
}
```

نمونه کد ۹۴: Query

* این متد یک `string` گرفته و اگر دیکشنری این کلاس شامل آن کلید باشد `value` متناظر با آن که لیستی از نام فایل هاست برمیگرداند و در غیراین صورت یک لیست خالی برمیگرداند

۳.۱۰ منابع

`microsoft docs` [**liberty۲۰۰۵programming**] `C# Notes for Professionals` [**csharp**]

جلسه ۱۱

شبیه سازی چیلین وارز

شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹

در این جلسه صفحه ی چیلین وارز را شبیه سازی می کنیم.

هدف اصلی این جلسه نحوه ی فکر کردن به مسئله است و نه جزئیات حل آن. در این جلسه به ارتباط بین کلاس ها و آبجکت ها دقت کنید و نحوه ی استفاده از کیو را تمرین کنید.

در ادامه ی این جلسه از مفاهیم زیر استفاده می کنیم:

- enum
- queue
- switch

۱.۱۱ حل مسئله

مرحله ی اول حل مسئله، استفاده از کلاس ها و متود های لازم است و سپس پیاده سازی هرکدام از آن ها.

```
static void Main(string[] args)
{
    Table t = new Table(rows: 10, cols: 8);
    t.Player1 = new Player(row: 2, col: 3, table: t);
    t.Player2 = new Player(row: 8, col: 5, table: t);
    t.Print();
}
```

نمونه کد ۹۵: کد اولیه در program.cs

به این منظور، برای شبیه سازی صفحه ی چیلین وارز مشابه نمونه کد ۹۵ ابتدا از کلاس های table و player و متود های داخل هر یک، مثل print استفاده می کنیم و سپس هر یک از این کلاس ها و متود ها را پیاده سازی می کنیم.

در قسمت `new Table` روی کلمه ی Table کنترل و . را می زنیم و گزینه ی

`Generate type 'table' -> Genarate class 'table' in new file` را انتخاب می کنیم.

همین مراحل را برای Player نیز انجام می دهیم تا کلاس های Table و Player ساخته شوند.

در `Table.cs` بهتر است اسم ممبر ورایبرل ها در سی شارپ ، پاسکال کیس باشد برای همین روی rows ، cols جداگانه F۲ زده و اسم آن ها را به Rows ، Cols تغییر می دهیم.

برای نمایش صفحه ی بازی یک آرایه ی دو بعدی از کاراکتر ها به عنوان ممبر ورایبل می سازیم و آن را در کانستراکتور، همانطور که در نمونه کد ۹۶ می بینید، نیو می کنیم :

```
internal class Table
{
    public int Rows;
    public  int Cols;
    private char[,] Board;

    public Table(int rows, int cols)
    {
        this.Rows = rows;
        this.Cols = cols;
        this.Board = new char[rows, cols];
    }
}
```

نمونه کد ۹۶: کلاس Table

تا الان کلاس ما به این شکل درآمد. ولی الان در متغیر Board میتوان هر کاراکتری گذاشت در حالی که ما می خواهیم فقط انواع مشخصی از کاراکتر ها برای مشخص کردن دیوار، سلول خالی و بازیکن استفاده کنیم. به این منظور از `Enum` استفاده می کنیم.

Enum مشخص می کند که چه مقادیری قابل جاگذاری است و از استفاده از مقادیر اشتباه و غیرموردنظر جلوگیری می کند.

```
enum CellType
{
    Empty, Wall, Player
}
```

نمونه کد ۹۷: استفاده از enum در کلاس Table

حال در کد قبلی خود از CellType به جای char استفاده می کنیم.

متود FillBoard را به کلاس خود اضافه می کنیم و آن را طوری می نویسیم که صفحه را با خانه های خالی پر کند:

```
private void FillBoard(CellType empty)
{
    for(int i=0; i<Rows; i++)
    for(int j=0; j<Cols; j++)
    Board[i,j] = CellType.Empty;^^I
}
```

نمونه کد ۹۸: متود FillBoard در کلاس Table

حالا متود MakeWalls را برای کشیدن دیوار دور تا دور صفحه بازی یا Board می نویسیم:

```
private void MakeWalls()
{
    for(int i=0; i<Rows; i++)
    {
        Board[i, 0] = CellType.Wall;
        Board[i, Cols-1] = CellType.Wall;
    }
    for(int i=0; i<Cols; i++)
    {
        Board[0, i] = CellType.Wall;
        Board[Rows-1, i] = CellType.Wall;
    }
}
```

نمونه کد ۹۹: متود MakeWalls در کلاس Table

با توجه به کد اولیه ما، صفحه بازی دو بازیکن دارد:

`public Player Player1` و `public Player Player2`

دو بازیکن را در کلاس Table تعریف می کنیم.

در این قسمت به دو موضوع باید توجه کرد:

- اول آن که هر بازیکن باید محدوده ی بازی خود را بداند.
- دوم آن که بعد از تعریف هر بازیکن در خانه ای از صفحه ، آن خانه، خانه ی بازیکن باشد و دیگر خالی محسوب نشود.

برای مسئله ی اول باید در کد اولیه ی خود، متغیر Table را به متغیر های Player اضافه کنیم و در کلاس Player به منظور استفاده از این متغیر، تغییراتی اعمال می کنیم:

```
internal class Player
{
    public int Row;
    public int Col;

    private Table Table;
    public Player(int row, int col, Table table)
    {
        this.Table = table;
        if (row < table.Rows && col < table.Cols)
        {
            this.Row = row;
            this.Col = col;
        }
        else
            Console.WriteLine("Error");
    }
}
```

نمونه کد ۱۰۰: کلاس player

برای این که بتوانیم از table.Rows و table.Cols استفاده کنیم، باید متغیر های Rows ، Cols در کلاس Table به صورت Public تعریف شوند. اما مقدار آن ها نباید خارج از این کلاس قابل تغییر باشد، بنابراین آن ها را مانند نمونه کد ۱۰۱ تعریف می کنیم:

```
public int Rows {get; private set;}
public int Cols {get; private set;}
```

نمونه کد ۱۰۱: گت و ست در کلاس Table

حالا به حل مسئله ی دوم می پردازیم و برای هر بازیکن در کلاس Table گت و ست تعریف می کنیم:

```
private Player _Player1;

public Player Player1
{
    get { return _Player1; }
    set {
        this._Player1 = value;
        Board[value.Row, value.Col] = CellType.Player;
    }
}
```

نمونه کد ۱۰۲: تعریف گت و ست برای Player در کلاس Table

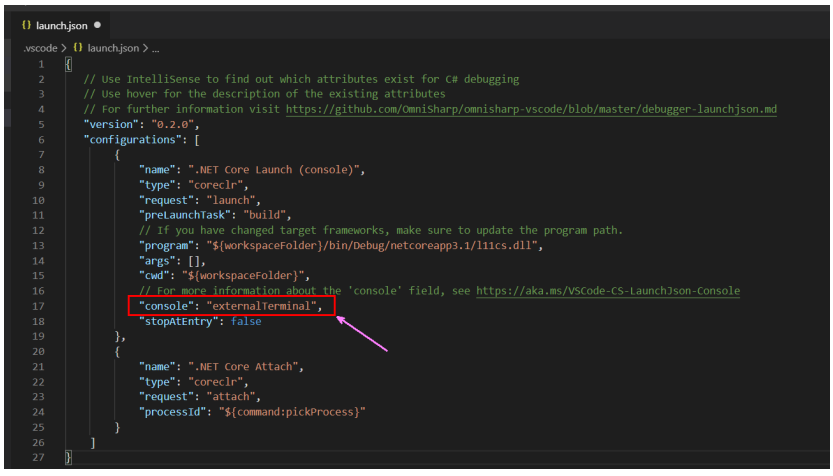
به طور مشابه برای استفاده از Cols ، Row در کلاس پلیر، آن ها را پابلیک می کنیم.

حالا قسمت آخر کد اولیه را پیاده سازی می کنیم: در کلاس Table متود Print را به صورت نمونه کد ۱۰۳ می نویسیم:

```
public void Print()
{
    Console.Clear();
    PrintFirstLine();
    for (int i =0; i<Rows; i++)
    {
        Console.Write(i + " ");
        for (int j=0; j<Cols; j++)
        PrintCell(i,j);
        Console.WriteLine();
    }
}
```

نمونه کد ۱۰۳: متود Print در کلاس Table

برای نمایش خروجی کد خود در اکسترنال ترمینال ، میتوان در فایل launch.json مقدار کنسول را مشابه ۱.۱۱ از اینترنال کنسول به اکسترنال ترمینال تغییر داد.



شکل ۱.۱۱: externalTerminal

متود PrintFirstLine را به مانند نمونه کد ۱۰۴ پیاده سازی می کنیم:

```
private void PrintFirstLine()
{
    Console.Write("  ");
    for (int i = 0; i < Cols; i++)
    Console.Write(i + " ");
    Console.WriteLine();
}
```

نمونه کد ۱۰۴: متود PrintFirstLine در کلاس Table

متود PrintCell را نیز به شکل نمونه کد ۱۰۵ می نویسیم:

```
private void PrintCell(int i, int j)
{
    CellType ct = Board[i,j];
    switch(ct)
    {
        case CellType.Empty:
        Console.Write('-');
        break;

        case CellType.Player:
        int playerNumber = GetPlayer(i, j);
        Console.Write(playerNumber);
        break;

        case CellType.Wall:
        Console.Write('w');
        break;
    }
    Console.Write(' ');
}
```

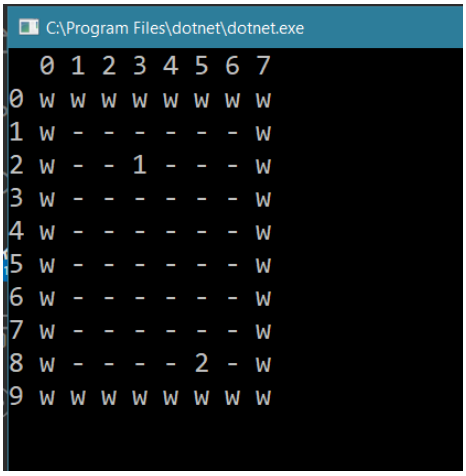
نمونه کد ۱۰۵: متود PrintCell در کلاس Table

در این متود از متود دیگری به نام GetPlayer استفاده کردیم، که آن را نیز به صورت زیر می نویسیم:

```
private int GetPlayer(int r, int c)
{
    int playerNumber = -1;

    if (Player1.Col == c && Player1.Row == r)
    playerNumber = 1;
    else if (Player2.Col == c && Player2.Row == r)
    playerNumber = 2;
    else
    Console.WriteLine("ERROR");

    return playerNumber;
}
```

نمونه کد ۱۰۶: متود GetPlayer در کلاس Table

با ران کردن Program.cs خروجی ما مشابه ۲.۱۱ می باشد:



شکل ۲.۱۱: اجرای Main در Program.cs

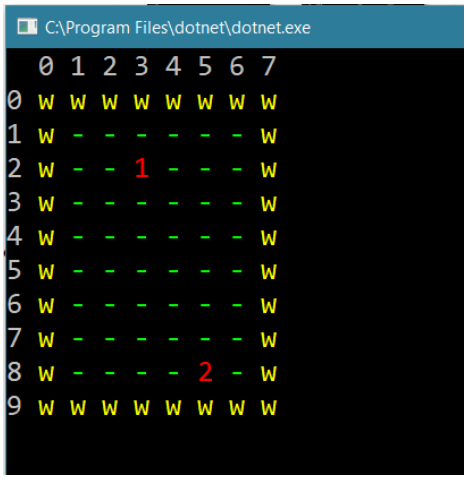
حال به متود Print ، مانند نمونه کد ۱۰۷ دستوراتی اضافه میکنیم تا صفحه ی بازی را رنگی چاپ کند.

```
private void PrintCell(int i, int j)
{
    var color = Console.ForegroundColor;
    CellType ct = Board[i,j];
    switch(ct)
    {
        case CellType.Empty:
            Console.ForegroundColor = ConsoleColor.Green;
            Console.Write('-');
            break;

        case CellType.Player:
            Console.ForegroundColor = ConsoleColor.Red;
            int playerNumber = GetPlayer(i, j);
            Console.Write(playerNumber);
            break;

        case CellType.Wall:
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.Write('w');
            break;

        case CellType.Visiting:
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.Write('o');
            break;
    }
    Console.Write(' ');
    Console.ForegroundColor = color;
}
```

نمونه کد ۱۰۷: تغییر متود Print در کلاس Table

برای استفاده از ConsoleColor باید عبارت `using System;` را به اول کد خود اضافه کنید. برای اطلاعات بیشتر در این زمینه می توانید از داکیومنت ماکروسافت استفاده کنید [**consoleColor**].

بعد از تغییر این قسمت از کد، خروجی برنامه به صورت زیر می شود:



شکل ۳.۱۱: اجرای Main بعد از نوشتن نمونه کد ۱۰۷

حالا که این قسمت از مسئله حل شد، می خواهیم پلیر ها را در صفحه حرکت بدهیم. بنابراین کلاس وکتور را تعریف می کنیم و به صورت زیر از آن استفاده می کنیم:

```
۱  Vector v = new Vector(row: 1, col: 1);
۲
۳  while ('q' != Console.ReadKey().KeyChar)
۴  {
۵      t.Player1.Move(v);
۶  }
```

نمونه کد ۱۰۸: استفاده از کلاس Vector

دستور Console.ReadKey() برنامه را متوقف کرده تا کاربر یک کلید را از روی کیبورد فشار دهد. بنابراین در برنامه ی بالا اگر کاربر کلید q را فشار دهد، برنامه ی اجرایی از حلقه خارج می شود و اگر بعد از آن کدی نوشته شده باشد، اجرا می شود.

حال، متود Move را در کلاس پلیر پیاده سازی می کنیم:

```
internal void Move(Vector v)
{
    this.Row += v.row;
    this.Col += v.col;
    Table.Update();
}
```

نمونه کد ۱۰۹: متود Move در کلاس Player

توجه کنید که Row ، Col در کلاس وکتور، باید به صورت پابلیک تعریف شده باشند تا بتوان در نمونه کد ۱۰۹ از آن ها استفاده کرد.

حالا متود Table.Update را نیز در کلاس Table مطابق نمونه کد ۱۱۰ پیاده سازی می کنیم:

```
public void Update()
{
    for(int i=1; i<Rows-1; i++)
    for(int j=1; j<Cols-1; j++)
    {
        if ((Player1.Row == i && Player1.Col == j) ||
        (Player2.Row == i && Player2.Col == j) )
        Board[i,j] = CellType.Player;
        else
        Board[i,j] = CellType.Empty;
    }
}
```

نمونه کد ۱۱۰: متود Update در کلاس Table

به حلقه ی خود در Main مطابق نمونه کد ۱۱۱ دستوراتی اضافه می کنیم:

```
while ('q' != Console.ReadKey().KeyChar)
{
    t.Player1.Move(v);
    v.Negate();
    t.Player2.Move(v);
    t.Print();
}
```

نمونه کد ۱۱۱: تغییر Main در Program.cs

می خواهیم جهت حرکت را برعکس کنیم و پلیر دوم را با آن بردار حرکت دهیم و همچنین می خواهیم هر بار که حلقه اجرا می شود، صفحه ی بازی پرینت شود، پس برای تمیز تر شدن کنسول در متود Print دستور

`Console.Clear();` را اضافه می کنیم که صفحه ی کنسول یا اکسترنال ترمینال را پاک می کند. اگر این دستور را اضافه نکنیم، با هر بار اجرا شدن حلقه، یک صفحه ی بازی زیر صفحه ی کشیده شده ی قبلی چاپ می کند.

متود Negate را در کلاس وکتور می نویسیم:

```
۱  public void Negate()
۲  {
۳      row = -1 * row;
۴      col = -1 * col;
۵  }
```

نمونه کد ۱۱۲: متود Negate در کلاس Vector

حالا می خواهیم نزدیک ترین دیوار به یکی از پلیر ها را بیابیم. برای این کار، چهار خانه ی اطراف آن را چک می کنیم، اگر دیوار نبود، اطراف هر کدام از آن جهار خانه را چک می کنیم و اگر همچنان دیواری پیدا نکردیم، این کار را برای هر یک از خانه های چک شده ی جدید ادامه می دهیم تا به این شکل، نزدیک ترین دیوار به پلیر مورد نظر را بیابیم. این متود مشابه BFS می باشد. بنابراین متود BFSVisit را به کلاس Table اضافه می کنیم:

```
۱  public void BFSVisit(int i, int j)
۲  {
۳      Queue<Vector> toVisit = new Queue<Vector>();
۴      toVisit.Enqueue(new Vector(i, j));
۵
۶      while (toVisit.Count != 0)
۷      {
۸          var v = toVisit.Dequeue();
۹          Board[v.row, v.col] = CellType.Visiting;
۱۰         foreach(Vector n in GetNeighbors(v))
۱۱         {
۱۲             if (Board[n.row, n.col] != CellType.Visiting)
۱۳             toVisit.Enqueue(n);
۱۴         }
۱۵         Print();
۱۶         Console.ReadKey();
۱۷     }
۱۸ }
```

نمونه کد ۱۱۳: متود BFSVisit در کلاس Table

همانطور که متوجه شدید، برای به کاربردن این قطعه از کد، باید Visiting را به `Enum cellType` اضافه کنیم. متود GetNeighbors را نیز به صورت زیر پیاده سازی می کنیم:

```
private List<Vector> GetNeighbors(Vector v)
{
    List<Vector> neighbors= new List<Vector>();
    if (v.row-1 > 0)
    neighbors.Add(new Vector(v.row-1, v.col));
    if (v.col-1 > 0)
    neighbors.Add(new Vector(v.row, v.col-1));
    if (v.row+1 < Rows-1)
    neighbors.Add(new Vector(v.row+1, v.col));
    if (v.col+1 < Cols-1)
    neighbors.Add(new Vector(v.row, v.col+1));

    return neighbors;
}
```

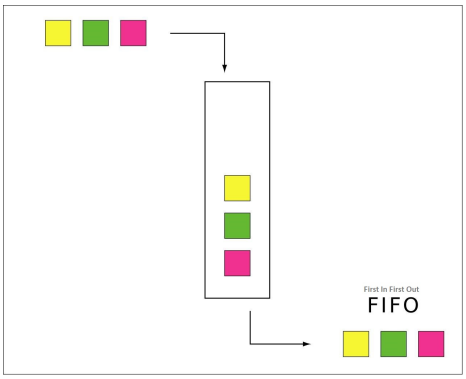
نمونه کد ۱۱۴: متود GetNeighbors در کلاس Table

حال به توضیح Queue که در این کد استفاده شد، می پردازیم.

۲.۱۱ Queue

در جلسات گذشته با کالکشن هایی از قبیل لیست، دیکشنری، هش ست و غیره آشنا شدیم. این جلسه با کالکشن دیگری به نام کیو آشنا خواهیم شد.

کیو یا صف مجموعه ای از اعضاست که به اصطلاح `First in, first out` است. یعنی عضو اضافه شده همیشه به اول آن اضافه می شود و تنها عضو انتهایی را می توان از آن خارج کرد.



شکل ۴.۱۱: out first in first

یعنی عضوی که زودتر از همه اضافه شده، زودتر از همه هم خارج می شود. مهم ترین متود های قابل استفاده در کیو، دو چیز است:

- Enqueue که عضوی را به اول کیو اضافه می کند.
- Dequeue که آخرین عضو کیو را خارج می کند.

حالا در متود PrintCell کیس ویزیتینگ را اضافه می کنیم:

```
case CellType.Visiting:
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.Write('o');
    break;
```

نمونه کد ۱۱۵: تغییر متود PrintCell در کلاس Table

این قسمت از کد، خانه های ویزیت شده را با o آبی رنگ نشان می دهد. خط `t.BFSVisit(5,4);` را به Main در Program.cs اضافه می کنیم تا نحوه ی کار متودی که نوشتیم را ببینیم. خروجی ما به شکل زیر خواهد بود:

شکل ۵.۱۱: اجرای متود BFSVisit

مطابق انتظار ما، این متود به ترتیب خانه های اطراف را به CellType.Visiting تبدیل کرد. حالا می توانیم از این متود در کدمان استفاده کنیم. مثلا می توانیم شرطی بگذاریم که پلیر ما در خلاف جهت نزدیک ترین دیوار حرکت کند یا به عنوان مثال می توان قبل از بررسی کردن همسایه های هر خانه، آن خانه را برای هر یک از همسایه ها به عنوان خانه ی قبل ذخیره کنیم، تا بعد از پیدا کردن مقصد، با توجه به خانه های قبل که ذخیره

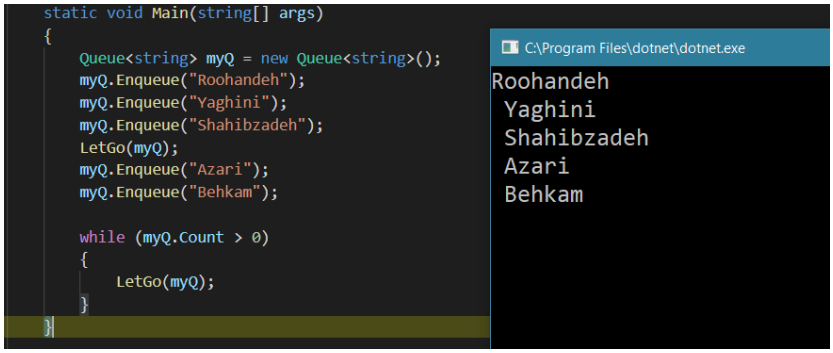
شده اند مسیری از خانه ی شروع تا مقصد پیدا کنیم.

برای این که مفهوم کیو را بهتر درک کنیم، مثال زیر را پیاده سازی می کنیم :

```
static void Main(string[] args)
{
    Queue<string> myQ = new Queue<string>();
    myQ.Enqueue("Roohandeh");
    myQ.Enqueue("Yaghini");
    myQ.Enqueue("Shahibzadeh");
    LetGo(myQ);
    myQ.Enqueue("Azari");
    myQ.Enqueue("Behkam");

    while (myQ.Count > 0)
        LetGo(myQ);
}

private static void LetGo(Queue<string> myQ)
{
    string person = myQ.Dequeue();
    Console.WriteLine(person);
    Console.ReadKey();
}
```

نمونه کد ۱۱۶: مثالی برای درک بهتر کیو در سی شارپ

همانطور که متوجه شده اید، با هر بار Dequeue کردن، اولین عضوی که وارد شده بود خارج می شود.



شکل ۶.۱۱: خروجی ۱۱۶

برای یادگیری بهتر، می توانید داکیومنت ماکروسافت را در زمینه ی کیو بخوانید [**queue**].

نحوه استفاده از کیو در پایتون نیز به شکل زیر است:

```
import queue
q = queue.Queue()
q.put(1)
x = q.get()
```

برای اطلاعات بیشتر درباره ی کیو در پایتون، می توانید داکیومنت پایتون را مطالعه کنید [**python**].

جلسه ۱۲

stacks- objects

بنفشه قلی نژاد - ۱۳۹۸/۱۲/۲۴

جزوه جلسه ۱۲ام مورخ ۱۳۹۸/۱۲/۲۴

برنامه‌سازی پیشرفته

stacks-objects

۱.۱۲ استک

۱.۱.۱۲ استک چیست؟

به طور کلی می توان گفت چیزی شبیه لیست است با ویژگی های منحصر به فرد تر و شبیه queue با تفاوت این که کیو یا صف، First in first out و استک First in Last out است.
به این معنی که هر چیزی که اول وارد Queue میشود،به همان ترتیب اولیه خارج میشود. اما در استک برعکس! یعنی هر آنچه که اول وارد میشود ، اخر از همه خارج میشود.
مانند گذاشتن چند قطعه کتاب به ترتیب روی یک دیگر، به این شکل که اخرین کتابی که روی کتاب دیگر گذاشته میشود، اولین کتاب قابل دسترس است.

استک در `csharp` اینگونه تعریف میشود:

```csharp
using System.Collections.Generic;

public class Program
{
   static void Main(string[] args)
    {
        Stack<string> stack = new Stack<string>();
        }
}
```

نمونه کد ۱۱۷: تعریف کلاس استک

۲.۱.۱۲ call stack

هنگام دیباگ کردن، در قسمت call stack متد ها را که به ترتیب فراخوانی شده اند نشان میدهد و کارکرد آن دقیقا مانند استک است.

به این نمونه کد توجه کنید:

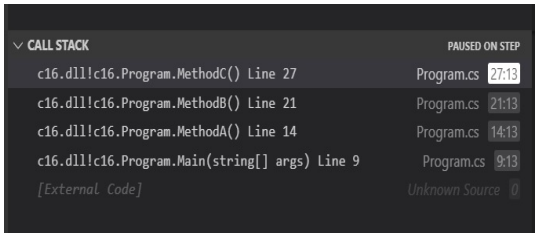
```
using System;

namespace c16
{
    class Program
    {
        static void Main(string[] args)
        {
            MethodA();
        }
         static void MethodA()
        {
            Console.WriteLine(Before" A: "In);
            MethodB();
            Console.WriteLine(After" A: "In);
        }

         static void MethodB()
        {
            Console.WriteLine(Before" B: "In);
            MethodC();
            Console.WriteLine(After" B: "In);
        }

        static void MethodC()
        {
            Console.WriteLine(C" "In);
        }


    }
}
```

نمونه کد ۱۱۸: call stack

در این مثال، زمانی که چند متد تودرتو در یک دیگر صدا زده می شوند،تا زمانی که متد باز شده بسته نشود، نمی توان تا انتهای برنامه پیش رفت.

پس به ترتیب از اخر به اول، هر متدی که اخر از همه باز میشود، اول از همه بسته میشود تا برنامه بدون خلل اجرا شود. خروجی این برنامه به این صورت است:

```
In A:Before

In B before

In C:

In B after

In C after
```



شکل ۱.۱۲: call stack

در صورت دیباگ کردن خط به خط این برنامه، در قسمت Call stack می بینیم که آخرین متد فراخوانی شده، بالا تر از همه قرار دارد.
همان طور که در تصویر ۱.۱۲ می بینید،دقیقا مانند کتاب هایی که به ترتیب روی یک دیگر اند،این متد ها نیز به ترتیب روی هم فراخوانی شده اند.

stack's API ۳.۱.۱۲

API مخفف Application Programing Interface است. منظور واسطه هایی است که برای هر کلاس در سی شارپ تعریف شده است. درواقع همان استفاده از متد های پیاده سازی شده ،برای آن کلاس بخصوص می باشد.
در API استک، بیشترین و کاربردی ترین متد ها شامل:

- push()
- pop()

درواقع: stack.push() داده ای را به قسمت بالای استک اضافه می کند و stack.pop() آخرین و بالا ترین داده را از استک حذف می کند و آن را برمی گرداند.
برای بهتر متوجه شدن، به این نمونه کد توجه کنید:

```
using System;
using System.Collections.Generic;
class program
{
    static void Main(string[] args)
        {
            Stack<string> stack = new Stack<string>();
            stack.Push("Main");
            stack.Push("MethodA");
            stack.Push("MethodB");
            stack.Push("MethodC");

            while (stack.Count > 0)
                Console.WriteLine(stack.Pop());
        }
    }
```

نمونه کد ۱۱۹:

خروجی استک:

```
MethodC

MethodB

MethodA

Main
```

همان طور که در قسمت استک چیست؟ به ماهیت استک اشاره شد، با نوشتن دستور stack.pop() اخرین داده را از استک حذف ، و اول از همه آن را برمی گرداند و چاپ می کند! و در این حلقه به اندازه ی استک، به ترتیب این عمل را اجرا می کند تا همه ی اعضا چاپ بشوند.

۴.۱.۱۲ کاربرد

نمونه کد زیر،برای تشخیص دادن حالت درست پرانتز گزاری ها، و نمونه ای کوچک از به کارگیری استک است:

```
program class Program
        private static bool IsBalanced(string s)
        {
            Stack<char> stack = new Stack<char>();
            foreach(char c in s)
            {
                if (c == '(' || c == '[')
                    stack.Push(c);

                if (c == ')' || c == ']')
                    if ( !stack.TryPop(out char top) || !IsCompatible(c, top))
                        return false;
            }
            return stack.Count == 0;
        }
        private static bool IsCompatible(char c, char t)
                {
                (c == ')' && t == '(')
                 ||  (c == ']' && t == '[')) ;
        }

                static void Main(string[] args)
        {
            string a = )]" (a+c) / (a+b) * (a-b) [( * "(a+b);
            string b = )" (a+c) / (a+b) * (a-b) ( * "(a+b;
            Console.WriteLine(IsBalanced(a));
            Console.WriteLine(IsBalanced(b));
        }
        }
```

نمونه کد ۱۲۰: پرانتز گذاری صحیح

خروجی:

```
True
False
```

در پرانتز گذاری ها همان طور که تعداد پرانتز ها مهم است، ترتیبشان نیز اهمیت دارد. یعنی هر پرانتزی که اول از همه باز می شود، آخر از همه نیز بسته میشود. دقیق به همان شکلی که یک استک عمل می کند. پس اگر پرانتزی باز شود، آن را در بالا ترین جای استک قرار دهدو(آن را push کند،)

و اگر بسته میشود، با کمک متد TryPop().* و متد isCompatible() که هردو بولین هستند، پرانتز ها را مقایسه می کند و اگر تا زمانی که اندازه ی استک صفر شود شرط برقرار باشد، نحوه ی پرانتز گذاری ها صحیح است.
متد isCompatible() نیز اخرین داده ی استک را با پرانتز بسته شده مقایسه می کند و اگر ترتیب آن صحیح بود، ارزش این متد true می باشد.

۲.۱۲ objects

۱.۲.۱۲ تعریف

همان طور که میدانیم، می توان یک کلاس را پیاده سازی کرده و شیی راجداگانه برایش تعریف کرد. اما زمانی که بخواهیم متد ها و یا ویژگی هایی که برای شی بخصوص تعریف کردیم را پیاده سازی کنیم ، برای هر نوع شیی علاوه بر ویژگی های به خصوص آن، یک سری متد های عام دیگر مانند:
`lrEquals() , Tostring(),GetType() , GEtHashCode` معرفی می کند. سوال این است که چرا برای هر شیی از انواع مختلف که می نویسیم، این توابع مشترک هستند؟
در سی شارپ علاوه بر َشیی که به طور عام تعریف میشود،یک کلاس object به طور خاص تعریف شده که می توان آن را به صورت یک شی جدا نوشت، که شامل ویژگی ها ومتد های بالااست. در واقع هر شیی که ما تعریف می کنیم، به نوعی یک نوع object است.
می توان این طور گفت که تمام ویژگی های آبجکت ، در هر شی موجود است.
تعریف آبجکت در `csharp`

```
۱ public class Program
۲ {
۳ Object obj;
۴ obj = new Object();
۵ }
```

نمونه کد ۱۲۱: تعریف آبجکت در سی‌شارپ

*این متد در صورتی که داده در استک موجود باشد، داده با را برگردانده وارزش آن درست است،و اگر نتواند داده را از استک بردارد، ارزش آن نادرست است

۲.۲.۱۲ overriding

می دانیم به هر حال، هر شی به نوعی از آبجکت ارث بری می کند. زمانی که بتوانیم متد هایی که در کلاس آبجکت به صورت عمومی تعریف شده اند را جور دیگری تغییر داد، به صورتی که بتوان چیزی را که میخواهیم از آن ها بدست بیاوریم و کارکرد آن را تغییر دهیم، به اصطلاح آن متد را `override` کردیم.

در این جلسه ما override کردن سه نوع متد را می آموزیم:

- equals()
- Tostring()
- GetHashCode()

۳.۲.۱۲ equals

تابع Equal یک تابع بولین و تعریف شده در کلاس آبجکت است. در حالت عادی زمانی که دو شی در حافظه آدرس برابر داشته باشند، به اصطلاح اشاره گر یا reference آن ها برابر باشند، آن دو را مساوی برمیگرداند و در غیر این صورت تابع مقدار غلط را برمیگرداند.

مثال:

```
۱  static void  main
۲  {
۳  Student s = new Student(name: "Ali", id:98521231);
۴  Student s1 = new Student(name: "Zahra", id:77521231);
۵  Student s2 = new Student(name: "Ali", id:98521231);
۶  s3 = s;
۷  Console.WriteLine(s.Equals(s3))
۸  Console.WriteLine(s.Equals(s1));
۹  Console.WriteLine(s.Equals(s2))
۱۰
۱۱ }
```

نمونه کد ۱۲۲: Equals

در این مثال، **student** یک کلاسیست که جداگانه با ویژگی های نام و شماره دانشجویی تعریف شده است. مقدار چاپ شده برای این قطعه کد `true, false, false` می باشد.
(در صورتی که مشخصا شی s و شی s۲ از لحاظ ویژگی یکسانند.)

برای این که بخواهیم دو شی منجصر به فرد را از لحاظ ویژگی هایشان با یک دیگر مقایسه کنیم، احتیاج به تغییر کاکرد تابع **Equals** در کلاس دانش آموز داریم.

کلاس :student

```
internal class Student
    {
        private string name;
        private int id;

     public Student(string name, int id)
        {
            this.name = name;
            this.id = id;
        }

     public override bool Equals(object obj)
        {
           if (! (obj is Student))
              return false;
        Student other = (Student) obj ;
        Student other = obj as Student;

     return this.Id == other.Id ;
    }
```

نمونه کد ۱۲۳: Equals

در این قسمت ما تعریف کردیم که اگر شی ما دانش آموز نبود، آن را **false** برگرداند و در صورت دانش آموز بودن، شی را به قالب کلاس دانش اموز دربیاورد، و اگر تمام ویژگی ها برابر باشند، مقدار برگداننده ی این متد **true** باشد. در صورت تعریف این متد، پاسخ کد ۱۲۲

به صورت `true, false, true` چاپ می کند.

اما راه بهتری نیز برای این کار وجود دارد!

در صورت به قالب دراوردن شی، به کلاسی که میخواهیم، در صورت null بودن، استثنایی از نوع nullreference exception پرتاب می کند.

نمونه کد ۱۲۲ فقط برای مقایسه دو شی از یک نوع انجام پذیر می باشد. ولی علاوه بر آن، می توان دو چیز متفاوت را مانند هر یک از ویژگی های دانش آموز، با رشته حرفی یا عدد را به صورت جداگانه، مقایسه کرد.

برای مثال به این قطعه کد توجه کنید:

```
 public override bool Equals(object obj)
{
if ((obj is Student))
return this.Id == (obj as Student).Id && this.Name == (obj as Student).Name;

if ((obj is string))
return this.Name == (obj as string);

if ((obj is int))
return this.Id == (int) obj;

return false;
}
```

نمونه کد ۱۲۴: Equals

همان طور که در نمونه کد ۱۲۴ می بینید،در صورت نوشتن `obj as (...)` آن شی را از نوع خاص تعیین شده در نظر میگیرد. به طوری که اگر null باشد، برنامه متوقف نمی شود،
همان طور که در خط سوم و چهارم، و پنجم و ششم مشاهده می کنید، می توان ویژگی یک شی از کلاس بخصوص را با رشته ی حرفی یا عدد نیز مقایسه کند. (البته برای int نمی توان از **obj as int** استفاده کرد. چرا که از نوع ساختار یا `struct` است و از جنس کلاس نیست. به همین علت باید آن را قالب بندی یا cast کرد)
با تغییر کارکرد تابع Equals به صورت نمونه کد ۱۲۴ حاصل کار به صورت زیر است:

```
public static void main
{
Student s = new Student(name: "Ali", id:98521231);
Student s1 = new Student(name: "Zahra", id:77521231);
//
 Console.WriteLine(s.Equals(s1));
Console.WriteLine(s.Equals(s2));
Console.WriteLine(s.Equals(98521231));
Console.WriteLine(s.Equals("Ali"));
Console.WriteLine(s.Equals("Zahra"));
Console.WriteLine(s.Equals(98989898));
}
```

نمونه کد ۱۲۵: Equals

جواب به ترتیب:

```
false, true, true, true, false, false
```

tostring ۴.۲.۱۲

این متد به صورت پیش فرض،می تواند object را به صورت رشته ی حرفی چاپ کند. برای مثال:

```
۱ static void Main(string[] args)
۲ {
۳ object obj = new object();
۴ Console.WriteLine(obj);
۵
۶ }
```

نمونه کد ۱۲۶: ToString

این برنامه در صورت اجرا شدن **System.object** را چاپ می کند. می توان با تغییر کارکرد این متد، شی از نوع کلاس خاص را با فراخوانی متد ToString تبدیل به رشته ی حرفی کرد. مثلا این متد را می توان در کلاس دانش اموز [۱۲۳] override کرد.

```
۱ public override string ToString() =>  $:{this.Id}" "{this.Name};
```

نمونه کد ۱۲۷: ToString

در متد **main:**

```
۱ static void Main(string[] args)
۲ {
۳ student stu = new student("Ali", 98532445);
۴ Console.Writeline(stu);
۵ string s = stu.ToString();
۶ Console.WriteLine(s)
۷ }
```

نمونه کد ۱۲۸: Equals

و به ازای نمونه کد ۱۲۸ رشته ی حرفی

```
Ali :98532445
```

چاپ میشود

GetHashCode ۵.۲.۱۲

متد **GetHashCode** برای زمانی است که می خواهیم از یک object یا شی، در دیکشنری استفاده کنیم.

در دیکشنری، برای بیشتر شدن سرعت لازم این است که در صورت برابر بودن دو شی، یک کد مساوی برگرداند و در صورت نبود، حدالامکان مساوی نباشند.
به همین علت، معمولا متد GetHashcode و Equals با هم پیاده سازی می شوند. در این صورت، می توان از خود شی از نوع کلاس خاص ،به عنوان کلید در دیکشنری استفاده کنیم.

برای مثال این متد را برای کلاس دانش آموز [۱۲۳] override می کنیم.

```
۱ public override int GetHashCode()
۲ {
۳ return this.Id.GetHashCode() ^ this.Name.GetHashCode();
۴ }
```

نمونه کد ۱۲۹: getHashCode

در متد Equals تعریف کردیم که در صورت برابر بودن نام و شماره ی دانشجویی دو شی یکسان نیز، آن ها را برابر در نظر بگیرد. به همین علت در متد، **xor GetHashCode** این دو ویژگی برگردانده میشود. به این صورت که از نام و یا شماری دانشجویی، می توان به عنوان کلید در دیکشنری استفاده کرد.

برای مثال:

```
static void main()
{
Dictionary<Student,List<int>> students=new Dictionary<Student,List<int>>();
Student ali = new Student(name:"Ali", id:98521231);
Student zahra = new Student(name: "Zahra", id:77521231);
students.Add(ali, new List<double>());
students.Add(zahra, new List<double>());

students[ali].Add(19);
students[zahra].Add(18);

Console.WriteLine(students[ali])
Console.WriteLine(students[zahra])

}^^I
```

نمونه کد ۱۳۰: getHashCode()

خروجی چاپ شده برای این کد، عدد ۱۹ و عدد ۱۸ است. همان طور که نشخص است، از ویژگی نام به عنوان کلید استفاده شده است.

در صورتی که قبل از نوشتن GetHashCode در کلاس دانش آموز، این کار امکان پذیر نبود.

جلسه ۱۳

Destructor

یاسمین مدنی - ۱۳۹۹/۱/۱۶

جزوه جلسه ۱۳ام مورخ ۱۳۹۹/۱/۱۶ درس برنامه‌سازی پیشرفته تهیه شده توسط یاسمین مدنی. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

۱.۱۳ عناوین کلی جلسه

موضوعات مطرح شده در این جلسه به شرح زیر است:

- دیستراکتور و فاینالایزر
- استراکت در زبان های c# و c++
- Reference Type VS Value Type

۲.۱۳ دیستراکتور و فاینالایزر

۱.۲.۱۳ دیستراکتور

هنگام نوشتن یک برنامه در راستای کنترل کردن منابع استفاده شده مانند فایل یا غیره همچنین هنگام استفاده مستقیم از حافظه نیازمند استفاده از Destructorبرای یک کلاس خواهیم بود به طور مثال کلاس و توابع زیر در زبان C++را درنظر بگیرید

```cpp
class Test
{
    int *pN;

public:
    Test(int n)
        : pN(new int[n])
    {
        cout << pN=" of size ,Constructor Test "In << n << endl;
    }

    ~Test()
    {
        cout << pN" of address ,Destructor "In << pN << endl;
        delete[] pN;
    }
};

void MethodForStackAllocDemo()
{
    Test a(5);
}

Test *MethodForHeapAllocDemo()
{
    Test *pTest = new Test(6);
    return pTest;
}
```

نمونه کد ۱۳۱: کلاس تست C++

توجه داریم که اگر constructor کلاس را به صورت زیر تعریف میکردیم به عنوان empty constructorشناخته می شد.

```cpp
Test(){}
```

نمونه کد ۱۳۲: کانستراکتور C++

از آنجا که در تعریف این کلاس مستقیما ازHeapبرای ذخیره آرایه مان استفاده کرده ایم نیاز است تا هنگام پایان کار با یکobject از جنس این کلاس حافظه را دوباره به سیستم بازگردانیم به همین سبب در دیستراکتور این کلاس حافظه به کار گرفته شده را حذف خواهیم کرد.

در کلاس یادشده

```
~Test(){}
```

نمونه کد ۱۳۳: دیستراکتور C++

پیاده سازی دیستراکتور کلاس است که برای انجام این کار کافیست از یک علامت مد در ابتدای کانستراکتور کلاس استفاده کنیم.

این نکته شایان توجه است که در ساخت یک objectجدید از کلاس تنها زمانی نیاز به استفاده از newداریم که بخواهیم پوینتر آن شی را به عنوان return valueداشته باشیم .

اگر یک شی رویStackتعریف شده باشد دیستراکتور در پایان هرscope صدا زده خواهد شد این به آن معناست که هر شی تنها در همان محدوده ای که تعریف شده قابل استفاده و دسترسی است و خارج آن فاقد اعتبار خواهد بود.برای مثال در کد زیرbتنها در محدوده شرط قابل دسترسی است

```
 if (true)
    {
        Test b(15);
        cout << statement" If "in << endl;
    }
```

نمونه کد ۱۳۴: C++

اگر شی رویHeapتعریف شده باشد باید پس از اتمام کار آن را حذف کنیم به مثال زیر توجه فرمایید:

```
int main()
{
 Test *pTest = MethodForHeapAllocDemo();
    delete pTest;
}
```

نمونه کد ۱۳۵: C++

در زبان برنامه نویسی C++بسته به دلخواه برنامه نویس می توان یک کلاس را رویHeap یاStackتعریف کرد این یکی از تفاوت های این زبان با سی شارپ است .در سی شارپ هر کلاس لزوما رویHeapتعریف می شود که در ادامه بیشتر به آن خواهیم پرداخت.

۲.۲.۱۳ فاینالایزر

سی شارپ دارای ویژگی به نامGarbage Collectorاست که مدیریت اتومات حافظه را ممکن می سازد .این به آن معناست که بار مدیریت حافظه های اختصاص یافته از دوش برنامه نویس برداشته می شود.برای اطلاعات بیشتر میتوانید به اینجا[**visualizationwebsite**].مراجعه کنید Garbage Collectorتنها متغیر ها و اشیایی که رویHeapقرار دارند را مورد بررسی قرار میدهد و متغیرهای ذخیره شده رویStack همانند C++ با پایان یافتن محدوده تعریف از دسترس خارج میشوند.

کلاس زیر در زبان سی شارپ را در نظر بگیرید

```
class Test
{
    public int N {get; set;}
    public Test(int n)
    {
        N = n;
        Console.Write($""{N});
    }

    ~Test()
    {
        Console.WriteLine($"{N}" finalizer "In);
    }
}
```

نمونه کد ۱۳۶: کلاس تست

مشابه دیستراکتور در زبان C++ را در سی شارپ فاینالایزر مینامند

```
~Test()
      {
          Console.WriteLine($"{N}" finalizer "In);
      }
```

نمونه کد ۱۳۷: فاینالایزر

فاینالایزر را به ندرت پیاده سازی خواهیم کرد اما باید به این نکات توجه کنیم که :

- فاینالایزرها برای Struct تعریف نمی شوند و مخصوص کلاس اند.
- هر کلاس تنها یک فاینالایزر دارد.
- فاینالایزرها رانمیتوان صدا زد و خودکار اجرا میشوند
- فاینالایزر پارامتر نمی گیرد

Struct ۳.۱۳

struct in cpp ۱.۳.۱۳

تفاوت استراکت ها با کلاس در این زبان در Public و Privateبودن آن هاست اجزای کلاس ها به طور پیش فرض Private واجزای استراکت به طور پیش فرض Public است.

```
1  struct Test
2  {
3  };
```

نمونه کد ۱۳۸: تعریف یک استراکت

struct in csharp ۲.۳.۱۳

استراکت ها با کلاس ها در این زبان متفاوت اند.

structیک Value Typeو کلاس هاReference Typeاند

```
1   struct Test
2     {
3         int a;
4         int b;
5         int c;
6     }
```

نمونه کد ۱۳۹: تعریف یک استراکت

این نکته شایان ذکر است که سایز کلی استراکت ها به اندازه سایز متغیر هایی خواهد بود که آن استراکت دارا می باشد.

در جلسات آتی در باره مفاهیمReference TypeوValue Typeبیشتر خواهیم دانست

جلسه ۱۴

داده نوع های Value Type و Reference Type

مسعود گلستانه - ۱۳۹۹/۱/۱۸

جزوه جلسه ۱۴ام مورخ ۱۳۹۹/۱/۱۸ درس برنامه‌سازی پیشرفته تهیه شده توسط مسعود گلستانه.

۱.۱۴ تفاوت میان داده های reference type و value type در سی شارپ

سی شارپ داده نوع ها را بسته به چگونگی ذخیره مقادیرشان در حافظه به دو دسته تقسیم میکند :

1. `Value Type`
2. `Reference Type`

به بسیاری از انواع اولیه داده های ساخته شده در سی شارپ مانند؛ `int , float, double,` `char, struct` (و نه رشته، به دلیلی که در ادامه گفته خواهد شد) `Value type` گفته می‌شوند. این تایپ ها مقداری ثابت دارند و هنگامی که شما متغیری از این نوع ایجاد می‌کنید، کامپایلر کدی را تولید می‌کند که یک بلوک از حافظه را، که به اندازه کافی برای نگه داشتن مقدار متناظر با آن بزرگ است، به آن اختصاص می‌دهد. به عنوان مثال، اعلام متغیر int باعث می‌شود کامپایلر ۴ بایت حافظه (۳۲ بیت) برای نگه داشتن مقدار عدد صحیح اختصاص دهد. در واقع حکمی که صادر می‌کند باعث می‌شود مقدار آن(مانند ۴۲) در این بلوک حافظه کپی شود.

از سویی با کلاس ها در سی شارپ به نحو متفاوتی رفتار می‌شود. هنگامی شما متغیری از نوع کلاس ایجاد می‌کنید، کدی که کامپایلر برای آن تولید می‌کند، بلوکی به بزرگی آن در حافظه ایجاد نمی‌کند بلکه فقط مقدار کمی از حافظه برای ذخیره سازی آدرس آن (به بلوک دیگری که حاوی آن است) اختصاص داده می‌شود. . (آدرس موقعیت مکانی آن چیزرا در حافظه مشخص می کند.) حافظه واقعی برای شی فقط زمانی اختصاص می یابد که از کلمه کلیدی new برای ایجاد آن استفاده شود. کلاس نمونه ای از نوع رفرنس است.

داده نوع های زیر همگی Reference Type هستند :

- رشته
- تمام آرایه ها، حتی اگر مقادیر آنها از نوع value type باشد
- کلاس
- Delegate

مهم:
توجه داشته باشید رشته در سی شارپ در واقع یک کلاس است. این امر به این دلیل است که هیچ اندازه‌ی استانداردی برای یک رشته وجود ندارد (رشته های مختلف می توانند شامل تعداد مختلفی از کاراکترها باشند) و تخصیص حافظه برای یک رشته به صورت پویا هنگام اجرای برنامه بسیار کارآمدتر از انجام این کار بصورت استاتیک در زمان کامپایل است[**MicrosoftVisualCsStepbyStep**].

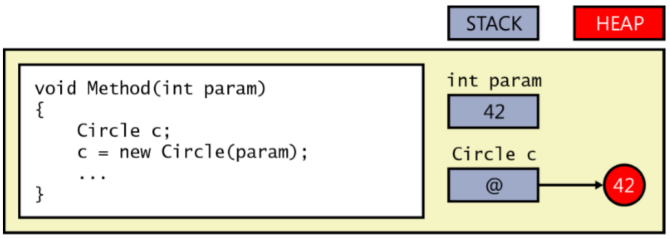
۱.۱.۱۴ فهم stack و heap

به نمونه کد سی شارپ زیر توجه کنید:

```Csharp
public class Circle
{
    //some property
}
void Test(int param)
{
Circle c;
c = new Circle(param);
...
}
```

Csharp ۱۴۰: کد نمونه

فرض کنید مقدار منتقل شده به پارام ۴۲ است. وقتی متد تست فراخوانی می شود ، یک بلوک حافظه (به اندازه لازم int) از پشته اختصاص داده می شود و با مقدار ۴۲ مقدار دهی اولیه می شود. با اجرای داخل متد، خط تعریف متغیر دایره c ، بلوک دیگری از حافظه که به اندازه کافی بزرگ است نیز برای نگه داشتن یک رفرنس (یک آدرس حافظه) از پشته اختصاص داده می شود اما بدون مقدار دهی اولیه باقی می ماند. در مرحله بعد ، یک قطعه دیگر از حافظه به اندازه کافی بزرگ برای یک شیء دایره از پشته اختصاص می یابد. این همان کاری است که کلمه کلیدی `new` انجام می دهد. کانستراکتور دایره تلاش می کند تا این حافظه پشته خام را به یک شیء دایره تبدیل کند. رفرنس به این شیء دایره در متغیر c ذخیره می شود. تصویر ۱.۱۴ این وضعیت را نشان می دهد.



شکل ۱.۱۴: تخصیص حافظه

- در این جا باید به دو نکته توجه داشته باشید:

۱. اگرچه شیء در هیپ ذخیره می شود ، اما رفرنس به شیء در پشته ذخیره می شود.

۲. حافظه‌ی هیپ بی نهایت نیست. اگر حافظه هیپ پر شود ، اپراتور `new` استثناء `OutOfMemoryException` را پرتاب می کند و شیء ایجاد نمی شود.

مهم:
برای بسیاری از توسعه دهندگان (مانند توسعه دهندگان زبان های مدیریت نشده C / C ++)، اصطلاحات reference type و value type در ابتدا عجیب به نظر می رسد. در C / C ++ ، شما یک تایپ را ایجاد می‌کنید ، و سپس کدی که از آن تایپ استفاده می کند تصمیم می گیرد که آیا متغیر نمونه از این تایپ را باید در پشته یا هیپ برنامه ذخیره کند. در حالیکه در زبان های مدیریت شده (managed) مانند سی شارپ برنامه نویسی که تایپ را تعریف می‌کند، مشخص می‌کند که نمونه هایی از آن تایپ در کجا ذخیره می‌شوند. برنامه نویسی که از آن تایپ استفاده می‌کند ، هیچ کنترلی بر این امر ندارد. **[CLRviaC]**

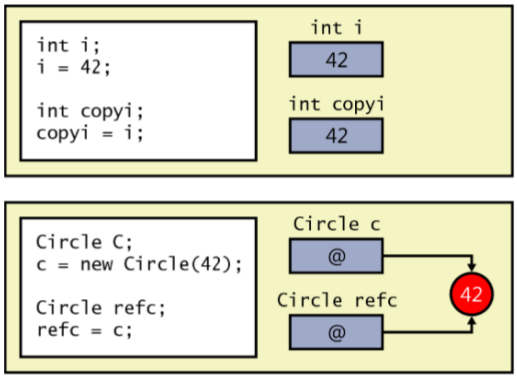
۲.۱۴ کپی سطحی (shallow copy)

وضعیتی را در نظر بگیرید [نمونه کد ۲۳۰]که در آن یک متغیر به نام i به عنوان int تعریف کنید و مقدار آن را ۴۲ اختصاص دهید. اگر متغیر دیگری به نام Copyi را به عنوان int اعلام کنید و سپس i را به Copyi اختصاص دهید ، Copyi همان مقدار i را نگه می دارد (۴۲). با وجود اینکه Copyi و i مقدار یکسانی دارند، دو بلوک حافظه جداگانه این مقادیر را نگه می‌دارند: یکی بلوک برای i و بلوک دیگر برای .Copyi اگر مقدار i را تغییر دهید ، مقدار Copyi تغییر نمی کند.

```Csharp
int i = 42;         // declare and initialize i
int copyi = i;      /* copyi contains a copy of the data in i and,
                       both contain the value 42 */
i++;               /* incrementing i has no effect on copyi;
                       i now contains 43, but copyi still contains 42 */

Circle c = new Circle(42);
Circle refc = c;
```

Csharp ۱۴۱: کد نمونه

تأثیر اعلام متغیر c به عنوان نوع کلاس، مانند دایره ، بسیار متفاوت است. هنگامی که شما c را به عنوان یک دایره اعلام می کنید ، c می تواند به یک شیء دایره مراجعه کند. مقدار واقعی نگهدارنده c آدرس یک شیء دایره در حافظه است. اگر متغیر دیگری به نام refc (همچنین به عنوان یک دایره) اعلام کنید و c را به refc اختصاص دهید ، refc یک کپی از همان آدرس c را در اختیار شما قرار می دهد. به عبارت دیگر همانطور که در شکل ۲.۱۴، فقط یک شیء دایره وجود دارد ، و هم اکنون refc و c به آن اشاره دارند.[**MicrosoftVisualCsStepbyStep**]



شکل ۲.۱۴: کپی سطحی

۳.۱۴ باکسینگ و آنباکسینگ

value type ها از وزن کمتری نسبت به رفرنس تایپ ها برخوردار هستند زیرا به عنوان اشیاء موجود در هیپ شناخته نشده، توسط garbage collector جمع آوری نمی‌شوند و پوینتری به آنها اشاره نمی‌کند. با این حال در مواردی، شما باید به نمونه ای از یک value type اشاره کنید.

۱.۳.۱۴ باکسینگ

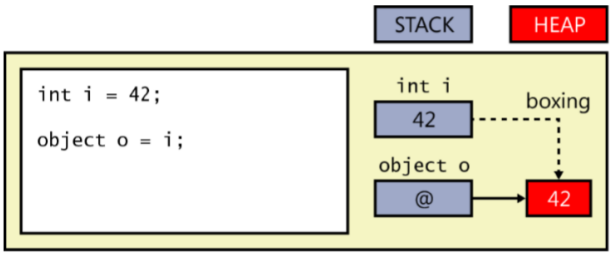
به عنوان مثال ، در عبارت زیر متغیر i (از نوع int ، یک value type) با ۴۲ و سپس متغیر o (از نوع شیء ، یک نوع رفرنس) را با i مقداردهی اولیه می کنیم:

```Csharp
int i = 42;
object o = i;
```

کد نمونه ۱۴۲: Csharp

همانطور که میدانیم i یک value type است و روی پشته قرار دارد. اگر رفرنس داخل o مستقیماً به i اشاره کند ، رفرنس به پشته ارجاع می شود. اما ، همه رفرنس ها باید به اشیاء موجود در هیپ اشاره کنند. ایجاد رفرنس به موارد موجود در پشته می تواند استحکام زمان اجرا را به خطر بیاندازد و یک نقص امنیتی بالقوه ایجاد کند ، بنابراین این کارمجاز نیست. درنتیجه، ران تایم بخشی از حافظه را از هیپ برای این کار

اختصاص می دهد ، مقدار عدد صحیح i را به این قطعه حافظه کپی می کند و سپس شیء o را به این نسخه ارجاع می دهد. به این کپی کردن خودکار یک چیز از پشته تا هیپ ، `boxing` گفته می شود. نمودار زیر نتیجه را نشان می دهد:



شکل ۳.۱۴: باکسینگ

مهم :
اگر مقدار اصلی متغیر i را تغییر دهید ، مقدار موجود در هیپ ارجاع شده از طریق o تغییر نخواهد کرد. به همین ترتیب ، اگر مقدار روی هیپ را تغییر دهید ، مقدار اصلی متغیر تغییر نخواهد کرد.

۲.۳.۱۴ آنباکسینگ

برای به دست آوردن مقدار نسخه باکس شده ، باید از کست استفاده کنید. این عملی است که بررسی می کند که تبدیل یک مورد از یک نوع به نوع دیگر قبل از تهیه نسخه کپی ایمن است. متغیر شی را با نام تایپ مورد نظر در پرانتزها پیش تعریف می کنید.

```
۱ int i = 42;
۲ object o = i; // boxes
۳ i = (int)o;  //unboxes
```

تأثیر این کست بسیار ظریف است. کامپایلر متوجه می شود که شما نوع int را در کست مشخص کرده اید. در مرحله بعد ، کامپایلر کدی را تولید می کند تا بررسی کند که در واقع در زمان اجرا به چه چیزی اشاره دارد. میتونه کاملا هر چیزی باشه فقط به این دلیل که کست شما می گوید o به یک int اشاره دارد ، این بدان معنا نیست که در واقع این کار را می کند. اگر واقعاً به یک int باکس شده اشاره کرده و همه چیز مطابقت

داشته باشد ، کست موفق می شوند و کد تولید شده توسط کامپایلر مقدار را از جعبه داخلی خارج می کند و آن را در i کپی می کندو در غیر این صورت اکسپشن InvalidCastException رخ می‌دهد.

۴.۱۴ Nullable ها در سی شارپ

مقدار null برای مقداردهی رفرنس تایپ ها به کار می‌رود . و یک value type نمی‌تواند مقدار null را در خود نگه دارد. برای مثال `int i = null` باعث می شود که در زمان اجرا با خطا روبرو شوید. در سی شارپ نسخه ۲ انواع nullable معرفی شدند که اجازه می‌دهند مقدار null را به یک متغیر type value انتساب داد. شما می توانید یک نوع nullable را با استفاده از عبارت Nullable<t> که t در آن یک نوع است را تعریف کنید :

```
۱ Nullable<int> i = null;
```

یک نوع nullable علاوه بر این که دارای محدوده ی داده ای نوع مورد نظر خود است می تواند مقدار null را هم در خود نگه دارد. برای مثال Nullable<int> علاوه بر اینکه می‌تواند مقداری بین ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را در خود نگه دارد، می‌تواند مقدار null را نیز در خود ذخیره کند. انواع Nullable نمونه ای از ساختار System.Nullable<T> هستند.

میتوانید از عملگر '؟' به شکل int؟ و long؟ به جای Nullable<T> برای تعریف متغیر های Nullable استفاده نمایید :

```
۱ int? i = null;
۲ double? D = null;
```

متغیر های i و j با تعریف زیر را در نظر بگیرید:

```
int? i = null;
int j = 99;
```

باید توجه داشته باشید که شما نمی توانید یک متغیر nullable را به یک متغیر از نوع معمولی اختصاص دهید. بنابراین ، با توجه به تعاریف متغیرهای i و j در مثال ، کد زیر مجاز نیست:

```
j = i;  //ilegall
```

اما برعکس آن صحیح است و کاربرد های زیادی در برنامه ها دارد:

```
i = j; //legall
```

جلسه ۱۵

Exceptions

یاسمن توکلی - ۱۳۹۹/۱/۲۳

۱.۱۵ exception چیست؟

exception نوعی خطا یا مشکل است که برنامه هنگام بیلد و اجرا به آن برمی خورد. لیست exception های متداول را در لینک زیر ببینید.

Exception List

برای مثال در برنامه زیر باید تعدادی عدد از ورودی دریافت شود و آنها چاپ شوند. اما اگر به جای عدد به آن رشته حرفی بدهیم با exception مواجه میشویم.

```
۱  public class Program
۲  {
۳      static void Main(string[] args){
۴          string number = Console.ReadLine();
۵          int count = int.Parse(number);
۶          for (int i = 0; i < count; i++)
۷          {
۸              System.Console.WriteLine(i);
۹          }
۱۰     }
۱۱ }
```

نمونه کد ۱۴۳: نمونه یک exception

ازexception به منظور handling error استفاده می کنیم. عبارت exception unhandled هنگامی رخ می دهد که خطایی در قسمتی از برنامه وجود داشته باشد و ما آن را تصحیح نکرده باشیم. هرexception حاوی پیامی است که به طور خلاصه منشاء مشکل را به ما نشان می دهد. با جست و جو در کد و اینترنت می توانیم خطا را رفع کنیم. داکیومنت های سایت مایکروسافت نیز برای پی بردن به اینکه هر دستور(مثلا در اینجاint.Parse) چه ویژگی هایی دارد و چه exception هایی می تواند بدهد، مفید هستند.

۲.۱۵ رفع exception

برای رفع این مشکل و جلوگیری ازcrash شدن برنامه، میتوانیم اقداماتی انجام دهیم. قسمتی از برنامه را که در آن ممکن است exception رخ دهد در بلوک try قرار می دهیم. بعد از آن، قطعه کدی را که اگر برنامه به خطایی در try برخورد اجرا شود را در بلوک catch قرار می دهیم. همانطور که در کد زیر مشاهده می کنید معمولا در پرانتزی جلوی catch نوع ارور را مشخص می کنند و و در خود بلوک از آن استفاده می کنند.

```
using System;

namespace ErrorHandlingApplication {
    class DivNumbers {
        int result;
        DivNumbers(){
            result = 0;}
        public void division(int num1, int num2) {
            try {
                result = num1 / num2;}
            catch (DivideByZeroException e) {
                Console.WriteLine({0}" caught: "Exception, e);}
            finally {
                Console.WriteLine({0}" "Result:, result);}
            }
        static void Main(string[] args) {
            DivNumbers d = new DivNumbers();
            d.division(25, 0);
            Console.ReadKey();
        }
    }
}
```

نمونه کد ۱۴۴: نمونه یک exception

در قطعه کد بالا چون ۲۵ بر ۰ تقسیم شده، برنامه به جای crash شدن، ابتدا وارد try شده و بعداز آنکه با خطا مواجه شد وارد catch می گردد و ارور DividedByZeroException را چاپ می کند.(e یک متغیر است که به جای DividedByZeroException از آن استفاده می شود.)

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
  at ErrorHandlingApplication.DivNumbers.division (System.Int32 num1, System.Int32 num2)
Result: 0
```

شکل ۱.۱۵: Exception Zero By Divided

در نهایت کد درون بلوک finally اجرا می شود. کد این بلوک بدون توجه به اینکه exception رخ داده یا نداده یا throw شده باشد، اجرا می گردد. مثلا اگر شما فایلی را باز کنید این فایل در نهایت چه exception داشته باشد چه نداشته باشد، باید بسته شود. شما می توانید exception های خودتان را با استفاده از کلید واژه throw به وجود آورید. User Defined Exception

برای مثال، در کد زیر کلاس exception تعریف شده در صورت شرایط خاص مثلا در اینجا اگر اسم دانش آموز null باشد، یک exception پرتاب می کند. توجه شود که باید حتما جلوی کلاس اکسپشن، Exception قرار داده شود:

```
۱   class Student
۲   {
۳       public void StudentName(string studentName){
۴           if (studentName == null)
۵               throw new InvalidStudentNameException(
۶               Invalid!" "Name);
۷       }
۸   }
۹   class InvalidStudentNameException : Exception
۱۰  {
۱۱      public string StudentName;
۱۲      public InvalidCourseIdException(string msg, string StudentName)
۱۳          : base(msg)
۱۴      {
۱۵          this.StudentName = StudentName;
۱۶      }
۱۷  }
```

نمونه کد ۱۴۵: Exception Class

```
class Program
{
    static void Main(string[] args)
    {
        Student newStudent = null;

        try
        {
            newStudent = new Student();
            newStudent.StudentName = "James007";

            ValidateStudent(newStudent);
        }
        catch(InvalidStudentNameException ex)
        {
            Console.WriteLine(ex.Message );
        }


        Console.ReadKey();
    }
    private static void ValidateStudent(Student std)
    {
        foreach (char c in std.StudentName)
        {
            if (std.StudentName == "0")
                throw new InvalidStudentNameException(std.StudentName);
        }
    }
}
```

نمونه کد ۱۴۶: user defined exception

با توجه به اینکه اسم ما دارای کاراکتر"۰" می باشد، یک exception پرتاب می شود:

```
Invalid Student Name: James000
```

شکل ۲.۱۵: exception invalid

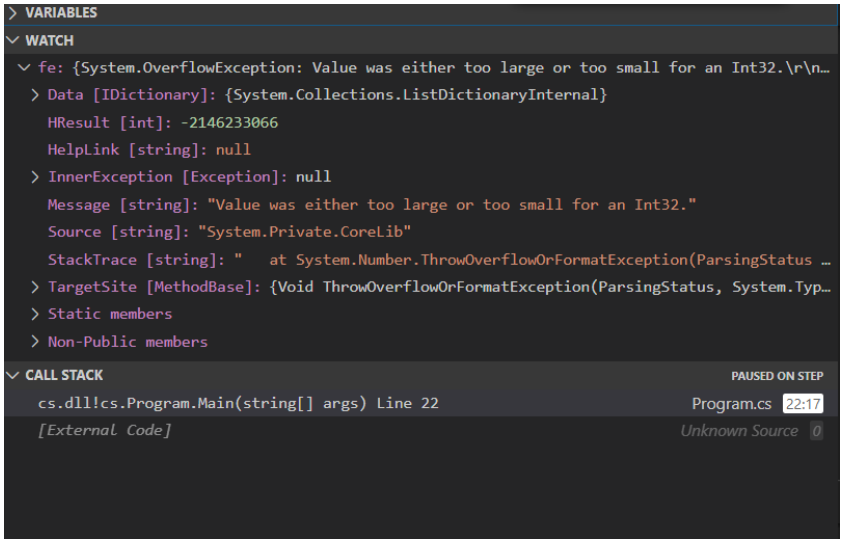
۳.۱۵ نکته ۱

بعضی متغیرها مثل int و double و bool به خودی خود Nullable نیستند. مثلا int count = null به ارور برمی خورد. برای همین از nullable value types استفاده می کنیم مثلا int؟ یا double؟

```
using System;
static void Main()
{
    int? count = null;
    do
    {
        try
        {
            Console.Write(" integer> "Input);
            string countString = Console.ReadLine();
            count = int.Parse(countString);
        }
        catch (OverflowException)
        {
            Console.WriteLine(again." Try integer. "Invalid);
        }
    }
    while(count == null);
    for (int i=0; i<count; i++)
        Console.WriteLine(i);
}
```

نمونه کد ۱۴۷: throwing user defined exception

با ورود یک عدد بسیار بزرگ مثل ۹۹۹۹۹۹۹۹۹۹، به ارور overflow exception برمی خوریم. دقت کنید هنگام debug هر exception حاوی یک پیام است. که این پیام علت خطا را بیان می کند. HResult هم کد این exception می باشد که میتوان آن را در اینترنت جست و جو کرد.



شکل ۳.۱۵: exception message

```
at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)\r\n
at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)\r\n
at System.Int32.Parse(String s)\r\n   at cs.Program.Main(String[] args) in c:\\git\\AP98992\\Tests\\cs\\Program.cs:line 18"
```

شکل ۴.۱۵: Stack Trace

عبارت Stack Trace نیز مسیر و خطی که exception در آن رخ داده است را نشان می دهد.

۴.۱۵ نکته ۲

می توانید درون بلوک do while نیز از try و catch استفاده کنید و تا مادامی که شرط while برقرار بود exception گرفته می شوند.

```
using System;
static void Main()
{
    int? count = null;
    do
    {
        try
        {
            Console.Write(" integer> "Input);
            string countString = Console.ReadLine();
            count = int.Parse(countString);
        }
        catch (OverflowException)
        {
            Console.WriteLine(again." Try integer. "Invalid);
        }
    }
    while(count == null);
    for (int i=0; i<count; i++)
        Console.WriteLine(i);
}
```

نمونه کد ۱۴۸: While Do Example

۵.۱۵ try/catch vs if/else

در برخی موارد به جای else/if میتوان از if برای چک کردن وجود خطای احتمالی استفاده کرد. در برنامه زیر دقت کنید که با دو روش exception handling انجام شده است:

```
try/catch using exception IndexOutOfRange //
int[] array = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int j = Convert.ToInt32(Console.ReadLine());
try
{
    int i = array[j];
}
catch (IndexOutOfRangeException)
{
    Console.WriteLine(range" of out "Index);
}
if/else using exception IndexOutOfRange //
int[] array = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int j = Convert.ToInt32(Console.ReadLine());
if (array.Length > j && j > -1)
{
    int i = array[j];
}
else
{
    Console.WriteLine(range" of out "Index);
}
```

نمونه کد ۱۴۹: catch/try vs else/if

به طور کلی همانطور که از اسم آن مشخص است، exception در شرایط استثنائی مثلا مواردی که برنامه از حالت عادی خود خارج شده و به مشکل خاصی برمی خورد، ایجاد می شود. معمولا از if/else برای اجرای یک شرط خاص بهره گرفته میشود که لزوما برای جلوگیری از crash شدن برنامه و handling error نیست. پرتاب کردن یک exception و استفاده از try/catch به دلیل طبقه بندی و جمع آوری اورها در یک جا، باعث راحتی کار ما در بررسی تمام شرایطی که درآن برنامه به خطا برمی خورد می شود. نکته دیگری که باید به آن توجه کرد این است که در بلوک try کدی نوشته می شود که احتمال خطا و پیش آمدن شرایط خاص برایش وجود داشته باشد و در catch سعی می شود روند عادی برنامه حفظ شود. علت دیگر استفاده از try/catch وجود Stack Trace و نشان دادن مسیری که خطا در آن رخ داده است می باشد. پرتاب کردن یک exception جزئیات بیشتری درباره آن خطا به ما ارائه می کند. به علاوه از catch/try برخلاف else/if به طور خاص برای رفع خطا استفاده می شود.

۶.۱۵ throwing using else/if

به دو مثال زیر دقت کنید. اگر ما به جای استفاده از پرتاب exception دلخواه از دستی رفع کردن خطا استفاده کنیم، در مقیاس بزرگتر تعداد پیام های خطایی که باید بنویسیم بسیار زیاد و پیچیده خواهد شد. مثلا اگر قرار باشد برای ثبت نام هر دانشجو علاوه بر درس ها، شماره دانشجویی و موارد دیگر را چک کنیم، باید برای هر متد

یک string منحصر به فرد تعریف کنیم. اگر هم متد از نوع bool باشد، متوجه نخواهیم شد که علت return false متد دقیقا کدام ارور خواهد بود. به علاوه امکان اینکه بعضی خطاها را یادمان برود رفع کنیم زیاد است:

```
private List<int> Coruses = new List<int>();
public string AddCourse(int courseId){
    if (courseId < 100 || courseId >= 1000)
    {
        return Invalid!" Is ID "Course
    }
    Coruses.Add(courseId);
    return null;
}
```

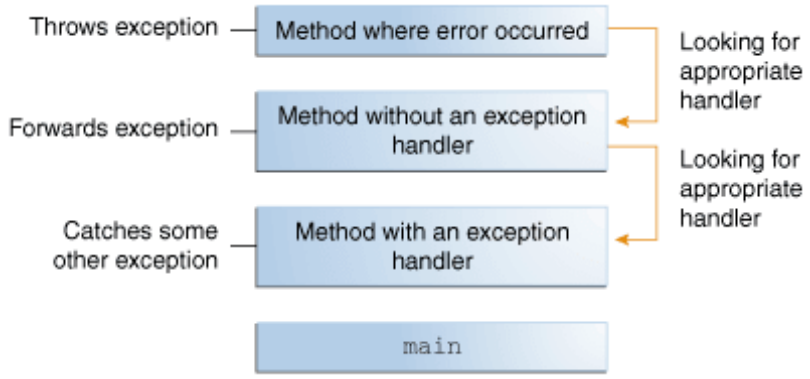
نمونه کد ۱۵۰: throwing exception using else/if and return

```
private List<int> Coruses = new List<int>();
public void AddCourse(int courseId)
{
    if (courseId < 100 || courseId >= 1000)
        throw new InvalidCourseIdException
        ($"{courseId}" range: of out "courseId, courseId);
    Coruses.Add(courseId);
}
```

نمونه کد ۱۵۱: throwing exception using exception throw

۷.۱۵ Call Stack and Re-Throwing Exceptions

در برخی موارد وقتی یک exception ظاهر شده یا به اصطلاح پرتاب می شود و سپس گرفته (catch) می شود، نیاز به دوباره پرتاب شدن(re-throw) دارد تا با متدهایی که در لایه های زیرین stack call قرار دارند و exception handling دارند، برطرف شود. به مثال عکس زیر توجه کنید:



شکل ۵.۱۵: call stack re-throwing exception

تصویر بالا لایه های حافظه stack و نحوه رفع خطا را نشان می دهد. ابتدا یک exception در متدی به وجود می آید. اگر درآن متد throw قرار داده باشیم، برنامه بعد از رسیدن به آن کد در حافظه استک به لایه پایینی یا به بیانی دیگر، متد قبلی در آن رجوع می کند. اگر در این متد exception برطرف شده باشد، برنامه به حالت عادی اجرای خود ادامه می دهد و ممکن است پیام خطایی در صفحه ظاهر شود اما اگر exception در متد قبلی handle نشده باشد و درون بلوک catch، کلید واژه throw قرار گیرد برنامه دوباره همین روند را با متد پایین تر در حافظه stack انجام می دهد تا زمانیکه همه خطاها برطرف شده باشند و برنامه به روند عادی خود ادامه می دهد. در واقع اگر بتوانیم کیس را در بلوک catch/try یا else/if برنامه handle می کنیم، اگر نشد آن را پرتاب می کنیم.

۸.۱۵ تفاوت throw با throw new exception

تفاوت عمده این دستور مسیر خطا یا exception است که stack trace نشان می دهد.

```
namespace c10cs
{
    class Student
    {
        private List<int> Coruses = new List<int>();
        public static void RegisterStudent
        (string name, int id, int[] courseIds)
        {
            Student s = new Student(name, id);
            foreach(var courseId in courseIds)
            {
                try
                {
                    s.AddCourse(courseId);
                }
                catch(CourseNotOfferedException ide)
                {
                    throw;
                }
            }
        }
        public void AddCourse(int courseId)
        {
            if (courseId < 100 || courseId >= 1000)
                throw new InvalidCourseIdException(
                ${courseId}" range: of out "courseId, courseId);

            EnsureCourseIsOffered(courseId);
            Coruses.Add(courseId);
        }
        void EnsureCourseIsOffered(int courseId)
        {
            if (courseId > 500)
                throw new CourseNotOfferedException(
                $semester" this offered not is "Course, courseId);
        }
    }
}
```

نمونه کد ۱۵۲: throw new exception vs throw

در برنامه بالا اگر courseId مثلا ۵۰۱ باشد، خطای CourseNotOfferedException پرتاب می شود و در call stack سراغ اولین متدی می رود که این خطا در آن catch شود. اما نکته مهم آنست که در trace stack محل به وجود آمدن خطا را متدی که در آن throw new exception داشته باشد، یعنی EnsureCourseIsOffered نشان می دهد. اما اگر خطای مورد نظر ما رفع نشود- مثلا اگر courseId عدد ۱۰ باشد و throw new exception نشده باشد- دوباره وارد اولین متد قبل خود که catch دارد می شود. اگر خطا handle نشده باشد، دوباره throw می شود و برنامه به اجرای خود ادامه می دهد. با این تفاوت که در stack trace محل به وجود آمدن خطا را همان جایی که اول در آن ظهور کرده یعنی AddCourse نشان می دهد.

جلسه ۱۶

Exceptions & Operators

بیان دیوانی آذر - ۱۳۹۹/۱/۲۵

۱.۱۶ Exceptions

درجلسه‌ی پیش در رابطه با کاربرد Exception ها که چگونه باعث آسان‌شدن Error handling می‌شوند، گفته‌شد. این جلسه نحوه ی رخ دادن Exception ها را میبینیم.

```
static void Main(string[] args)
{
    Console.WriteLine(Before" - Main "In);
    MethodA();
    Console.WriteLine(After" - Main "In);
}
```

نمونه کد ۱۵۳: تابع Main که در آن تابع MethodA صدا زده می‌شود.

```
private static void MethodA()
{
    Console.WriteLine(Before" - A "In);
    MethodB();
    Console.WriteLine(After" - A "In);
}
```

نمونه کد ۱۵۴: تابع MethodA که در آن تابع MethodB صدا زده می‌شود.

```
private static void MethodB()
{
    Console.WriteLine(Before" - B "In);
    MethodC();
    Console.WriteLine(After" - B "In);
}
```

نمونه کد ۱۵۵: تابع MethodB که در آن تابع MethodC صدا زده می‌شود.

```
private static void MethodC()
{
    Console.WriteLine(Before" - C "In);
    MethodD();
    Console.WriteLine(After" - C "In);
}
```

نمونه کد ۱۵۶: تابع MethodC که در آن تابع MethodD صدا زده می‌شود.

```
private static void MethodD()
{
    Console.WriteLine(Before" - D "In);
    Console.WriteLine(After" - D "In);
}
```

نمونه کد ۱۵۷: تابع MethodD بدون اکسپشن(حالت عادی)

```
private static void MethodD()
{
    Console.WriteLine(Before" - D "In);
    throw new InvalidOperationException();
    Console.WriteLine(After" - D "In);
}
```

نمونه کد ۱۵۸: تابع MethodD با اکسپشن

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In D - After
In C - After
In B - After
In A - After
In Main : After
```

خروجی برنامه ۱: در حالت عادی

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
```

خروجی برنامه ۲: در حالی که به تابع MethodD اکسپشن اضافه شده است

همینطور که می‌بینید در اینجا به خاطر پرت شدن Exception در خط ۴ام در MethodD بقیه برنامه اجرا نمی‌شود.

```
1  static void Main(string[] args)
2  {
3      Console.WriteLine(Before" - Main "In);
4      try
5      {
6          MethodA();
7      }
8      catch
9      {
10         Console.WriteLine(Catch" - Main "In);
11     }
12     Console.WriteLine(After" - Main "In);
13 }
```

نمونه کد ۱۵۹: تابع Main که به آن (try - catch) اضافه شده‌است.

همینطور که می‌بینید در اینجا بعد از پرتاب شدن اکسپشن به خط دهم تابع Main ۱۰ میرویم و بعد از خارج شدن از بلاک catch خط دوازدهم ۱۲ اجرا میشود و برنامه تمام می‌شود

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In Main - Catch
In Main - After
```

خروجی برنامه ۳: در حالی که به تابع Main (try - catch) اضافه شده‌است.

```
۱  private static void MethodB()
۲  {
۳      Console.WriteLine(Before" - B "In);
۴      try
۵      {
۶          MethodC();
۷      }
۸      catch
۹      {
۱۰         Console.WriteLine(Catch" - B "In);
۱۱         throw;
۱۲     }
۱۳     Console.WriteLine(After" - B "In);
۱۴ }
```

نمونه کد ۱۶۰: تابع MethodB که به آن (try - catch) اضافه شده‌است.

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In B - Catch
In Main - Catch
In Main - After
```

خروجی برنامه ۴: در حالی که به تابع MethodB (try - catch) اضافه شده‌است.

همینطور که می بینید در اینجا بعد از پرتاب شدن اکسپشن به خط دهم تابع MethodB ۱۰ میرویم و بعد دوباره اکسپشن رو پرتاب میکنیم و به اینجا ۱.۱۶ می‌رویم.

```
private static void MethodB()
{
    Console.WriteLine(Before" - B "In);
    try
    {
        MethodC();
        Console.WriteLine(C" Calling After - B "In);
    }
    catch
    {
        Console.WriteLine(Catch" - B "In);
    }
    finally
    {
        Console.WriteLine(Finally" - B "In);
    }
    Console.WriteLine(After" - B "In);
}
```

نمونه کد ۱۶۱: تابع MethodB که به آن بلاک (finally) اضافه شده‌است.

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In D - After
In C - After
In B - After Calling C
In B - Finally
In B - After
In A - After
In Main - After
```

خروجی برنامه ۵: در حالی که به تابع MethodB بلاک (finally) اضافه شده‌است. و در MethodD اکسپشن پرتاب نشده‌است. (حالت عادی)

خب در اینجا بلاک جدیدی به اسم finally می‌بینید، کد موجود در این بلاک به هر روی اجرا می‌شود، چه استثناء رخ دهد چه ندهد. (البته واضح است که اگر اکسپشن catch نشود، این بلاک اجرا نخواهدشد)

نکته ی جالب! حتی اگر ما قبل از ورود به بلاک finally برگردیم (return کنیم)، کد این بلاک اجرا خواهدشد.

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In B - Catch
In B - Finally
In B - After
In A - After
In Main - After
```

خروجی برنامه ۶: در حالی که به تابع MethodB بلاک (finally) اضافه شده‌است و در MethodD اکسپشن پرتاب شده‌است

همینطور که می بینید در اینجا بعد از پرتاب شدن اکسپشن به خط دهم تابع MethodB ۱۱ می‌رویم و بعد برنامه به صورت عادی اجرا می‌شود و تمام می‌شود.

```
private static void MethodB()
{
    Console.WriteLine(Before" - B "In);
    try
    {
        MethodC();
        Console.WriteLine(C" Calling After - B "In);
    }
    catch (OverflowException)
    {
        Console.WriteLine(Catch" - B "In);
    }
    finally
    {
        Console.WriteLine(Finally" - B "In);
    }
    Console.WriteLine(After" - B "In);
}
```

نمونه کد ۱۶۲: تابع MethodB که در آن بلاک (catch) فقط اکسپشن از نوع (OverflowException) catch می کند

کمکی! اگر با انواع اکسپشن‌ها آشنا نیستید، میتوانید با مراجعه به اینجا [**stackify**] در قسمت Common .NET Exceptions با انواع اکسپشن‌ها در سی شارپ آشنا شوید.

```
In Main - Before
In A - Before
In B - Before
In C - Before
In D - Before
In B - Finally
In Main - Catch
In Main - After
```

خروجی برنامه ۷: در حالی که تابع MethodB که در آن بلاک (catch) فقط اکسپشن از نوع catch (OverflowException) می‌کند.

همینطور که میبینید بعد از پرتاب شدن اکسپشن از بلاک try خارج می شویم و خط ۷ اجرا نمی‌شود و بعد وارد بلاک finally می‌شویم و بعد ۱.۱۶ می‌رویم.

```
۱  private static void MethodB()
۲  {
۳      Console.WriteLine(Before" - B "In);
۴      StreamReader reader = null;
۵      try
۶      {
۷          reader = new StreamReader("in.txt");
۸          string line = reader.ReadLine();
۹          MethodC();
۱۰         Console.WriteLine(C" Calling After - B "In);
۱۱     }
۱۲     catch (OverflowException)
۱۳     {
۱۴         Console.WriteLine(Catch" - B "In);
۱۵     }
۱۶     finally
۱۷     {
۱۸         reader.Dispose();
۱۹         Console.WriteLine(Finally" - B "In);
۲۰     }
۲۱     Console.WriteLine(After" - B "In);
۲۲ }
```

نمونه کد ۱۶۳: مثالی از کاربرد بلاک finally

در اینجا مطمئنیم استریم‌ریدر ما حتما Dispose می‌شود، حتی اگر اکسپشن پرتاب شود.

۲.۱۶ Indexers

با تعریف Indexer برای کلاس‌مان میتوانیم با استفاده از [ ] به کلاس همانند آرایه دسترسی پیدا کنیم.

```
public <return type> this[<parameter type> index]
{
get{
// return the value from the specified index
}
set{
// set values at the specified index
}
}
```

نمونه کد ۱۶۴: سینتکس Indexer

توجه کنید! شما برای getter باید به محدوده ی Indexer تان توجه کنید تا ناخواسته به اکسپشن بر نخورید.

```
class PBEntry
{
    public PBEntry this[string name]
    {
        get
        {
            return Data[name];
        }
        set
        {
            this.Data[name] = value;
        }
    }
}
```

نمونه کد ۱۶۵: مثالی از Indexer

همانطور که در قطعه کد زیر میبینید، ما میتوانیم Indexer رو برای struct ها نیز تعریف کنیم.

```
struct IntBits
{
public bool this[int index]
    {
        get
        {
            return (bits & (1 << index)) != 0;
        }
        set
        {
            if (value)
                bits |= (1 << index);
            else
                bits &= ~(1 << index);
        }
    }
}
```

نمونه کد ۱۶۶: قطعه کد نمونه بر گرفته از کتاب [**sharp۲۰۱۳microsoft**]

جلسه ۱۷

Operator Overloading

نیکی مجیدی فرد - ۱۳۹۸/۱/۳۰

جزوه جلسه ۱۷ام مورخ ۱۳۹۸/۱/۳۰
برنامه‌سازی پیشرفته

overloading operator

Operator Conversion ۱.۱۷

- implicit

```
public static operator implicit target-type(source-type v) { return value; }
```

- explicit

```
public static operator explicit target-type(source-type v) { return value; }
```

در اینجا target-Type نشان دهنده نوعی است که می خواهیم source-Type را به آن تبدیل کنیم و value مقدار کلاس پس از تبدیل است .
conversion operator برای تبدیل کردن یک نوع داده ای به نوع داده ای دیگر استفاده می شود و یک شی از کلاس شما را به نوع دیگری که می خواهید تبدیل می کند .

- تفاوت implicit و explicit

تفاوت این دو در به ترتیب غیر مستقیم و مستقیم cast کردن است .
در مثال پایین ما کلاسی داریم که دو property ساعت و دقیقه از نوع int , یک constructor و یک copy constructor دارد که در ان اپراتور + را overload کرده ایم به کد زیر جهت بقیه توضیحات توجه کنید :

```
    class Time
    {
        private int h;
        private int m;

        public Time(int h, int m)
        {
            this.h = h;
            this.m = m;
        }

        public Time(Time other)
        {
            this.h = other.h;
            this.m = other.m;
        }

        public void AddTo(Time t)
        {
            int newValue = t.m + this.m;
            this.m = newValue % 60;
            this.h = t.h + this.h + (newValue / 60);
        }

        public static Time operator+(Time lhs, Time rhs)
        {
            Time t = new Time(lhs);
            t.AddTo(rhs);
            return t;
        }
    }
```

.

implicit ۱.۱.۱۷

حال می خواهیم یک ساعت، به ساعت دیگری اضافه کنیم :

```
            Time t = new Time(12, 30);
            Time t2 = new Time(1, 0);
            Time t3 = t + t2;
```

در کد بالا یک نوع داده داریم و داده هایمان از نوع کلاس Time است. حالا به کد زیر توجه کنید :

```
Time t4 = t + 1;
```

آیا مجاز به انجام این کار هستیم؟ خیر.ما مجاز نیستیم یک داده از نوع کلاس Time را با داده ای از نوع int جمع کنیم ، می توانیم از implicit استفاده کنیم ; به کد زیر توجه کنید :

```
        public static Time operator+(Time lhs, Time rhs)
        {
            Time t = new Time(lhs);
            t.AddTo(rhs);
            return t;
        }

        public static implicit operator Time(int fromHour)
        {
            return new Time(fromHour, 0);
        }
```

ابتدا implicit نوع داده ای int را به Time تبدیل می کند سپس با استفاده از اپراتوری که از قبل تعریف کردیم می توانیم دو داده از نوع Time را با هم جمع کنیم پس با استفاده از این دو توانستیم یک داده از نوع int و داده ای از نوع Time را با هم جمع کنیم .

- همانطور که در ابتدا اشاره کردیم ما می توانیم هر نوع داده ای دیگری را نیز مثل string,float,... را هم به Time تبدیل کنیم برای مثال می خواهیم string هایی به قالب "hh:mm" را به Time تبدیل کنیم مانند مثال زیر :

```
            Time t7 = "12:30";
```

برای این کار می توانیم در ابتدا تابعی بنویسیم که قسمت ساعت و قسمت دقیقه string را از هم جدا کند ،سپس با استفاده از implicit نوع داده ای string را به Time تبدیل کنیم .

```
        public static Time Parse(string time)
        {
            int colPos = time.IndexOf(':');
            string hour = time.Substring(0, colPos);
            string minute = time.Substring(colPos+1);
            return new Time(int.Parse(hour), int.Parse(minute));
        }

        public static implicit operator Time(string time) => Parse(time);
```

- برای بهتر شدن مفهوم به کد زیر توجه کنید :

```
۱             Time t8 = t7 + "1:30" + 4;
```

با توجه به توابعی که در بالا نوشته بودیم ،آیا کد بالا کار می کند ؟ بله، به این صورت که در ابتدا string "12:30" به Time تبدیل شده سپس این داده که حالا از نوع Time است با t7 جمع می شود سپس ۴ که از نوع int است به Time تبدیل شده و با نوع داده ای Time دیگر جمع می شود .

explicit ۲.۱.۱۷

برای cast کردن مستقیم استفاده می شود .
به کد زیر توجه کنید :

```
۱ Time t9 = (Time) 5.0;
```

در این مثال می خواهیم داده ای از نوع double را مستقیما به نوع Time تبدیل کنیم پس می توانیم از explicit conversion استفاده کنیم :

```
۱         public static explicit operator Time(double time)
۲         {
۳             int hour = (int) time;
۴             int minute = (int) ((time - hour) * 60);
۵             return new Time(hour, minute);
۶         }
```

Pairs Operator ۲.۱۷

```
== !=

< >

<= >=
```

این نوع اپراتور ها باید به طور جفت پیاده سازی شوند و bool برمی گردانند .

۱.۲.۱۷ ==، !=

به کد زیر توجه کنید :

```
۱  public override bool Equals(object obj)
۲  {
۳      Time t = obj as Time;
۴      if(t != null)
۵          return t.h == this.h && t.m == this.m;
۶      return false;
۷  }
۸
۹  public static bool operator == (Time lhs,Time rhs)
۱۰ {
۱۱     return !lhs.Equals(rhs);
۱۲ }
۱۳
۱۴ public static bool operator !=(Time lhs,Time rhs)
۱۵ {
۱۶     return !lhs.Equals(rhs);
۱۷ }
```

operator != == پیادهسازی کرده ایم ولی کد بالا StackOverflowError می دهد . نحوه اجرا شدن کد بالا به این صورت است که ابتدا Equals در خط ۱۱ صدا می شود . در تابع Equals خط 4 ، t از نوع کلاس Time است پس بعد از این خط، خط ۱۴ و سپس ۱۶ اجرا می شود، بعد از خط ۱۶ که در آن باز Equals صدا زده می شود ،خط اول و دوباره خط ۴ اجرا می شود و همینگونه ادامه پیدا می کند .
برای جلوگیری از این مشکل می توانیم از ReferenceEquals استفاده کنیم و به جای صدا کردن اپراتور , reference شان را با هم مقایسه کنیم :

```
۱  public static bool ReferenceEquals (object objA, object objB);
```

همانطور که می بینید این Method اگر دو object مثل هم باشند true و در غیر این صورت false برمی گرداند .

```
Parameters
obj A Object
obj B Object


returns
Boolean
true if objA is the same instance as objB or if both are null;
otherwise, false.
```

با توجه به توضیحات داده شده، کد بالا را می توانیم به صورت زیر بنویسیم :

```
public static bool operator ==(Time lhs, Time rhs)
{
    if (! ReferenceEquals(lhs, null))
            return lhs.Equals(rhs);
     return false;
}

public static bool operator !=(Time lhs, Time rhs) => ! (lhs == rhs);
```

۲.۲.۱۷ =< ،=> and < ،>

pair operator بعدی > است که برای مثال می توانید به کد زیر توجه کنید :

```
 public static bool operator <(Time lhs, Time rhs)
    {
        if (lhs.h == rhs.h)
            return lhs.m < rhs.m;

        return lhs.h < rhs.h;
    }
```

در اینجا ساعت این دو شی را با هم مقایسه کردیم و در صورت برابر بودن ساعتشان به مقایسه دقیقه شان می پردازیم .
همانطور که گفتیم باید pair operator ها را جفت پیاده سازی کنیم ،پس اینجا > نیز پیاده سازی کنیم که می توانیم خروجی آن را به صورت `!(lhs < rhs)` بنویسیم که چون حالت `lhs == rhs` را هم در بر می گیرد این حالت را حذف کرده و نهایتا آن را به صورت زیر می توانیم بنویسیم :

```
public static bool operator >(Time lhs, Time rhs) =>
    ! (lhs < rhs) && lhs != rhs;
```

operators true/false ۳.۲.۱۷

برای pair operator بعدی می توانیم به `true false` اشاره کنیم و بیشتر برای وقتی استفاده می شود که بخواهیم چک کنیم، مثلا :

```
if (t9)
    t9 = t9  + 1;
```

حال اینجا به عنوان مثال

```
public static bool operator true(Time t)
{
    return t.h != 0 || t.m != 0;
}
public static bool operator false(Time t)
{
    if (t)
        return false;
    return true;
}
```

اپراتور true را طوری پیاده سازی کردیم که اگر ساعت یا دقیقه داده مان صفر نباشد،خروجی اش true خواهد بود و سپس اپراتور false را هم برای آن پیاده سازی کردیم .

[**visualizationwebsite**] [**CLRS**]

جلسه ۱۸

واسط ها

باوان دیوانی آذر - ۱۳۹۹/۲/۱

۱.۱۸ چالش ۱

فرض کنید ما کلاس `Student` ، مانند کد زیر را داریم :

```
۱  public class Student
۲  {
۳      public string Name {get; private set;}
۴      public int Id {get; private set;}
۵      public double GPA {get; private set;}
۶
۷      public Student(string name, int id, double gpa)
۸      {
۹          this.Name = name;
۱۰         this.Id = id;
۱۱         this.GPA = gpa;
۱۲     }
۱۳ }
```

نمونه کد ۱۶۷: تعریف کلاس `Student`

همچنین در تابع اصلی `Main` یک آرایه از جنس `Student` داریم ، میخواهیم این آرایه را بر حسب شماره دانشجویی `Id` و معدل `GPA` مرتب کنیم.

چالش: این کار را چگونه با نوشتن فقط یک تابع انجام بدهیم؟

ابتدا اینکار را با نوشتن دو تابع انجام می‌دهیم . بنابراین یک تابع برای مرتب کردن دانش آموزان بر حسب شماره دانشجویی و دیگری بر حسب معدل می نویسیم.

```
۱  private static void IdSort(Student[] students)
۲  {
۳          for (int i = 0; i < students.Length; i++)
۴              for (int j = i + 1; j < students.Length; j++)
۵                  if (students[i].Id < students[j].Id)
۶                      Swap(students, i, j);
۷  }
```

نمونه کد ۱۶۸: تعریف تابع `IdSort`

```
۱  private static void GPASort(Student[] students)
۲  {
۳          for (int i = 0; i < students.Length; i++)
۴              for (int j = i + 1; j < students.Length; j++)
۵                  if (students[i].GPA < students[j].GPA)
۶                      Swap(students, i, j);
۷  }
```

نمونه کد ۱۶۹: تعریف تابع `GPASort`

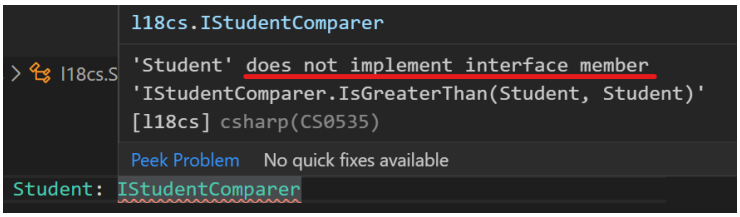
خب همانطور که می‌بینید این دو تابع تقریبا شبیه هم هستند و فقط در خط هفت با یکدیگر فرق دارند. در اینجا هست که واسط ها `Interfaces` به کمک ما می‌آیند . واسط ها مزایای زیادی دارند ، مثلا باعث می‌شوند که کدها قابلیت بهتری در نگهداری ، انعطاف پذیری و استفاده مجدد داشته باشند . همچنین یک کلاس می تواند همزمان از چند واسط ارث بری کند .

برای حل این چالش ابتدا یک واسط به نام `IStudentComparer` تعریف میکنیم ، برای تعریف واسط از کلید واژه `Interface` استفاده می‌کنیم .

```
۱ interface IStudentComparer
۲ {
۳     bool IsGreaterThan(Student s1, Student s2);
۴ }
```

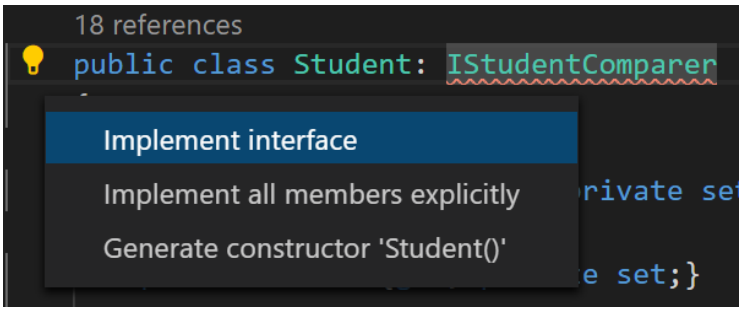
نمونه کد ۱۷۰: تعریف واسط `IStudentComparer`

سپس در کلاس `Student` از واسطمان ارث بری می‌کنیم . خب همانطور که می‌بینید با انجام این کار با یک خطا روبرو می‌شویم .



شکل ۱.۱۸: خطای شماره یک

برای رفع این خطا از لامپ زردی که در سمت چپ خطی که خطا را داریم قرار گرفته ، کمک می‌گیریم و با زدن گزینه `Implement interface` خطا رفع می‌شود .



شکل ۲.۱۸: رفع خطای شماره یک

خب همانطور که می‌بینید یک تابع به کلاسمان اضافه می‌شود .

```
public bool IsGreaterThan(Student other)
{
    throw new System.NotImplementedException();
}
```

نمونه کد ۱۷۱: تعریف تابع `IsGreaterThan`

اگر دقت کنید ما به یک مشکل برخورد می‌کنیم . ما برای حل چالشمان میخواستیم فقط یک تابع برای مرتب کردن آرایه بر حسب چند ویژگی بنویسیم ولی در این تابع ما فقط میتوانیم این مقایسه را بر حسب یک ویژگی انجام دهیم . پس برای حل این مشکل دو کلاس تعریف می کنیم که از واسط `IStudentComparer` ارث بری می‌کنند .

```
class StudentIdComparer: IStudentComparer
{
    public bool IsGreaterThan(Student s1, Student s2)
        => s1.Id < s2.Id;
}
```

نمونه کد ۱۷۲: تعریف کلاس `StudentIdComparer`

```
class StudentGPAComparer: IStudentComparer
{
    public bool IsGreaterThan(Student s1, Student s2)
        => s1.GPA < s2.GPA;
}
```

نمونه کد ۱۷۳: تعریف کلاس `StudentGPAComparer`

هم اکنون در کلاس `Program` ، تابع `Sort` یک متغیر از جنس `IStudentComparer` را به عنوان ورودی دریافت می‌کنیم . سپس از تابع `ISGreaterThan` واسطمان استفاده می‌کنیم .

```
private static void Sort(Student[] students, IStudentComparer cmp)
{
    for(int i=0; i<students.Length; i++)
        for(int j=i+1; j<students.Length; j++)
            if (cmp.IsGreaterThan(students[i], students[j]))
                Swap(students, i, j);
}
```

نمونه کد ۱۷۴: تعریف تابع `Sort`

برای صدا زدن تابع از دو کلاسی که برای مرتب کردن نوشته‌ایم ، استفاده می‌کنیم و یک شیء از کلاس مورد نظر را می‌سازیم و به تابعمان به عنوان ورودی می‌دهیم .

```
۱     Sort(students, new StudentGPAComparer());
۲     Sort(students, new StudentIdComparer());
```

نمونه کد ۱۷۵: صدا زدن تابع `Sort`

برای اینکه هربار یک شیء ازکلاس‌هایمان نسازیم ، می‌توانیم یک کلاس به نام `StudentComparer` تعریف کنیم و در آن دو ویژگی از جنس کلاس‌هایمان را تعریف کنیم .

```
۱  static class StudentComparer
۲  {
۳      public static StudentIdComparer StudentIdComparer = new StudentIdComparer();
۴      public static StudentGPAComparer StudentGPAComparer = new StudentGPAComparer();
۵  }
```

نمونه کد ۱۷۶: تعریف کلاس `StudentComparer`

حالا برای اطمینان از اینکه تابعمان درست کار می‌کند ، آن را تست می‌کنیم .

```
۱     Sort(students, StudentComparer.StudentGPAComparer);
۲     PrintStudents(students);
۳
۴     Sort(students, StudentComparer.StudentIdComparer);
۵     PrintStudents(students);
```

نمونه کد ۱۷۷: تست تابع `Sort`

```
۱  private static void PrintStudents(Student[] students)
۲  {
۳      foreach(var s in students)
۴          Console.WriteLine($" {s.GPA} {s.Id} {s.Name} ");
۵  }
```

نمونه کد ۱۷۸: تعریف تابع `PrintStudents`

۲.۱۸ چالش ۲

خب ، اگر تست کنید متوجه می‌شوید که تابعمان به درستی کار می‌کند ، حالا فرض کنید که کلاس `Teacher` داریم.

```
public class Teacher
{
    string Name;
    int Id;
    double Rating;

    public Teacher(string name, int id, double rating)
    {
        this.Name = name;
        this.Id = id;
        this.Rating = rating;
    }
}
```

نمونه کد ۱۷۹: تعریف کلاس `Teacher`

فرض کنید که یک آرایه از جنس `Teacher` و یک آرایه دیگر از جنس `Student` داریم و می‌خواهیم فقط با نوشتن یک تابع `Sort` بر اساس نام مرتبشان کنیم .

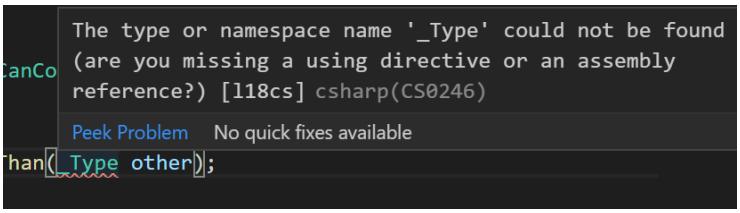
به نظرتان اگر بخواهیم در تابع `Sort` آرایه ای را به عنوان ورودی بگیریم ، ورودی دریافتی رو از چه جنسی باید تعریف کنیم ؟ آیا ما همچنان می توانیم از واسط `IStudentComparer` استفاده کنیم؟

یک واسط به نام `ICanCompare` را تعریف می‌کنیم .

```
public interface ICanCompare
{
    int IsGreaterThan(_Type other);
}
```

نمونه کد ۱۸۰: تعریف واسط `ICanComparer`

همانطور که می بینید به یک خطا برخورد می‌کنیم .



شکل ۳.۱۸: خطای شماره دو

دلیل این خطا این است که هر وقت می‌خواهیم از `_Type` استفاده کنیم باید کنار نام آن کلاس یا واسط یا تابع باید `<_Type>` را اضافه کنیم .

```
۱  public interface ICanCompare<_Type>
۲  {
۳          int IsGreaterThan(_Type other);
۴  }
```

نمونه کد ۱۸۱: تعریف واسط `ICanComparer`

خوشبختانه خطا رفع شد. الان باید کلاس های `Teacher` و `Student` این واسط را ارث بری کنند .

```
۱  public class Teacher : ICanCompare<Teacher>
۲  {
۳      string Name;
۴      int Id;
۵      double Rating;
۶
۷      public Teacher(string name, int id, double rating)
۸      {
۹          this.Name = name;
۱۰         this.Id = id;
۱۱         this.Rating = rating;
۱۲     }
۱۳     public int IsGreaterThan(Teacher other) => this.Name.CompareTo(other.Name);
۱۴ }
```

نمونه کد ۱۸۲: تعریف کلاس `Teacher`

```
public class Student : ICanCompare<Student>
{
    string Name;
    int Id;
    double GPA;

    public Student(string name, int id, double gpa)
    {
        this.Name = name;
        this.Id = id;
        this.GPA = gpa;
    }
    public int IsGreaterThan(Student other) => this.Name.CompareTo(other.Name);
}
```

نمونه کد ۱۸۳: تعریف کلاس `Student`

اکنون باید تابع `Sort` را بنویسیم .

```
private static void Sort<_Type>(_Type[] students)
where _Type:ICanCompare<_Type>
{
    for(int i=0; i<students.Length; i++)
        for(int j=i+1; j<students.Length; j++)
            if (students[i].IsGreaterThan(students[j])>0)
                Swap(students, i, j);
}
```

نمونه کد ۱۸۴: تعریف تابع `Sort`

خب حال نوبت به تست کردن تابعمان می‌رسد .

```
    Sort(teachers);
    Sort(students);
```

نمونه کد ۱۸۵: تست تابع `Sort`

خوشبختانه تابعمان به درستی کار می‌کند. همانطور که می‌بینید این راه کمی طولانی بود ، پس بزودی قرار است که با توجه به نکته زیر یک راه بسیار کوتاه‌تر به شما معرفی کنم .

۳.۱۸ نکات

نکته ۱: متغیر های آرایه و لیست ، خود یک تابع `Sort` دارند (اسم این تابع برای آرایه `Array.Sort` و برای لیست `Sort` است) و فقط لازم است که کلاسی را که می‌خواهیم شیءهایش مرتب شوند ، این واسط `IComparer` را ارث بری کرده باشند .

پس برای حل چالش ۲ ، فقط لازم بود کلاس های `Teacher` و `Student` از واسط `IComparer` ارث بری کنند.

نکته ۲: برای تعریف فیلد باید حتما از پراپرتی استفاده کنید .

۴.۱۸ خلاصه بندی

هر زمان چند کلاس زیرمجموعه چیزی باشند ، می‌توانیم از واسط استفاده کنیم .

مثال ۱ : فرض کنید که می‌خواهید برای هر یک از اشکال یک کلاس بنویسید . به همین خاطر می‌توانید یک واسط به نام `IShape` تعریف کنید و این واسط یه لیستی از جنس رأس بگیرد و تابع‌هایی برای بدست آوردن محیط و مساحت داشته باشد .

مثال ۲ : فرض کنید که می‌خواهید فضای بیمارستان را تعریف کنید . در نتیجه شما یک گروه بیمار دارید و انواع مختلف دکتر، به عنوان مثال گروهی جراح زیبایی هستند ، گروهی جراح مغز و غیره . برای انجام این کار می‌توانیم از دو واسط `IPerson` و `IDoctor` استفاده کنیم ، بطوریکه کلاس بیمار از واسط `IPerson` ارث بری کند و کلاس دکتر از `IDoctor` و `IPerson` ارث بری کند .

برای مشاهده مثال‌های بیشتر به مراجع [۱، ۲] مراجعه کنید .

تعدادی واسط هم خودشان تعریف شده‌اند و ما می‌توانیم از آنها ارث بری کنیم . ما در این جلسه با واسط `IComparer` آشنا شدیم و در جلسه بعدی قرار است با واسط‌های بیشتری مثل `IEnumerator` آشنا شویم .

۵.۱۸ تمرینات اضافی

برای تسط بیشتر می‌توانید تمرین‌های زیر را انجام دهید .

تمرین ۱: فرض کنید که کلاسی به نام `Point` دارید که سه ویژگی `X` و `Y` و `Manitude` را دارد ، همچنین یک آرایه از جنس این کلاس هم داریم . یک تابع `Sort` بنویسید که این آرایه را بر حسب سه ویژگی گفته شده مرتب کند .

تمرین ۲: سوالات و توضیحات در دو لینک [۳، ۴] وجود دارند ، پس شما سعی کنید تست کیس‌ها را پاس کنید .

جلسه ۱۹

Interface IEnumerable، IDisposable

آزاده دارابی مقدم - ۱۳۹۹/۲/۶

جزوه جلسه ۱۹ام مورخ ۱۳۹۹/۲/۶درس برنامه‌سازی پیشرفته تهیه شده توسط آزاده دارابی مقدم. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته ، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش‌مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید. مطالبی که در ادامه آمده فقط جنبه راهنمایی شیوه استفاده از لاتک می‌باشد. خواهشمند است این پاراگراف و مطالب بعدی را از نسخه جزوه‌ای که تحویل می‌دهید، حذف کنید.

۱.۱۹ Generic Interface

یکی از کاربردهای Generic این است که بتوان نوع داده ی ساده و پیچیده را مانند عدد، رشته و ... را به عنوان یک پارامتر به متد ها، کلاس ها و اینترفیس اضافه کند. برای نوشتن اینترفیس عمومی بهتر است برای

مجموعه کلاس های عمومی یک اینترفیس تعریف کنیم.

```
internal interface IShape
{
    void Draw();
    double GetArea();
}
```

نمونه کد ۱۸۶: اینترفیس عمومی در سی‌شارپ

```
internal class Circle: IShape
{
    private Point point;
    private int v;
    public double GetArea() => this.v * this.v * Math.PI;
    public void Draw()
    {
        Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
    }
    public Circle(Point point, int v)
    {
        this.point = point;
        this.v = v;
    }
}
```

نمونه کد ۱۸۷: کلاس circle

```
internal class Triangle: IShape
{
    private Point point1;
    private Point point2;
    private Point point3;
    public double GetArea()
    {
        return (this.point1.X*(this.point2.Y - this.point3.Y)
        + this.point2.X*(this.point3.Y - this.point1.Y)
        + this.point3.X*(this.point1.Y - this.point2.Y))/2;
    }
    public void Draw()
    {
        Console.WriteLine(" Drawing Triangle at {point1} ,{point2} ,{point3} ");
    }
    public Triangle(Point point1, Point point2, Point point3)
    {
        this.point1 = point1;
        this.point2 = point2;
        this.point3 = point3;
    }
}
```

نمونه کد ۱۸۸: کلاس Triangle

```
using System;
using System.Collections.Generic;
using System.IO;
class Program
{
    static void Main(string[] args)
    {
        Triangle t = new Triangle(new Point(2,4),
            new Point(3, -6),
            new Point(7, 8));
        Circle c  = new Circle(new Point(5, 4), 4);
        DrawShapesWithStats(t);
        DrawShapesWithStats(c);
    }
    static void DrawShapesWithStats(IShape shape)
    {
        shape.Draw();
        Console.WriteLine(shape.GetArea());
    }
}
```

نمونه کد ۱۸۹: تابع main

```
Output:Drawing Triangle at (2,4),(3,-6),(7,8)
27
Drawing Circle at (5,4), with radius: 4
50.26548245743669
```

۲.۱۹ Generic Constraints

گاهی وقت ها، نیاز داریم که زمان ساخت شئ، تنها نوع داده ای که از نوع Type Value و یا فقط reference type باشد را قبول کند. برای اینکار، می توانیم از محدودیت ها(Constraint) استفاده کنیم. برای استفاده از Constraint از کلمه where استفاده می کنیم.

محدودیت ()new مشخص می کند که یک آرگمان در تعریف کلاس عمومی باید دارای یک سازنده بدون پارامتر عمومی باشد. هنگامی که یک کلاس عمومی نمونه های جدیدی از Type را ایجاد می کند ، همانطور که در مثال زیر نشان داده شده است ، محدودیت جدید را روی پارامتر Type اعمال می کنیم.

به عنوان مثال در نمونه کد بالا برای استفاده از محدودیت ها و ()new در کلاس Program، میتوان تابع DrawShapeWithStats را این گونه نوشت:

```
static void DrawShapesWithStats<T>(T shape) where T: IShape, new()
{
    T s1 = new T();
    shape.Draw();
    Console.WriteLine(shape.GetArea());
}
```

نمونه کد ۱۹۰: Generic Constraints

۳.۱۹ IDisposable

زمانی که شئ ای روی یک کلاس ایجاد می کنیم، برای متغیر تعریف شده فضایی در Stack ایجاد می شود، شئ در حافظه Heap که یک حافظه مدیریت شده است ایجاد می شود و آدرس حافظه Heap در Stack قرار می گیرد. حافظه Heap حافظه ای است که دائماً توسط سرویسی به نام Collector Garbage مدیریت می شود و اشیاء بدون استفاده آن، توسط این سرویس حذف می شوند.

شیوه پاک سازی دستی منابع استفاده شده توسط اشیاء با استفاده از پیاده سازی اینترفیس IDisposable این اینترفیس متدی با نام Dispose دارد که به شکل ()void Dispose صدا زده می شود و بعد از پیاده سازی می توان دستورات جهت پاک سازی منابع را داخل آن نوشت.

به عنوان مثال در کلاس Circle داریم:

```
internal class Circle: IShape, IDisposable
{
    private Point point;
    private int v;
    public void Dispose()
    {
        Console.WriteLine(" Clearing Circle ");
    }
    public double GetArea()
    {
        return this.v * this.v * Math.PI;
    }
    public void Draw()
    {
        Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
    }
    public Circle(){}
    public Circle(Point point, int v)
    {
        this.point = point;
        this.v = v;
    }
}
```

نمونه کد ۱۹۱: تابع main

StreamReader Class ۴.۱۹

برای خواندن فایل های متنی (txt) از کلاس StreamReader استفاده می کنیم.StreamReader اغلب با استفاده از جمله using نوشته می شود. این ساختار(using) به پاکسازی منابع سیستم کمک می کند.

```
static void Main()
{
    string line;
    using (StreamReader reader = new StreamReader("file.txt"))
    {
        line = reader.ReadLine();
    }
    Console.WriteLine(line);
}
```

نمونه کد ۱۹۲: خواندن فایل با streamreader

Stopwatch ۵.۱۹

در سی شارپ برای اندازه گیری زمان یک برنامه از کلاس Stopwatch استفاده می کنند.در نمونه کد زیر با استفاده از این کلاس زمان خواندن یک فایل را اندازه و سپس با Dispose حافظه را آزاد می کنیم.

```
using System;
using System.Diagnostics;
internal class Timer: IDisposable
{
    private Stopwatch s = new Stopwatch();
    private string Name;
    public Timer(string name)
    {
        this.Name = name;
        s.Start();
    }
    public void Dispose()
    {
        s.Stop();
        Console.WriteLine(" Elapsed Time({this.Name}): {s.Elapsed.ToString()} ");
    }
}
```

نمونه کد ۱۹۳: کلاس Timer

```
class Program
{
    static void Main(string[] args)
    {
        using (Timer t = new Timer("ReadAllLines"))
        {
            string[] lines = File.ReadAllLines("file.txt");
        }
        using (Timer t2 = new Timer("StreamReader"))
        {
            StreamReader reader;
            using (reader = new StreamReader("file.txt"))
            {
                string line;
                while (null != (line = reader.ReadLine()))
                {
                    if (line.Length > 5)
                        break;
                }
            }
        }
    }
}
```

نمونه کد ۱۹۴: اندازه گیری زمان اجرای برنامه خواندن فایل

```
Elapsed Time(ReadAllLines): 00:00:00.1567477
Elapsed Time(StreamReader): 00:00:00.0001820
```

IEnumerable ۶.۱۹

Enumerator شی ای است که قابلیت بازگرداندن هر مورد از یک لیست و گروه داده ای را به صورت یک به یک و ترتیبی که از آن درخواست میشود را دارد.Enumerator به طبقه بندی موارد آگاه است و پیگیری می کند که کجای رشته قرار دارد.و بعد از آن مقدار مورد جاری را بنا به در خواست باز می گرداند.

اینترفیس IEnumerable توسط یک کلاس IEnumerable پیاده سازی می شود.Enumerable یک نوع است که یک متد به نام GetEnumerator دارد که این متد یک enumerator برمی گرداند.

Current مقدار جایگاه جاری در رشته را برمی گرداند.

MoveNext متدی است که enumerator را به موقعیت بعدی مکانی پیش می برد. و یک مقدار بولین برمی گرداند که نشان دهنده معتبر بودن موقعیت بعدی و یا انتهای رشته می باشد.

موقعیت ابتدایی enumerator ، قبل از اولین مورد در رشته است. پس تابع MoveNext باید قبل از اولین دسترسی به Current فراخوانی شود.

Yield يك عنصرمجموعه را برمي گرداند وموقعيت مكان نما را به عنصر بعدي هدايت مي كند.

در نمونه کد زیر کاربردی از این اینترفیس را میبینم.

```
using System.Collections.Generic;
public class PowersOf2
{
    static void Main()
    {
        foreach (int i in Power(2, 8))
        {
            Console.Write(" {0} ", i);
        }
    }
    public static IEnumerable<int> Power(int number, int exponent)
    {
        int result = 1;

        for (int i = 0; i < exponent; i++)
        {
            result = result * number;
            yield return result;
        }
    }
}
```

نمونه کد ۱۹۵: کاربرد yield در اینترفیس IEnumerable

```
Output: 2 4 8 16 32 64 128 256
```

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main()
    {
        List<int> list = new List<int>();
        list.Add(1);
        list.Add(5);
        list.Add(9);
        List<int>.Enumerator e = list.GetEnumerator();
        Write(e);
    }
    static void Write(IEnumerator<int> e)
    {
        while (e.MoveNext())
        {
            int value = e.Current;
            Console.WriteLine(value);
        }
    }
}
```

نمونه کد ۱۹۶: کاربرد GetEnumerator در اینترفیس IEnumerable

```
Outut: 1 5 9
```

[**DotNetPerls**] [**csharpdocumentation**] [**sharp۲۰۰۲microsoft**]

جلسه ۲۰

چگونگی کارکرد مموری

سعید شهیب زاده - ۱۳۹۹/۲/۸

جزوه جلسه ۲۰ام مورخ ۱۳۹۹/۲/۸درس برنامه‌سازی پیشرفته تهیه شده توسط سعید شهیب زاده. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

۱.۲۰ Memorymanager Example

در این جلسه مثالی در باره چگونگی کارکرد مموری در سیستم عامل زده شد که در مورد آن صحبت می شود.

در ابتدا کلاسی به نام Memorymanager تعریف میکنیم که دارای ویژگی های زیر است:

```
public class Memorymanager
{
    private int size;
    private int[] Memory;
    private int FirstFree;
    private const Int16 NoMoreValues = Int16.MaxValue;
    public MemoryManager(int size)
    {
        this.size = size;
        this.Memory = new int[size];
        this.FirstFree = 0;
        this.Memory[0] = (int) new IntBlock((Int16)size, NoMoreValues);
    }
}
```

نمونه کد ۱۹۷: کلاس Memorymanager

در این کلاس ما بلاک هایی از مموری خواهیم داشت که در ارایه Memory ذخیره خواهند شد

سایز ارایه به وسیله متغیر size مشخص میشود متغیر FirstFree ادرس اولین بلاک خالی در مموری را در خود خواهد داشت و متغیر NoMoreValue همانطور که از اسمش مشخص است برای آن است که بدانیم آیتم دیگری نخواهیم داشت.

همچنین یک ساختار به نام IntBlock را می سازیم برای راحت تر شدن تبدیل های اینتجر که به صورت زیر است:

```
internal struct IntBlock
{
    private Int32 @value;

    public Int16 size {
        get => SplitNumbers(@value).size;
        set => CombineNumbers(value, this.next);
    }

    public override string ToString() => $,{next})""({size};

    public Int16 next
    {
        get => SplitNumbers(@value).next;
        set => CombineNumbers(this.size, value);
    }

    public static explicit operator int(IntBlock b) => b.value;

    public static implicit operator IntBlock(int b) => new IntBlock(b);

    public IntBlock(Int32 num) => this.value = num;

    public IntBlock(Int16 size, Int16 next)
    {
        @value = CombineNumbers(size, next);
    }
    private static int CombineNumbers(Int16 size, Int16 next)
    {
        int result = size;
        result = result << 16;
        result = result | (UInt16)next;
        return result;
    }

    private static (Int16 size, Int16 next) SplitNumbers(int num)
    {
        int size = num;
        size = size >> 16;
        int next = ((int)Math.Pow(2, 16) - 1) & num;
        return ((Int16)size, (Int16)next);
    }
}
```

نمونه کد ۱۹۸: کلاس IntBlock

متد CombineNumbers دو عدد از نوع اینتجر ۱۶ بیتی به عنوان ورودی میگیرد و این دو عدد را باهم ترکیب کرده و یک اینتجر ۳۲ بیتی به عنوان خروجی تحویل میدهد. این کار به خاطر جای دادن دو عدد که سایز و ادرس بلاک خالی بعدی است در یک عدد اینتجر هست که بتوانیم در یک خانه ارایه مموری دو عدد داشته باشیم.

متد SplitNumbers هم دقیقا برعکس متد CombineNumbers هست و میتواند یک عدد اینتجر ۳۲ بیتی را به دو عدد درون یک تاپل برگرداند.

علاوه بر ساختار IntBlock یک کلاس دیگر به نام MemoryBlock داریم که همانطور که از اسمش مشخص است بلاک های مموری را مشخص میکند و دارای دو عدد که ادرس شروع و پایان آن بلاک هست و دارای یک ارایه که محتوای آن بلاک را مشخص میکند.

در زیر میتوانیم این کلاس را نیز مشاهده کنیم:

```
class MemoryBlock
{
    private int[] Data;
    public int Start;
    public int End;

    public override string ToString()
    {
        return $"{this.Start}->{this.End}";
    }

    public int Size => End - Start + 1;
    public MemoryBlock(int[] data, int start, int end) =>
        (Data, Start, End) = (data, start, end);
}
```

نمونه کد ۱۹۹: کلاس MemoryBlock

حالا که کلاس های IntBlock و MemoryBlock را دیدیم میتوانیم به سراغ کلاس Memoryman-ager و متد های آن برویم

این کلاس یک متد به نام AcquireMemory دارد که در واقع با گرفتن یک سایز، یک بلاک از مموری با همان سایز را تحویل میدهد.

میتوانیم این متد را ببینیم:

```
public MemoryBlock AcquireMemory(int size)
{
    int loc = FirstAvailable(size);
    return new MemoryBlock(data:this.Memory, start:loc, end:loc+size-1);
}
```

نمونه کد ۲۰۰: متد AcquireMemory

علاوه بر تخصیص بلاک در مموری یک متد برای حذف بلاک هایی که نمیخواهیم هم نیاز داریم این متد را DeleteMemory نامگذاری میکنیم و پیاده سازی آن به صورت زیر است:

```
public void DeleteMemory(MemoryBlock mb)
{
    var current = this.FirstFree;
    while (true)
    {
        IntBlock bi = this.Memory[current];
        if (bi.next == NoMoreValues)
        {
            this.Memory[current] = (int) new IntBlock(bi.size, (Int16) mb.Start);
            this.Memory[mb.Start] = (int) new IntBlock((Int16) mb.Size, NoMoreValues);
            break;
        }
    }
}
```

نمونه کد ۲۰۱: متد DeleteMemory

این کلاس همچنین متد های CombineNumbers و SplitNumbers را داراست که نحوه پیاده سازی آنها را پیش تر دیدم.

ما همچنین به یک متد برای بررسی بلاک های خالی مموری نیاز داریم ، این متد FreeBlocks نام دارد و پیاده سازی آن به صورت زیر است.

```
public IEnumerable<IntBlock> FreeBlocks()
{
    int current = this.FirstFree;
    while (current != NoMoreValues)
    {
        IntBlock cb = this.Memory[current];
        yield return cb;
        current = cb.next;
    }
}
```

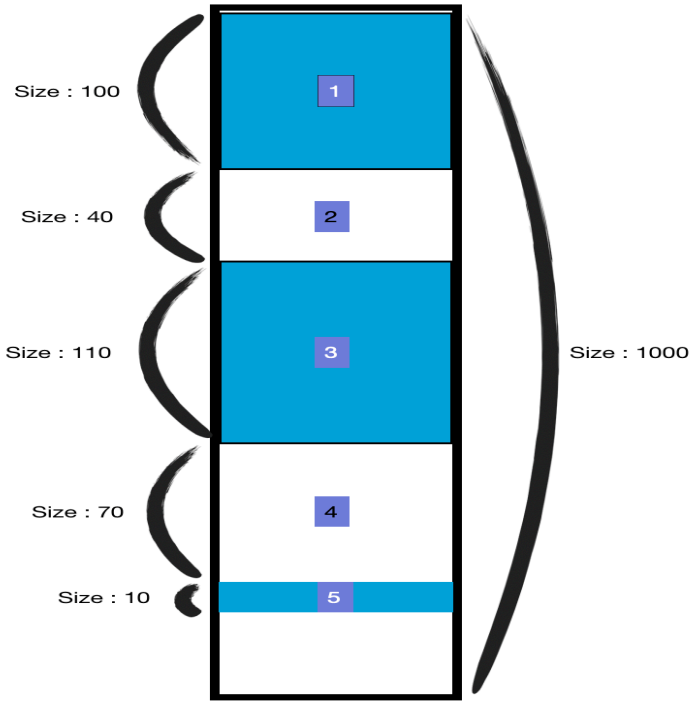
نمونه کد ۲۰۲: متد FreeBlocks

این متد از نوع IEnumerable است به این معنی که میتوانیم بر روی آن foreach بزنیم وعناصر را یکی یکی با دستور yield return برگردانیم.

دستور yield return به این صورت عمل میکند که درون حلقه قرار میگیرد و وقتی متد به آن رسید یک

عنصر از آن ارایه را برمیگرداند و از متد خارج میشود و بعد دوباره به متد برمیگردد از ادامه دستور return yield شروع میکند تا دوباره به آن برسد و عنصر بعدی را برمیگرداند

متد دیگری که کلاس MemoryManager داراست متد FirstAvailable است که به عنوان ورودی یک سایز میگیرد و در مموری اولین بلاک خالی که ظرفیت لازم یا همان سایز مورد نظر را دارد پیدا میکند و ادرس آن را برمیگرداند



برای مثال در تصویر بالا اگر یک بلاک با اندازه ۵۰ بخواهیم متد FirstAvailable ادرس بلاک ۴ام که عدد ۲۵۰ است را برمیگرداند

حالا می توانیم نحوه پیاده سازی متد را ببینیم :

```
public int FirstAvailable(int size)
{
    int current = this.FirstFree;
    int last = this.FirstFree;
    int location = NoMoreValues;
    while(current != NoMoreValues)
    {
        IntBlock cb = this.Memory[current];
        IntBlock lb = this.Memory[last];
        if (cb.size >= size)
        {
            location = current;
            if (last == current)
                this.FirstFree = current + size;
            else
                this.Memory[last] = (int) new IntBlock( lb.size,
                    cb.size > size ?
                    (Int16)(current + size) :
                    cb.next);

            if (cb.size > size)
                this.Memory[current + size] = (int) new IntBlock(
                    (Int16)(cb.size - size), cb.next
                );

            break;
        }
        last = current;
        current = cb.next;
    }

    if (current == NoMoreValues)
        throw new InvalidOperationException();

    return location;
}
```

نمونه کد ۲۰۳: متد FirstAvailable

و در آخر یک متد هم برای چاپ کردن بلاک های خالی در کنسول داریم که میتوانیم پیاده سازی آن را مشاهده کنیم:

```
public void PrintBlocks()
{
    foreach(var info in this.FreeBlocks())
        Console.WriteLine(info);
}
```

نمونه کد ۲۰۴: متد PrintBlocks

حالا که با نحوه کار مموری اشنا شدیم میتوانیم استفاده از آن را هم ببینیم برای مثال یک مموری با اندازه ۱۰۰۰ درست کردیم و از متد های AcquireMemory و DeleteMemory استفاده میکنیم و بلاک هایی از انرا اشغال میکنیم و با استفاده از PrintBlocks میبینیم که به درستی کار میکند:

```
static void Main(string[] args)
{
    MemoryManager mm = new MemoryManager(size:100);
    mm.PrintBlocks();

    var m10 = mm.AcquireMemory(10);
    mm.PrintBlocks();

    var m20 = mm.AcquireMemory(85);
    mm.PrintBlocks();

    mm.DeleteMemory(m10);
    mm.PrintBlocks();

    var m3 = mm.AcquireMemory(8);
    mm.PrintBlocks();

    IntBlock b = new IntBlock(5, 10);
    int[] memory = new int[10];
    memory[5] = (int) b;
}
```

نمونه کد ۲۰۵: تابع Main

Output:

(۱۰۰،۳۲۷۶۷)

(۹۰،۳۲۷۶۷)

(۵،۳۲۷۶۷)

(۵،۰)

(۱۰،۳۲۷۶۷)

(۵،۸)

(۲،۳۲۷۶۷) (عدد ۳۲۷۶۷ به معنی آن است که بلاکی بعد از آن بلاک وجود ندارد)

جمع بندی :

در این جلسه با نحوه کار مموری در سیستم عامل های مختلف با استفاده از یک مثال آشنا شدیم همچنین نحوه کار با IEnumerable و راحت تر شدن کار به وسیله آن را دیدیم .

با آرزوی موفقیت و سلامتی

جلسه ۲۱

اشاره گر به تابع

مهدیه نادرری - ۱۳۹۸/۲/۱۳

۱.۲۱ اشاره گر به تابع *در زبان پایتون

همان طورکه از قبل می دانیم تمامی پارامترها (آرگومان ها) در زبان پایتون با reference پاس داده می شوند، بدین معنی که اگر آنچه یک پارامتر به آن اشاره دارد را در تابع تغییر دهید، تغییر در تابع فراخواننده نیز منعکس می شود. درنتیجه ما می توانیم مانند مثال زیر یک ورودی را به عنوان تابع روی متغیر های دیگر صدا بزنیم. سپس با فراخوانی مناسب تابع اصلی نتیجه ی دلخواه را دریافت کنیم.

*function pointer

```
def add(a, b):
    return a+b

def mul(a, b):
    return a*b

def sub(a, b):
    return a-b

def combine(list_a, list_b, fn):
    list_result = []
    for i in range(0, len(list_a)):
        result = fn(list_a[i], list_b[i])
        list_result.append(result)
    return list_result

def print_pretty(list):
    for i in range(0, len(list)):
        print(f"[{i}]={list[i]}")

def main():
    a = [1, 2, 3, 4, 5]
    b = [2, 3, 4, 5, 7]
    c = combine(a, b, add)
    print_pretty(c)

if __name__ == "__main__":
    main()
```

نمونه کد ۲۰۶: ارسال یک تابع به عنوان ورودی تابع دیگر در پایتون

به این کد تابع دیگری مثل ()negative به کد اضافه می کنیم و در main به جای add در تابع ()combine از آن استفاده می کنیم . در پایتون ()negative را با هر تعداد ورودی صدا بزنیم، پردازشگر ایرادی نمیگیرد ولی موقع اجرای برنامه با خطا ی کامپایل روبرو می شویم. بررسی نکردن نوع متغیر ها از ویژگی های این زبان است و در زمان run time اگر درست نبود خطا میدهد و در غیر این صورت اجرا میکند.

```
def negative(a):
    return -1 * a
```

۲.۲۱ اشاره‌گر به تابع در زبان سی شارپ

در سی شارپ برای رفع مشکل مثال قبل باید از قبل برای تابعی که قرار است به عنوان ورودی، آرگومان تابعی دیگر باشد؛ تعداد متغیر های ورودی ونوع آنها و نوع خروجی مشخص شود. استفاده از کلید واژه ی

delegate ضمانت میکند آنچه در ادامه می آید ، یک نوع تابع است نه خود تابع و کسی نمیتواند تابعی از نوع دیگر به تابع ثانی بدهد.

```
using System;
using System.Diagnostics;
using System.IO;

namespace c14cs
{
    class Program
    {
        delegate int binary_op_int(int a,int b);
        static int mul(int a, int b) => a*b;
        static int add(int a, int b) => a+b;
        static int[] combine(int[] a, int[] b, binary_op_int fn)
        {
            int[] result = new int[a.Length];
            for (int i = 0; i < result.Length; i++)
            {
                result[i] = fn(a[i], b[i]);
            }
            return result;
        }
        static void Main(string[] args)
        {
            int[] list_a = new int[] {3, 2, 3, 1, 5};
            int[] list_b = new int[] {1, -2, 2, 4, 5};
            var c = combine(list_a, list_b, add);
        }
    }
}
```

```
using System;
using System.Diagnostics.CodeAnalysis;

namespace l21cs
{
    public class Student
    {
        public int id;
        public double GPA;

        public Student(int id, double GPA)
        {
            this.id = id;
            this.GPA = GPA;
        }
    }
}
```

در کد بالا پیاده‌سازی همان مثال پایتون را در سی شارپ مشاهده می کنید. برای مرتب کردن آرایه ها

تابع جدیدی به نام Sort() می سازیم.

```
static void Sort(int[] list)
{
    for (int i = 0; i < list.Length; i++)
    {
        for (int j = i+1; j < list.Length; j++)
        {
            if(list[i] < list[j])
                (list[i], list[j]) = (list[j], list[i]);
        }
    }
}
```

(list[i], list[j]) = (list[j], list[i]) یک روش برای Swap() هست که با جابجایی دو قسمت تاپل انجام میشود.

حالا اگر بخواهیم تابع Combine() را برای نوع جدیدی مانند Student بنویسیم، اول لازم است کلاس آن را در فایل جدید Student.cs بسازیم. حال برای اینکه تابع Sort() برای آن بنویسیم باید :

```
static void Sort(Student[] list, IStudentComparer cmp)
{
    for (int i = 0; i < list.Length; i++)
    {
        for (int j = i+1; j < list.Length; j++)
        {
            if(cmp.IsGreater(list[i], list[j]))
                (list[i], list[j]) = (list[j], list[i]);
        }
    }
}
```

اینترفیس IStudentComparer پیاده سازی بشود که میخواهیم برای مقایسه ی دو Student تابعی داشته باشد. و از آن در کلاس ها استفاده کنیم.برای پیاده سازی آن میتونیم از کلاس جدیدی در Main استفاده کنیم. و کلاس StudentComparer که اینترفیس مورد نظررا دارد، بسازیم. به طور دلخواه ویژگی مورد مقایسه در IsGreater() را GPA درنظر می گیریم. البته میتوانستیم در همان کلاس Student هم این کار را انجام دهیم.در این صورت می توان به جای شرط if در تابع Sort() بنویسیم: List[i].CompareTo(list[j]).

```
namespace l21cs
{
    internal interface IStudentComparer
    {
        bool IsGreater(Student s1, Student s2);
    }
}
```

```
namespace l21cs
{
    public class StudentComparer : IStudentComparer<Student>
    {
        public bool IsGreater(Student s1, Student s2)
        {
            return s1.GPA > s2.GPA;
        }
    }
}
```

اگر بخواهیم Sort() برای هر نوع داده ای، اجرا شود باید تغییراتی در کد های قبل اعمال کنیم.(نمونه کد ۲۳۰).

```
static void sort<Type>(Type[] list, IStudentComparer<Type> cmp)
```

```
internal interface IStudentComparer<Type>
    {
        bool IsGreater(Type s1, Type s2);
    }
```

کاری که تا کنون انجام دادیم تعریف یک interface اینترفیس بود که یک متود دارد. این متود دو پارامتر از نوع Student میگیرد و یک bool برمیگرداند. خب این هم یک نوع delegate است . در نتیجه بهتر است یک delegate جدید تعریف کنیم و تابع Sort() را بر اساس آن بازنویسی کنیم.

```
delegate bool student_comparer(Student s1, Student s2));
```

```
static void sort<_Type>(Student[] list,student_comparer stdcmp)
        {
            for(int i=0; i<list.Length; i++)
                for(int j=i+1; j<list.Length; j++)
                    if (stdcmp(list[i], list[j]))
                        (list[i], list[j]) = (list[j], list[i]);
        }
```

دو تابع جدید برای مقایسه ی اعضای مجموعه نوشته ایم، یکی بر اساس GPA و دیگری بر اساس Id.

```
static bool StdCmpGPA(Student s1, Student s2) => s1.GPA > s2.GPA;
static bool StdCmpId(Student s1, Student s2) => s1.id > s2.id;
```

وقتی دو تابع همنام باشند، یکی از آنها overload دیگری است. در یکی از overload های Sort() میتوان از شیئی از کلاس StudentComparer استفاده کرد و در دیگری از تابع IsGreater() آن و البته از هر تابع مشابه دیگر. در مقایسه ی اینکه نوشتن این برنامه با اینترفیس بهتر است یا دلگیت می توان اشاره کرد که اگه اینترفیسی لازم داشتیم که چند متود داشت ، استفاده از دلگیت خیلي جالب نبود. برای دیدن نتیجه مرتب کردن لیست ، لازم است تابعی برای چاپ کردن بنویسیم.

```
private static void print<_Type>(_Type[] stdlist)
        {
            Console.WriteLine("-------------------------");
            foreach(var s in stdlist)
                Console.WriteLine(s);
        }
});
```

برای بهتر نوشتن و مدیریت آن در کلاس Student تابع Tostring() را Override() میکنیم.

```
public override string ToString() => ${id}" - "{GPA};
```

توابع را صدا میزنیم و برنامه را اجرا میکنیم.

```
static void Main(string[] args)
        {
            Student[] stdlist = new Student[]{
                new Student(98521234, 8.12),
                new Student(97532412, 8.14),
                new Student(98234324, 8.13),
                new Student(97989899, 8.11),
            };

            sort(stdlist, new StudentComparer());
            sort(stdlist, new StudentComparer());
^^I print(stdlist);
            sort(stdlist, new StudentComparer().IsGreater);
            print(stdlist);
            sort(stdlist, StdCmpGPA);
            print(stdlist);
            sort(stdlist, StdCmpId);
            print(stdlist);
^^I}
```

بعد از چاپ کردن معلوم نمی شود همه ی توابع درست کار میکنند یا نه. چون یکبار لیست موردنظر مرتب شده و هربار همون طور باقی میماند. باید تابعی بنویسیم که لیست را بی نظم کند. میتوانیم از همان تابع Sort() استفاده کنیم که تابع ورودی آن اعضا لیست را بصورت رندوم مقایسه می کند. یعنی عددی که برمیگرداند تصادفی باشد. و با ویژگی های GPA و Id ارتباطی نداشته باشد.

```
static bool RndCmp<Type>(Type t1, Type t2) =>  new Random().NextDouble() < 5.0;
```

وقتی از NextDouble استفاده می کنیم عددی بین ۰ تا ۱ برگردانده میشود و حال احتمال کمتر از ۵.۰ بودن این عدد ، ۵۰ درصد است و کاملا تصادفی است.

```
    Student[] stdlist = new Student[]{
                new Student(98521234, 8.12),
                new Student(97532412, 8.14),
                new Student(98234324, 8.13),
                new Student(97989899, 8.11),
            };

            sort(stdlist, new StudentComparer());
            print(stdlist);
            sort(stdlist, RndCmp);

            sort(stdlist, new StudentComparer().IsGreater);
            print(stdlist);
            sort(stdlist, RndCmp);

            sort(stdlist, StdCmpGPA);
            print(stdlist);
            sort(stdlist, RndCmp);

            sort(stdlist, StdCmpId);
            print(stdlist);
```

همانطور که مي بينيد توابع صدا زده شده در Main() هم از الگوي خاصي ميکنند. يك ليست مشخص را مرتب ميکنند، آن را چاپ کرده و بعد دوباره بهم ميريزند. البته با يک تغيير کوچک ميتوان گفت طرح اصلی آن اين است که اول بهم بريزد بعد سورت و سپس چاپ کند. درنتيجه ميتوانيم تابعی بنويسيم که اين سه کار را پشت سر هم انجام دهد. پس تابع RunSortExpriment() را طوری مينويسيم که يک String برای عنوان کاری که انجام ميدهد ، يك ليست و آبجکتي از کلاس StudentComparer و دو تابع برای Sort() و print() به عنوان ورودی ميگيرد. و delegate های زير

```
delegate void sort_delegate<_Type>(_Type[] list, student_comparer stdcmp);
delegate void print_delegate<_Type>(_Type[] stdlist);
```

را برای آن تعريف ميکنيم.

```
static void RunSortExperiment<_Type>(
    string label,
    _Type[] list,
    sort_delegate<_Type> sortfn,
    print_delegate<_Type[]> printfn,
    StudentComparer cmp)
        {
            sortfn(list, RndCmp);
            sortfn(list, cmp);
            Console.WriteLine(label);
            printfn(list);
        }
```

برای راحت کردن کارها می توانیم از Acnomiss delegate استفاده کنیم یعنی نوع فانکشن را بدون تعریف متود جداگانه تعریف کنیم و نوع ورودی و خروجی و تعدادشان را در همانجا که قرار است استفاده شوند ، مشخص کنیم. در حقیقت هر آنچه براي تعريف يك تابع لازم است را بايد نوشت، البته مي توان آن را خلاصه كرد و اگر متغير هايي را از قبل تعريف كرديم ميتوانيم ديگر نوع آنهارا در زمان استفاده در تابع ننويسيم. لازم نيست نوع متغير هاي اطراف را وارد كنيد و هر وقت بخواهيد قابل دسترس هستند. مثلا زمان صدا زدن تابع Sort() بنويسيم:

```
sort(stdlist, (s1, s2) => { return s1.GPA > s2.GPA;});
```

یا بطور خلاصه تر

```
 sort(stdlist, (s1, s2) => s1.GPA > s2.GPA );
```

که این روش همان استفاده از lambda expression ها است. مثلا در متود Find() در لیست ها predicate که در واقع يك delegate است دارد: اين تابع به عنوان ورودي يك Student ميگيرد و يک bool برميگرداند. به ترتيب اعضاي ليست را با عدد داداه شده مقايسه ميكند و اگر برابر شد true برميگرداند.

```
List<Student> std_list = new List<Student>(stdlist);
var s97532412 = std_list.Find( (Student s) => {
            return s.id == 97532412;
});
```

در فریم ورک دات نت delegate هایی از پیش تعیین شده وجود دارد که میتوانند هر تعداد پارامتر با انواع مختلف داشته باشند از جمله Func و Action. مثلا در تابع Sort() به جای استفاده از StudentComparer ، می توانیم Func<Student,Student, bool> stdcmp را بکار ببریم. این نوع حداقل یک متغیر میگیرد که نشان دهنده ی نوع خروجی آن است و در ادامه انواع ورودی هایش نوشته میشود. در عوض Action نوع خروجی ندارد و اگر تابع مورد نظر مثل print() باشد میتوانیم از آن استفاده کنیم.

اگر به delegate ها ی تعریف شده نگاهي بیندازیم، از نظر تعداد ورودي و خروجي شباهت هايي دارند که سبب شده بتوانیم به صورت Generic تعریفشان کنیم. و برخی از آن هارا حذف کنیم.

```
delegate int binary_op_int(int a, int b);
delegate void sort_delegate<_Type>(_Type[] list, Func<_Type, _Type, bool> stdcmp);
delegate void print_delegate<_Type>(_Type[] stdlist);
```

حال میتوانیم تابع RunSortExpriment() را بازنویسی کنیم.

```
static void RunSortExperiment<_Type>(
            string label,
            _Type[] list,
            sort_delegate<_Type> sortfn,
            Action<_Type[]> printfn,
            Func<_Type, _Type, bool> cmp)
        {
            sortfn(list, RndCmp);
            sortfn(list, cmp);
            Console.WriteLine(label);
            printfn(list);
        }
```

و در Main() تغییرات لازم را ایجاد نماییم.

```
RunSortExperiment("InterfaceMethod",
            stdlist,
            sort,
            print,
            new StudentComparer().IsGreater);
RunSortExperiment("StdCmpGPA", stdlist, sort, print, StdCmpGPA);
RunSortExperiment("StdCmpId", stdlist, sort, print, StdCmpId);
```

جلسه ۲۳

Closure - Async Pattern

پارمیدا مجمع صنایع - ۱۳۹۹/۰۲/۲۰

مطالب مطرح شده در این جلسه به شرح زیر است:

- مفهوم Closure در دو زبان سی‌شارپ و سی پلاس پلاس
- Multi-Threading

۱.۲۳ Closure در c++

ابتدا وکتوری از اعداد صحیح را تشکیل می‌دهیم. سپس از فانکشن for_each موجود در کتابخانه algorithm استفاده می‌کنیم. (نمونه کد ۲۰۷)

```cpp
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int main(int argc, char const *argv[])
{
    vector<int> vNums {1, 2, 5, 3, 4, 1};

    for_each(vNums.begin(), vNums.end(), [] (int n) {
        cout << n << endl;

    return 0;
}
```

نمونه کد ۲۰۷: تعریف وکتور و استفاده از for_each C++

همان‌طور که می‌بینید، به عنوان ورودی به فانکشن for_each امان ابتدای وکتور و سپس انتهای وکتور را می‌دهیم و در آخر یک فانکشن یا یک expression lambda به آن می‌دهیم. که این فانکشن روی تک تک اعضای وکتورمان اجرا می‌شود.

*برای مشاهده مطالب بیشتر درمورد lambda expression در c++ می‌توانید به این سایت رجوع کنید.
[**lambdaex**]

همچنین روشی دیگر برای پیمایش در طول وکتور و اجرای فانکشنی بر روی تک به تک اعضا علاوه بر روش بالا وجود دارد.(نمونه کد ۲۰۸)

```cpp
int main(int argc, char const *argv[])
{
    vector<int> vNums {1, 2, 5, 3, 4, 1};
    vector<int>::iterator it;

    for(it = vNums.begin(); it != vNums.end(); it++)
    {
        cout << *it << endl;
    }

    return 0;
}
```

نمونه کد ۲۰۸: تعریف iterator برای وکتور for_each C++

توضیح: در این روش، ابتدا یک iterator برای پیمایش در طول وکتورمان تعریف می‌کنیم. سپس با استفاده از آن یک حلقه ی فور نوشته و مقدار اولیه ی آن رو برابر آغاز وکتورمان قرار می‌دهیم و حداکثر مقداری که می‌پذیرد را برابر انتهای وکتورمان قرار می‌دهیم و هر دفعه iterator یک عدد زیاد می‌شود. سپس در حلقه ی فورمان، هر دفعه مقدار فعلی it که یک iterator می‌باشد، چاپ می‌شود.

نکته: برای بدست آوردن مقدار فعلی iterator از علامت * استفاده می‌کنیم. برای مفهوم بهتر، عملکرد *it همانند Current() در IEnumerable در زبان سی‌شارپ است. و عملکرد i++ همانند MoveNext() در IEnumerable در زبان سی‌شارپ است.

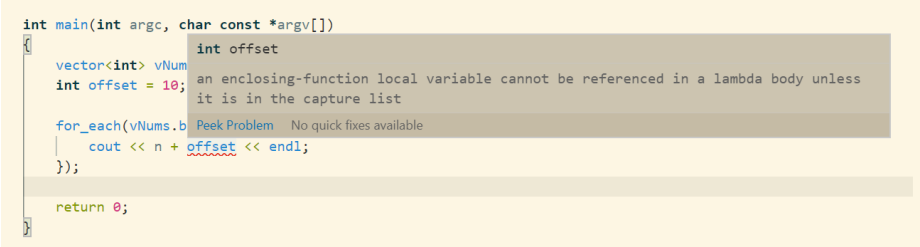
*حال برای مطرح کردن نکته ی جدیدی به (نمونه کد ۲۰۷) باز می‌گردیم. با نگاه کردن به سومین ورودی for_each در می‌یابیم که یک expression lambda است. یک ورودی از نوع int می‌گیرد و آن را چاپ می‌کند و به خط بعدی می‌رود. در واقع با اجرای کد، اعداد داخل وکتور هر کدام در خطی جداگانه چاپ می‌شوند.

حال متغیر جدیدی از نوع int به نام offset تعریف می‌کنیم و مقدار آن را برابر ۱۰ قرار می‌دهیم. این بار می‌خواهیم از offset در داخل expression lambda خود استفاده کنیم. کد خود را به شکل زیر تغییر می‌دهیم.(نمونه کد ۲۰۹)

```cpp
int main(int argc, char const *argv[])
{
    vector<int> vNums {1, 2, 5, 3, 4, 1};
    int offset = 10;

    for_each(vNums.begin(), vNums.end(), [] (int n) {
        cout << n + offset << endl;
    });

    return 0;
}
```

نمونه کد ۲۰۹: استفاده از متغیر محلی در lambda expression

در این مرحله ما با ارور زیر مواجه می‌شویم:



شکل ۱.۲۳: خطا در استفاده از متغیر محلی در فانکشن

علت مواجه شدن با این خطا، استفاده از متغیر محلی در فانکشنمان است. چون درواقع ورودی سوم for_each تعریفی از یک فانکشن جدید است و واضح است که offset متغیری ناشناخته در این فانکشن می‌باشد.

برای حل این مشکل از Clause Capture استفاده می‌کنیم. به این شکل که داخل براکت نام متغیری که می‌خواهیم از آن استفاده کنیم را می‌نویسیم. اینگونه فانکشن ما متغیر را شناخته و بدون هیچ خطایی کد اجرا می‌شود.(نمونه کد ۲۱۰)

```cpp
int main(int argc, char const *argv[])
{
    vector<int> vNums {1, 2, 5, 3, 4, 1};
    int offset = 10;

    for_each(vNums.begin(), vNums.end(), [offset] (int n) {
        cout << n + offset << endl;
    });

    return 0;
}
```

نمونه کد ۲۱۰: استفاده از Clause Capture در C++

نکته: برای اینکه مشخص کنیم که متغیرهای داخل Capture Clause به صورت pass by value یا pass by reference در نظر گرفته شوند، به این صورت عمل می‌کنیم:

- اگر می‌خواهیم متغیرمان pass by value شود، کافیست جلوی نام آن علامت `=` در Capture Clause گذاشته شود. و اگر می خواهیم تمام متغیرهای محلی‌مان pass by value شوند، کافیست داخل براکت فقط یکبار علامت `=` را بگذاریم و در این صورت نیازی به نوشتن نام تک تک متغیر ها نیست.
- اگر می‌خواهیم متغیرمان pass by reference شود، کافیست جلوی نام آن علامت `&` در Capture Clause گذاشته شود. و اگر می خواهیم تمام متغیرهای محلی‌مان pass by reference شوند، کافیست داخل براکت فقط یکبار علامت `&` را بگذاریم و در این صورت نیازی به نوشتن نام تک تک متغیر ها نیست.

برای مشاهده ی آنچه گفته شد، نمونه کد زیر را ببینید.

```
int main(int argc, char const *argv[])
{
    vector<int> vNums {1, 2, 5, 3, 4, 1};
    int offset = 10;
    int sum = 0;

    offset : pass by value, sum : pass by reference

    for_each(vNums.begin(), vNums.end(), [offset, &sum] (int n) {
        cout << n + offset << endl;
        sum += n;
    });

    offset and sum : pass by value
    for_each(vNums.begin(), vNums.end(), [=] (int n) {
        cout << n + offset << endl;
        sum += n;
    });

    offset : pass by reference, sum : pass by value
    for_each(vNums.begin(), vNums.end(), [&offset, sum] (int n) {
        cout << n + offset << endl;
        sum += n;
    });

    offset and sum : pass by referene
    for_each(vNums.begin(), vNums.end(), [&] (int n) {
        cout << n + offset << endl;
        sum += n;
    });

    return 0;
}
```

شکل ۲.۲۳: انواع استفاده از Clause Capture

این مفهوم که می‌توانیم از متغیرهای محلی در فانکشنمان استفاده کنیم، از مفهومی به نام closure گرفته شده است. درواقع در این مفهوم یک آبجکت ساخته می‌شود. که این آبجکت یک متود دارد که همان فانکشن ما است و تمام متغیرهای محلی * را داخل آن قرار می‌دهد. اینگونه است که می‌توانیم از متغیرهای محلی در فانکشن خود استفاده کنیم.

*برای درک بهتر آنچه گفته شد، می‌توانید به این سایت ها رجوع کنید. [**cppclosure**] [**moreabout**]

Local Variables*

۲.۲۳ Closure در #c

برای آشنا شدن با این مفهوم، ابتدا لیستی از اکشن را تعریف می‌کنیم؛ سپس یک حلقه فور می‌نویسیم و داخل آن lambda expression های ایجاد شده را به لیست اکشن اد می‌کنیم.(نمونه کد ۲۱۱)

```
1  public class Program
2  {
3        static void Main(string[] args)
4        {
5              List<Action> funcs = new List<Action>();
6              for (int i = 0; i < 10; i++)
7              {
8                  funcs.Add(() => Console.WriteLine(i);
9              }
10
11             foreach(var fn in funcs)
12                fn();
13
14       }
15
16 }
```

نمونه کد ۲۱۱: ایجاد لیستی از اکشن

توضیح: علت اینکه در lambda expression خود می‌توانیم از متغیر i استفاده کنیم، مفهوم closure است که در زبان سی‌شارپ، پنهان و نهفته است. درواقع چون i یک متغیر محلی نزدیک به فانکشن ما می‌باشد، استفاده از آن برای ما ممکن شده است.

نکته: نکته ی حائز اهمیت در این قسمت این است که در حلقه ی فور ما کدی اجرا نمی‌شود؛ بلکه در این حلقه صرفا اکشن هایی تعریف و به لیست اضافه می‌گردند. قسمتی که این اکشن ها شروع به اجرا شدن می‌کنند در حلقه ی foreach ما است.

نکته ی جالب: با اجرا شدن Main برنامه، فکر می‌کنیم که اعداد ۰ تا ۹ هرکدام در خطی جداگانه در خروجی چاپ می‌شوند. در حالی که کاملا اشتباه کرده ایم و خروجی ۱۰ بار عدد ۱۰ را چاپ می‌کند.

علت: علت این اتفاق، همان نکته ایست که در بالا گفته شد. زیرا اسکوپ مربوط به i مربوط به کل حلقه ی فور است. پس ما درواقع فقط یک i داریم. که این i pass by reference می‌شود. درواقع با اجرای Main ابتدا حلقه ی فور اجرا شده، و i که مقدار اولیه صفر را دارد، یکی یکی زیاد می‌شود تا به عدد ۱۰ می‌رسد؛ و زمانی که foreach اجرا می‌شود و اکشن ها شروع به اجرا می‌کنند ما فقط یک متغیر i داریم که مقدار آن ۱۰ می‌باشد. بنابراین عدد ۱۰، ۱۰ بار چاپ می‌شود.

توضیح: در بیان کلی تر باید بگوییم که علت این اتفاق همان مفهوم closure می‌باشد. زیرا گفتیم که طبق این مفهوم آبجکتی ساخته شده و تمام متغیرهای محلی را داخل آن قرار می‌دهد. بنابراین به ازای هر متغیر فقط یک مقدار از آن متغیر داریم؛ زیرا فقط یکبار i را داخل آبجکت قرار می‌دهد.

حال می‌خواهیم برای جا افتادن مفهوم تغییری را در کد ایجاد کنیم: این بار، در حلقه ی فور تغییری ایجاد می‌کنیم که بعد از چاپ i یکی به مقدارش اضافه کند.(نمونه کد ۲۱۲)

```
1  public class Program
2  {
3        static void Main(string[] args)
4        {
5              List<Action> funcs = new List<Action>();
6              for (int i = 0; i < 10; i++)
7              {
8                  Action fn = () =>
9                  {
10                     () => Console.WriteLine(i);
11                     i++;
12                 };
13
14                 funcs.Add(fn);
15             }
16
17             foreach(var fn in funcs)
18                fn();
19       }
20 }
```

نمونه کد ۲۱۲: ایجاد لیستی از اکشن

توضیح: در این قسمت، مقدار اولیه i که به fn داده می‌شود، همان ۱۰ می‌باشد؛ بعد از اجرای اولین اکشن و چاپ عدد ۱۰ در خروجی مقدار i یکی زیاد می‌شود. و به اکشن بعدی درون لیست funcs، i با مقدار ۱۱ داده می‌شود. سپس این عدد نیز چاپ می‌شود و دوباره مقدار i یکی زیاد می‌شود و به اکشن بعدی داخل لیست، i با مقدار ۱۲ داده می‌شود و به همین ترتیب تا عدد ۱۹ در خروجی چاپ می‌شود.

۳.۲۳ Multi-Threading

اگر ما بخواهیم چند کار را همزمان با هم انجام دهیم، باید از این مفهوم استفاده کنیم.در اینجا برای مثال تابعی نوشتیم که یک عدد به عنوان ورودی دریافت کند و از صفر تا آن عدد را چاپ کند و بعد از چاپ هر عدد ۲۰۰ میلی ثانیه صبر کند.(نمونه کد ۲۱۳)

```
1  class Program
2  {
3      static void CountToN(int n)
4      {
5          for (int i = 0; i < n; i++)
6          {
7              Console.WriteLine(i);
8              Thread.Sleep(200);
9          }
10     }
11     static void Main(string[] args)
12     {
13         CountToN(6);
14         Console.WriteLine("Doing-other-things");
15     }
16 }
```

نمونه کد ۲۱۳: انجام فقط یک کار

توضیح: همان طورکه می‌بینید، وقتی Main برنامه اجرا می‌شود، حلقه ی فور شروع به اجرا شدن می‌کند و ابتدا عدد ۰ چاپ می‌شود، سپس ۲۰۰ میلی ثانیه صبر می‌کند و دوباره عدد ۱ را چاپ می کند و دوباره ۲۰۰ میلی ثانیه صبر می کند و به همین ترتیب تا عدد ۵ در خروجی چاپ می‌شود. در این لحظه است که خط ۱۴ برنامه اجرا می شود. یعنی درواقع ابتدا باید تابع CountToN کامل اجرا بشود و سپس کارش که تمام شد بقیه ی برنامه انجام می‌شود. و در این حین ما نمی‌توانیم کاری انجام بدهیم.

خروجی کد بالا طبق چیزی که گفته شد، به صورت زیر می‌باشد:

```
0
1
2
3
4
5
Doing-other-things
```

۱.۳.۲۳ DownloadAsyncSimple

توجه: در این قسمت می‌خواهیم یاد بگیریم چگونه چند کار را بتوانیم همزمان با هم انجام بدهیم. پس یک مثال می‌زنیم که هم در واقعیت کاربرد دارد و هم انجام آن طول می‌کشد. مثالی که می‌زنیم، در مورد دانلود کردن محتوای سایت ها به صورت استرینگ است و می‌خواهیم زمان انجام آن را اندازه گرفته و همزمان کارهای دیگری را انجام دهیم.(نمونه کد ۲۱۴)

```
class Program
{
    private static void DownloadAsyncSimple()
    {
        using (HttpClient client = new HttpClient())
        {
            var asyncResultIust = client.GetStringAsync("http://www.iust.ac.ir/");
            var asyncResultGithub = client.GetStringAsync("https://github.com/");
            var asyncResultGoogle = client.GetStringAsync("https://www.google.com");
            int i = 0;
            while (asyncResultIust.IsCompleted != true ||
                  asyncResultGithub.IsCompleted != true ||
                  asyncResultGoogle.IsCompleted != true)
            {
                Console.WriteLine($"{i++}" "Waiting...");
                Console.Write($","google:{asyncResultIust.IsCompleted}");
                Console.Write($","stack:{asyncResultGithub.IsCompleted}");
                Console.WriteLine($"varzesh:{asyncResultGoogle.IsCompleted}");
                Console.WriteLine(Work" "Doing);
                Thread.Sleep(50);
            }
            Console.WriteLine(asyncResultIust.Result.Length);
        }
    }
    static void Main(string[] args)
    {
        DownloadAsyncSimple();
    }
}
```

نمونه کد ۲۱۴: دانلود محتوای سایت ها

توضیح مرحله به مرحله:

مرحله ۱: ابتدا، آبجکتی از کلاس HttpClient به نام client می‌سازیم.*خط ۵*

مرحله ۲: client متدی به نام GetStringAsync دارد که یک ورودی از نوع استرینگ می‌گیرد که آدرس یک سایت است و محتوای دانلود کرده را به صورت یک استرینگ به عنوان خروجی می‌دهد. در واقع ما سه متغیر asyncResultGoogle و asyncResultStackOverflow و asyncResultVarzesh را تعریف

کردیم تا محتوای دانلود شده را در آن ها بریزیم. *خط ۷ و ۸ و ۹*

- نکته ای که در متد GetStringAsync وجود دارد، این است که خط ۷ و ۸ و ۹ لزوما به این معنی نمی‌باشد که همان موقع نتیجه آماده شده است و در متغیر ریخته شده است، بلکه چون متد به شیوه ی async است، همان جا کار رها می‌شود و تا نتیجه آماده شود، می‌توانیم به صورت همزمان کارهای دیگری انجام دهیم.

مرحله ۳: خروجی GetStringAsync یک تسک از استرینگ می‌باشد که یک ویژگی بولین به نام IsCompleted دارد که اگر true باشد به معنی این است که تسک به طور کامل انجام شده و نتیجه به صورت استرینگ آماده است و اگر false باشد برعکس. در این قسمت شرط while تا زمانی است که حداقل حتی یکیشان هم آماده نشده باشد. و همان طور که می‌بینید در حین اینکه محتواهای ۳ سایت در حال دانلود شدن می‌باشد، ما در حال انجام کارهای دیگری هستیم.(DoingWork) و در انتها نیز طول یکی از محتواهای دانلود شده را چاپ می‌کنیم. *خط ۱۱ تا ۲۲*

قسمتی از ابتدا و انتهای خروجی این نمونه کد در صفحه ی بعد آمده است. (به علت سرعت اینترنت ممکن است خروجی شما متفاوت باشد.)

```
Waiting... 0
iust:False,github:False,google:False
DoingWork
Waiting... 1
iust:False,github:False,google:False
DoingWork
Waiting... 2
iust:False,github:False,google:False
DoingWork
Waiting... 3
iust:False,github:False,google:False
DoingWork
Waiting... 4
iust:False,github:False,google:False
DoingWork
Waiting... 5
iust:False,github:False,google:False
DoingWork
Waiting... 6
iust:False,github:False,google:False
DoingWork
Waiting... 29
iust:True,github:False,google:True
DoingWork
Waiting... 30
iust:True,github:False,google:True
DoingWork
Waiting... 31
iust:True,github:False,google:True
DoingWork
Waiting... 32
iust:True,github:False,google:True
DoingWork
120203
```

۲.۳.۲۳ DownloadTaskDemo

*در این قسمت می‌خواهیم با متد دیگری در رابطه با تسک ها آشنا شویم.(نمونه کد ۲۱۵)

```
class Program
{
    private static void DownloadTaskDemo()
    {
        using (HttpClient client = new HttpClient())
        {
            var asyncResultGoogle = client.GetStringAsync("http://www.google.com/");
            var asyncResultGithub = client.GetStringAsync("http://github.com");
            var asyncResultVarzesh = client.GetStringAsync("http://varzesh3.com");

            asyncResultGoogle.ContinueWith(str => Console.WriteLine($"Goog-Done-{str.Result.Length}"));
            asyncResultGithub.ContinueWith(str => Console.WriteLine($"Github-Done-{str.Result.Length}"));
            asyncResultVarzesh.ContinueWith(str => Console.WriteLine($"Varzeh-Done-{str.Result.Length}"));

            Task.WaitAll(asyncResultGoogle, asyncResultGithub, asyncResultVarzesh);
        }
        Console.WriteLine("Done.");
    }
    static void Main(string[] args)
    {
        DownloadTaskDemo2();
    }
}
```

نمونه کد ۲۱۵: ContinueWith

توضیح: ContinueWith یک اکشن که ورودیش یک Task<string> می‌باشد را به عنوان ورودی خود می‌گیرد و طرز کار آن این است که هر موقع تسک اولیه تمام شد، در ادامه ی آن تسک دیگری را انجام دهد. و در انتها با نوشتن Task.WaitAll و دادن تسک های اولیه به عنوان ورودی برنامه منتظر می‌ماند تاهمه ی تسک ها انجام شوند و سپس بقیه ی کد را اجرا می‌کند.

خروجی نمونه کد بالا به صورت زیر می‌باشد.

```
Goog-Done-47062
Varzeh-Done-50592
Github-Done-131868
Done.
```

۳.۳.۲۳ GetUrlContentLength

*در بخش بعدی می‌خواهیم یک تابع بنویسیم که آدرس سایت را به صورت استرینگ از ورودی بگیرد و خروجی آن تسکی از نوع int است.(نمونه کد ۲۱۶)

```
class program
{
    static Task<int> GetUrlContentLength(string url)
    {
        HttpClient client = new HttpClient();
        var asyncResultGoogle = client.GetStringAsync(url);
        Task<int> result = asyncResultGoogle.ContinueWith((ts) => ts.Result.Length);
        return result;
    }
    static void Main(string[] args)
    {
        var tasks = new Task<int>[]{
                GetUrlContentLength("http://stackoverflow.com"),
                GetUrlContentLength("http://www.google.com"),
                GetUrlContentLength("http://varzesh3.com"),
            };

        Task.WaitAll(tasks);

        foreach (var t in tasks)
        {
            Console.WriteLine(t.Result);
        }
    }
}
```

نمونه کد ۲۱۶: ContinueWith

توضیح: ابتدا، مانند قسمت های قبلی یک آبجکت از کلاس HttpClient به نام client تعریف کردیم.*خط ۵* سپس خروجی متد GetStringAsync را که یک تسک از نوع استرینگ می‌باشد را در asyncResultGoogle ریختیم. *خط ۶* سپس تسک جدیدی را با نوع خروجی int به نام result تعریف کردیم و گفتیم که هر موقع تسک asyncResultGoogle تمام شد، در ادامه طول محتوای دانلود شده را به ما برگرداند.*خط ۷* در Main برنامه نیز آرایه ای از تسک هایی تعریف کردیم که خروجیشان از نوع int می‌باشد و با دستور Task.WaitAll(tasks) گفتیم که صبر کند تا تمامی تسک ها تمام بشود و سپس با دستور foreach در طول آرایه پیمایش می‌کنیم و خروجی هر تسک را چاپ می‌کنیم که خروجی یک عدد صحیح است که برابر با طول استرینگ دانلود شده می‌باشد. خروجی نمونه کد بالا بصورت زیر می‌باشد:

```
114553
47039
50557
```

۴.۳.۲۳ GetUrlContentLengthAsync

*در این قسمت می‌خواهیم مفهوم جدیدی را بررسی کنیم. وقتی متدی async باشد، می‌توانیم در آن از کلیدواژه ی await استفاده کنیم. (نمونه کد ۲۱۷)

```
class program
{
    static async Task<int> GetUrlContentLengthAsync(string url)
    {

        HttpClient client = new HttpClient();
        Console.WriteLine(${url}" "Before-Await:);
        string result = await client.GetStringAsync(url);
        Console.WriteLine(${url}" "After-Await:);
        return result.Length;

    }
    static void Main(string[] args)
    {
        var tasks = new Task<int>[]{
                GetUrlContentLengthAsync("http://stackoverflow.com"),
                GetUrlContentLengthAsync("http://www.google.com"),
                GetUrlContentLengthAsync("http://varzesh3.com"),
            };

        Task.WaitAll(tasks);

        foreach (var t in tasks)
        {
            Console.WriteLine(t.Result);
        }
    }
}
```

نمونه کد ۲۱۷: async method

توضیح: این متد مشابه قسمت قبلی می‌باشد، با این تفاوت که وقتی از کلمه ی await استفاده می‌کنیم و متد را جایی صدا می‌زنیم، خط های قبل از این کلمه اجرا می‌شوند و سپس برنامه متوقف می‌شود و دوباره به جایی که متد صدا زده شده می‌رود و دوباره خط های قبل از await اجرا می‌شوند، تا موقعی که متد دیگر جایی صدا زده نشود و حال خطوط بعد از await شروع به اجرا شدن می‌کنند و به محض اینکه تمام شود طول استرینگ دانلود شده را برمی‌گرداند.

خروجی این نمونه کد به صورت زیر می‌باشد:

```
Before-Await: http://stackoverflow.com
Before-Await: http://www.google.com
Before-Await: http://varzesh3.com
After-Await: http://varzesh3.com
After-Await: http://www.google.com
After-Await: http://stackoverflow.com
114210
12835
50305
```

برای درک بهتر مفهوم async و await می‌توانید به لینک های [**asyncmic**] و [**asyncprog**] و [**asyncprogc**] رجوع کنید.

جلسه ۲۴

Thread و Tasks و Async Pattern

زهرا مومنی نژاد - ۱۳۹۹/۷/۱۸

جزوه جلسه ۲۴ام مورخ ۱۳۹۹/۷/۱۸درس برنامه‌سازی پیشرفته تهیه شده توسط زهرا مومنی نژاد. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

در این جلسه به مفهوم زیر به صورت گسترده پرداخته شد: Thread
و همچنین نکاتی در جهت تکمیل مطالب زیر بیان شد: Async، Tasks pattern

۱.۲۴ Await و Async

وقتی ما با Ui سروکار داریم و یک متد که زمان اجرای آن طولانیست (مثل خواندن یک فایل بزرگ و ذخیره آن در پایگاه داده) را در رویداد کلیکِ یک دکمه میگذاریم زمانی که روی آن دکمه کلیک شود رابط کاربری اپلیکیشن قفل شده و به اصطلاح هنگ میکند ، زیرا رابط کاربری و بقیه متد ها در برنامه نویسی همگام (Synchronous) در یک نخ (Thread) از سی پی یو اجرا میشوند پس رابط کاربری تا زمانی که فعالیت متد خاتمه نیابد پاسخی به کاربر نمیدهد.
برنامه نویسی ناهمگام در این شرایط بسیار کارآمد است ، زیرا در این روش رابط کاربری و متد ها به هم متکی نبوده و متد ها به صورت جداگانه اجرا میشوند. این دو برچسب هستند که مشخص میکنند در کدام بخش کد

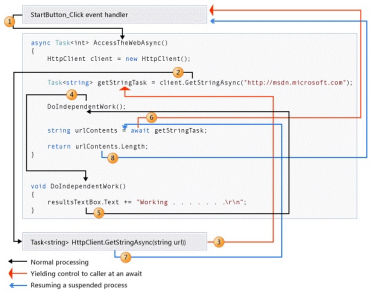
پاسخ دهی باید بعد از اتمام کار از سر گرفته شود.

زمانی که شما در بخشی از کد خود از کلمه کلیدی async و بر روی متدها، عبارات لامبدا یا متدهای بدون نام استفاده می کنید، در حقیقت می گویید که این قطعه کد به صورت خودکار باید به صورت Asynchronous فراخوانی شود و زمان استفاده از کدی که به صورت async تعریف شده، CLR به صورت خودکار thread جدیدی ایجاد کرده و کد را اجرا می کند. اما زمان فراخوانی کدهایی که به صورت async تعریف شده اند، استفاده از کلمه await این امکان را فراهم می کند که اجرای thread جاری تا زمان تکمیل اجرای کدی که به صورت async تعریف شده، می بایست متوقف شود.

نکات مهم در الگوی متد

- کلمه کلیدی async قبل از نوع بازگشتی متد.
- نوع بازگشتی از نوع Task که در اینجا به صورت ز می باشد زیرا خروجی متد یک عدد صحیح است.
- نام متد که با کلمه Async خاتمه یافته است.

نحوه کار متدهای async



شکل ۱.۲۴: نحوه ی کار متد async

توضیحات عکس بالا به ترتیب شماره:

- یک رویداد که متد AccessTheWebAsync را فراخوانی کرده و با استفاده از عملگر await منتظر پایان کار این متد است.

- در داخل متد AccessTheWebAsync یک نمونه از HttpClient ایجاد شده و محتوای سایت با فراخوانی متد GetStringAsync دانلود می شود.

- فعالیتی که در داخل متد GetStringAsync انجام می شود، باعث معلق شدن فرآیند اجرای متد می شود. ممکن است منتظر ماندن برای اتمام این فرآیند باعث قفل شده منابع شود. برای جلوگیری از بروز این مشکل متد GetStringAsync کنترل اجرای برنامه را به متدی که آن را فراخوانی کرده است باز می گرداند. متد GetStringAsync یک Task<TResult> باز می گرداند که در مثال بالا Task<string> است. مقدار بازگشتی از این متد در متغیر getStringTask ذخیره می شود.

- از آنجا که getStringTask هنوز با عملگر await اجرا نشده است، می توان کار دیگری که به نتیجه بازگشتی متد GetStringAsync وابسته نیست را انجام داد. در مثال بالا متد DoIndependent-Work این کار را انجام می دهد.

- متد DoIndependentWork یک متد عادی synchronous است و بعد از انجام کار خود به متدی که آن را فراخوانی کرده است باز می گردد.

- کنترل اجرای برنامه به متدی که AccessTheWebAsync را فراخوانی کرده است باز می گردد و فعالیت هایی که به نتیجه getStringTask وابسته نیستند را انجام می دهد. زمانی که اجرای Task پایان یابد کنترل اجرای برنامه به AccessTheWebAsync باز می گردد و سایر کدها را اجرا می کند.

- رشته موجود در getStringTask (تولید شده توسط متد GetStringAsync) توسط عملگر await گرفته شده و در متغیر urlContents ذخیره می گردد.

- حال متد AccessTheWebAsync محتوای سایت را دارد و می تواند طول آن را محاسبه کرده و به عنوان خروجی باز گرداند. سپس کار متد AccessTheWebAsync کامل می شود و برنامه می تواند کار خود را ادامه دهد.

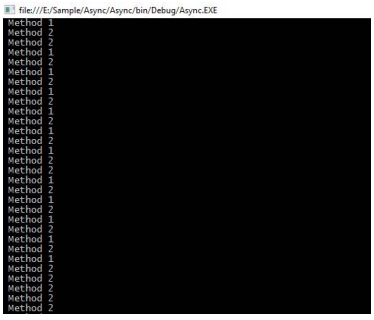
یک مثال برای فهم بهتر از مطالب فوق: در این مثال دو متد به هم متکی نیستند.

```
class Program
    {
        static void Main(string[] args)
        {
            Method1();
            Method2();
            Console.ReadKey();
        }

        public static async Task Method1()
        {
            await Task.Run(() =>
            {
                for (int i = 0; i < 100; i++)
                {
                    Console.WriteLine(1" Method ");
                }
            });
        }


        public static void Method2()
        {
            for (int i = 0; i < 25; i++)
            {
                Console.WriteLine(2" Method ");
            }
        }
}
```

نمونه کد ۲۱۸: کد مثالی در سی شارپ

درکد بالا متد یک و متد دو به هم متکی نیستند و ما از متد Main آن هارا صدا میزنیم و میبینیم که متد ها کاری به یکدیگر ندارند.

خروجی:



شکل ۲.۲۴: خروجی کد مثالی در سی شارپ

ما نمیتوانیم کلمه await را بدون async استفاده کنیم ، و اگر ما در متد Main کلمه Async را استفاده کنیم به این ارور میخوریم:

شکل ۳.۲۴: ارور دریافتی

۲.۲۴ Tasks

- Task.WaitAll()

تا زمانی که به طور کامل انجام شود برنامه را بلاک می کند و در واقع ما نمی توانیم حین اجرا فعالیت های دیگری را انجام دهیم و باید صبر کنیم تا به طور کامل انجام و تکمیل شود و بعد از آن برنامه را ادامه می دهد.

- Task.WhenAll()

برنامه بلاک نمی شود و می توانیم هم زمان با آن قسمت های دیگر را اجرا کنیم و تا زمانی تمامی برنامه اجرا شود ادامه دارد با این تفاوت که دیگر نیازی نیست که برنامه بلاک شود و هر زمان که کار به صورت کامل انجام شد برنامه تسکی را ریترن می کند که در واقع همان شماره پیگیری می باشد و ما با توجه به آن متوجه می شویم که کار به صورت کامل انجام شده است.

- Task.WhenAny()

استفاده از متد ،WhenAny هر کدام از هایTask در حال پردازش که خاتمه یابند، کل عملیات خاتمه خواهد یافت. فرض کنید نیاز دارید تا دمای کنونی هوای منطقه‌ی خاصی را از چند وب سرویس مختلف دریافت کنید. می‌توان در این حالت تمام این‌ها را توسط WhenAny ترکیب کرد و هر کدام که زودتر خاتمه یابد، عملیات را پایان خواهد داد.

کاربرد دیگر WhenAny زمانی است که برای مثال می‌خواهید تعداد زیادی Url را پردازش کنید، اما نمی‌خواهید برای نمایش اطلاعات، تا پایان عملیات تمامی آن‌ها مانند WhenAll صبر کنید. می‌خواهید به محض پایان کار یکی از ،هاTask عملیات نمایش نتیجه‌ی آن‌را انجام دهید.

- Task.Run()

برای فراخوانی متد استفاده می شود.

- Task.Start()

زمانی که Task جدیدی را ایجاد می کنید به وسیله متد Start که برای کلاس Task تعریف شده است می توانید عملیات اجرای Task را شروع کنید.

- Parallel.Invoke()

در واقع Programming Parallel یعنی تقسیم یک مسئله به مسائل کوچکتر و سپردن آن ها به واحد های جداگانه برای پردازش کردن.این مسائل کوچک به صورت همزمان شروع به اجرا می کنند. Programming Parallel وظیفه یا Task را به اجزا مختلفی تقسیم می کند.

- Delay

برای ایجاد وقفه در سی شارپ می توانیم از دستور زیر استفاده کنیم.

```
System.Threading.Thread.Sleep(time);
```

شکل ۴.۲۴: دستور قابل استفاده جهت ایجاد وقفه در سی شارپ

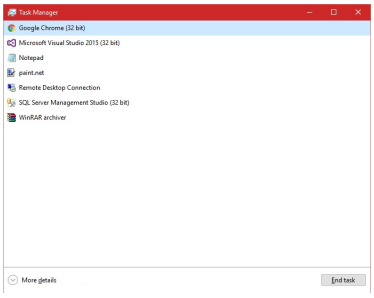
در این کد به جای کلمه time زمان مورد نظر خود را به میلی ثانیه(هزارم ثانیه) وارد کنید.
از Task.Delay یک مکانیزم غیر قفل کننده را جهت صبر کردن به همراه بازگشت یک Task، ارائه می‌دهد. یکی از کاربردهای Delay منهای صبر کردن تا مدت زمانی مشخص، ایجاد مکانیزم timeout است. برای مثال حالت Task.WhenAny را درنظر بگیرید. اگر در اینجا timeout مدنظر ما ۳ ثانیه باشد، می‌توان یکی از هاTask را Task.Delay با آرگومان مساوی ۳۰۰۰ معرفی کرد. اگر هر کدام از هایtask تعریف شده زودتر از ۳ ثانیه پایان یافتند که بسیار خوب؛ در غیر اینصورت Task.Delay معرفی شده کار را تمام می‌کند.

۳.۲۴ Thread

فرض کنید زمانی که در حال تماشای یک فیلم هستید دیگر امکان انجام کارهای دیگر، مثلاً تایپ در برنامه Word یا برنامه نویسی نباشد. اما هیچ گاه این مشکلات برای شما بوجود نمی آید، زیرا سیستم عامل ها به بهترین شکل عملیات هم زمانی را پیاده سازی کرده و به شما این اجازه را می دهند تا در آن واحد نسبت به

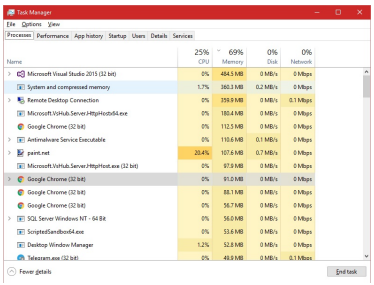
انجام چندین عملیات اقدام کنید. در زبان سی شارپ نیز این امکان برنامه نویسان داده شده است تا نسبت به پیاده سازی عملیات ها به صورت همزمان اقدام کنند. برای اینکار باید از Thread ها استفاده کنیم.

قبل از شروع کد نویسی بهتر است که با یکسری مفاهیم اولیه آشنا شده و سپس به سراغ قابلیت ها برنامه نویسی Asynchronous در زبان سی شارپ برویم. **Process** زمانی که کاربر برنامه ای را اجرا می کند مقداری از حافظه و همچنین منابع به این برنامه تخصیص داده می شوند. اما همانطور که گفتیم یکی از قابلیت های سیستم های عامل این است که می توان چندین برنامه را به صورت همزمان اجرا کرد. یکی از وظایف سیستم عامل تفکیک حافظه و منابع برای هر یک از برنامه های در حال اجرا است که این جدا سازی بوسیله Process ها انجام می شود. در حقیقت هر Process مرزبندی بین برنامه های اجرا است برای جدا سازی منابع و حافظه های تخصیص داده شده. دقت کنید که لزوماً تعداد Process برابر با تعداد برنامه های در حال اجرا نیست، یک برنامه می تواند یک یا چند Process را در زمان اجرا درگیر کند. در سیستم عامل ویندوز می توان از بخش Manager Task لیست برنامه های در حال اجرا و Process ها را مشاهده کرد. در تصویر زیر لیست برنامه هایی که بر روی سیستم من در حال اجرا است را مشاهده می کنید:



شکل ۵.۲۴: لیست برنامه های در حال اجرا

در صورتی که بر روی دکمه More details کلیک کنید می توانید از تب Processes لیست Process های در حال اجرا را مشاهده کنید:



شکل ۶.۲۴: لیست Process های در حال اجرا

هر یک Process های در حال اجرا حافظه، منابع تخصیص داده شده و روند اجرای مربوط به خود را دارند. در تصویر بالا نیز مشخص است، برای مثال در زمان گرفتن عکس بالا، Prcess مربوط به برنامه paint.net مقدار ۴.۲۰ درصد از CPU و همچنین ۶MB.۱۰۷ از حافظه را اشغال کرده است. در اینجا بیشتر به بحث Usage CPU باید دقت کنیم که نشان دهنده میزان استفاده یک Process از CPU است. CPU Usage در حقیقت یک ترتیب اجرا است که اصطلاحاً به آن Thread می گویند. هر Process می تواند شامل یک یا چندین Thread باشد که هر Thread وظیفه انجام یک عملیات خاص را بر عهده دارد. اما زمان اجرای هر Process یک Thread اولیه اجرا می شود که به آن اصطلاحاً Thread Main گفته می شود.

با دو مفهوم Process و Thread آشنا شدیم، اما همانطور که گفتیم در زبان سی شارپ می توانیم Thread هایی ایجاد کنیم که هر Thread یک کار خاص را انجام می دهد. برای کار با Thread ها و برای شروع، با کلاسی به نام Thread که در فضای نام System.Threading قرار دارد کار می کنیم. به صورت زیر می توانیم یک thread جدید با استفاده از کلاس Thread ایجاد کرده و آنرا اجرا کنیم:

```
static void Main(string[] args)
{
    var thread1 = new Thread(Thread1Job);
    var thread2 = new Thread(Thread2Job);
    var thread3 = new Thread(Thread3Job);
    thread1.Start();
    thread2.Start();
    thread3.Start();
}

public static void Thread1Job()
{
    for (int counter = 0; counter < 50; counter++)
    {
        Console.WriteLine(" thread1: "From + counter);
    }
}

public static void Thread2Job()
{
    for (int counter = 0; counter < 50; counter++)
    {
        Console.WriteLine(" thread2: "From + counter);
    }
}

public static void Thread3Job()
{
    for (int counter = 0; counter < 50; counter++)
    {
        Console.WriteLine(" thread3: "From + counter);
    }
}
```

نمونه کد ۲۱۹: کد مثالی در سی شارپ

همانطور که مشاهده می کنید در کد بالا ۳ شئ از نوع Thread ایجاد کردیم و برای پارامتر Constructor متد مورد نظر را ارسال کردیم. Constructor کلاس Thread پارامترش از نوع Delegate است و به همین دلیل می توان یک متد را جهت اجرا در Thread به عنوان پارامتر به آن ارسال کرد. بعد از تعریف thread ها به ترتیب آن را بوسیله متد Start اجرا می کنیم. در تصویر زیر خروجی کد بالا را مشاهده می کنید که کد های Thread ها به صورت همزمان اجرا شدند:

شکل ۷.۲۴: خروجی کد مثالی در سی شارپ

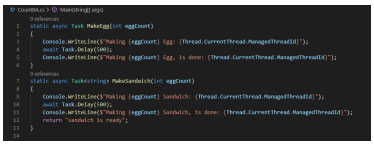
اگر در کد بالا متد ها را بدون استفاده از Thread ها فراخوانی می کردیم Thread۲Job پس از اجرای Thread۱Job اجرا شده و الی آخر.

برای فهم بهتر مثال دیگری را بیان می کنیم:

در این مثال یک فضای ملموس تری بیان شده برای درک کامل تر مبحث فوق که موضوع کلی آن در مورد درست کردن صبحانه است، شامل دو بخش می باشد که اولی درست کردن تخم مرغ می باشد و دومی درست کردن ساندویچ. در این مثال نیاز به استفاده از Thread کاملا حس می شود چون در صورت عدم استفاده ما باید صر کنیم که متد اول(درست کردن صبحانه) تمام شود و بعد برنامه متد دوم را اجرا کند، در صورتی که می توان در طی آماده سازی تخم مرغ، مراحل متد دوم را نیز پیش برد و نباز به وقفه نیست.

می توانیم متد های درست کردن تخم مرغ و ساندویچ را مطابق زیر با توجه به مراحل آن بنویسیم، که صرفا در این قسمت متد ها تعریف می شوند:

یکی از کاربرد های مفید برنامه نویسی این است که می توان فعل مورد نظر را چندین بار تکرار کرد بدون اینکه نیاز باشد آن را چندین بار نوشت، در اینجا نیز اگر بخواهیم مثلا چند تخم مرغ و ساندیچ را درست کنیم می توانیم از کد زیر استفاده کنیم:



شکل ۸.۲۴: متد قابل استفاده جهت درست کردن تخم‌مرغ و ساندویچ به تعداد دلخواه

شکل ۹.۲۴: Main متد فوق

نکته ای که وجود دارد این است که برای این قسمت از کد اگر بخواهیم از ترد ها استفاده کنیم کار سخت می شود زیرا ترد object می گیرد و کست و تبدیل کردن و در نهایت نوشتن متد را سخت تر و پیچیده تر می کند اما می توانیم با استفاده از تسک متد را راحت تر پیاده سازی کنیم، تسک پارامتر می گیرد و برمی گرداند و می توانیم در صورت نیاز مثلا بگوییم وقتی یک تسک تماما شد تسک بعدی را انجام دهد.
در نهایت تسک در بعضی موارد مانند متد فوق کار را برای ما راحت تر می کند.

تا اینجا متد های درست کردن تخم مرغ و ساندویچ و همچنین درست کردن آن ها به تعداد دلخواه را پیاده سازی کردیم و حال برای اجرا و انجام این متد ها نیاز داریم که مانند زیر عمل کنیم:

```
۱   static void Main2(string[] args)
۲   {
۳       EggParam p = new EggParam();
۴       p.EggCount = 4;
۵       Thread tEgg = new Thread(
۶           new ParameterizedThreadStart(MakeEgg)
۷       );
۸       tEgg.Name = Thread" "Egg;
۹       Thread tSandwich = new Thread(MakeSandwitch);
۱۰      tSandwich.Name = Thread" "Sandwitch;
۱۱
۱۲      tEgg.Start(p);
۱۳      tSandwich.Start();
۱۴
۱۵      tEgg.Join();
۱۶      tSandwich.Join();
۱۷      Console.WriteLine(p.TimeSpent);
۱۸      Console.WriteLine(Ready!" is "Breakfast);
۱۹  }
```

نمونه کد ۲۲۰: Main برنامه

برخی اوقات Thread های ایجاد شده به داده های مشترک در سطح برنامه دسترسی دارند و وظیفه ما به عنوان برنامه نویس این است که مطمئن باشیم دسترسی چند Thread به داده های مشترک باعث بروز مشکل نمی شود. برای آشنایی بیشتر با این موضوع شرایطی را در نظر بگیرید که یک متد قرار است در چندین thread مختلف به صورت جداگانه اجرا شود، بعد از شروع کار هر thread زمانبندی اجرا توسط CLR به هر thread به صورت خودکار انجام شده و ما نمی توانیم دخالتی در این موضوع داشته باشیم، ممکن است در این بین اختصاص زمان به یک thread بیش از thread دیگر انجام شود و در این بین خروجی مناسب مد نظر ما ایجاد نمی شود. برای آشنایی با این موضوع متد PrintNumbers که در زیر تعریف کردیم را در نظر بگیرید:

```
public static void PrintNumbers()
{
    Console.Write(" > numbers printing is "{0}, Thread.CurrentThread.Name);
    for (int counter = 0; counter < 10; counter++)
    {
        Thread.Sleep(200*new Random().Next(5));
        Console.Write(,""{0}, counter);
    }
    Console.WriteLine();
}
```

نمونه کد ۲۲۱: متد PrintNumbers

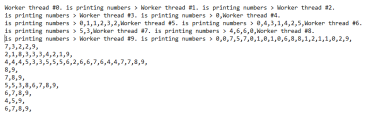
در مرحله بعد متد Main را به صورت زیر تغییر می دهیم تا ۱۰ thread ایجاد شده و سپس کلیه thread ها اجرا شوند:

```
static void Main(string[] args)
{
    Thread[] threads = new Thread[10];

    for (int index = 0; index < 10; index++)
    {
        threads[index] = new Thread(PrintNumbers);
        threads[index].Name = string.Format(#{0}." thread "Worker, index);
    }

    foreach (var thread in threads)
    {
        thread.Start();
    }

    Console.ReadLine();
}
```

نمونه کد ۲۲۲: Main برنامه

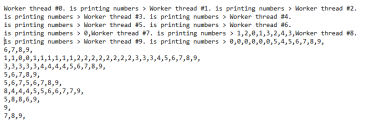
همانطور که مشاهده می کنید کلیه thread ها به صورت همزمان اجرا می شوند، اما پس از اجرا کد بالا،

خروجی برای بار اول به صورت خواهد بود، البته دقت کنید که با هر بار اجرا خروجی تغییر می کند و ممکن است برای بار اول خروجی زیر برای شما تولید نشود:



شکل ۱۰.۲۴: خروجی کد

اگر برنامه را مجدد اجرا کنید خروجی متفاوتی از خروجی قبلی دریافت خواهیم کرد:



شکل ۱۱.۲۴: خروجی کد

همانطور که مشاهده می کنید خروجی های ایجاد کاملاً با یکدیگر متفاوت هستند. مشخص است که در اینجا مشکلی وجود دارد و همانطور که در ابتدا گفتیم این مشکل وجود همزمانی یا Concurrency در زمان اجرای thread هاست. زمابندی CPU برای اجرای thread ها متفاوت است و با هر بار اجرا زمان های متفاوتی به thread ها برای اجرا تخصیص داده می شود.

یکی از راه های مدیریت همزمانی در زمان اجرای Thread ها استفاده از کلمه کلیدی lock است. این کلمه کلیدی به شما این اجازه را می دهد تا یک scope مشخص کنید که این scope باید به صورت synchronized بین thread ها به اشتراک گذاشته شود، یعنی زمانی که یک thread وارد scope ای شد که با کلمه کلیدی lock مشخص شده، thread های دیگر باید منتظر شوند تا thread جاری که در scope قرار دارد از آن خارج شود. برای استفاده از lock شما اصطلاحاً می بایست یک token را برای scope مشخص کنید که معمولاً این کار با ایجاد یک شئ از نوع object و مشخص کردن آن به عنوان token برای synchronization استفاده می شود. شیوه کلی استفاده از lock به صورت زیر است:

```
lock(token)
{
    // all code in this scope are thread-safe
}
```

شکل ۱۲.۲۴: شیوه کلی استفاده از lock

اصطلاحاً می گویند کلیه کدهایی که در بدنه lock قرار دارند thread-safe هستند. برای اینکه کد داخل متد PrintNumbers به صورت thread-safe اجرا شود، ابتدا باید یک شئ برای استفاده به عنوان token در کلاس Program تعریف کنیم:

```
class Program
{
    public static object threadLock = new object();

    ....
```

شکل ۱۳.۲۴: تعریف شئ برای استفاده به عنوان token

در قدم بعدی کد داخل متد PrintNumbers را به صورت زیر تغییر می دهیم:

```
۱  public static void PrintNumbers()
۲  {
۳      lock (threadLock)
۴      {
۵          Console.Write(" > numbers printing is "{0}, Thread.CurrentThread.Name);
۶          for (int counter = 0; counter < 10; counter++)
۷          {
۸              Thread.Sleep(200 * new Random().Next(5));
۹              Console.Write(,""{0}, counter);
۱۰         }
۱۱         Console.WriteLine();
۱۲     }
۱۳ }
```

نمونه کد ۲۲۳: تغییر متد PrintNumbers

با اعمال تغییر بالا، زمانی که thread جدیدی قصد وارد شدن به scope مشخص شده را داشته باشد، باید منتظر بماند تا کار thread جاری به اتمام برسد تا اجرای thread جدید شروع شود. با اعمال تغییر بالا، هر چند بار که کد نوشته شده را اجرا کنید خروجی زیر را دریافت خواهید کرد:

```
Worker thread #0. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #1. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #2. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #3. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #4. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #5. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #6. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #7. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #8. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
Worker thread #9. is printing numbers > 0,1,2,3,4,5,6,7,8,9,
```

شکل ۱۴.۲۴: خروجی کد

برای کسب اطلاعات بیشتر ودرک بهتر مفاهیم فوق می توانید به لینک های زیر مراجعه کنید:

[**Asynchronous**] [**task**] [**parallel**] [**thread**] [**asyncprogc**] [**asyncprog**] [**asyncmic**]

جلسه ۲۵

LINQ : Language Integrated Query

فاطمه میرجلیلی - ۱۳۹۹/۲/۲۷

جزوه جلسه ۲۵ام مورخ ۱۳۹۹/۲/۲۷درس برنامه‌سازی پیشرفته تهیه شده توسط فاطمه میرجلیلی.

در ادامه پرداختن به موضوع Delegate واستفاده عمده از Events ، Multi-Threading ، و Strategy Pattern * که در جلسات پیشین گفته شد به معرفی یکی از feature های زبان سی شارپ به نام `Linq` می پردازیم.

Linq کوتاه شده عبارت `Language` `Integrated` `Query` به معنای زبان جست‌وجوی یکپارچه است.دلیل این نامگذاری ناشی از کاربرد linq در استخراج داده از یک منبع داده است.همانطور که گفته شد linq از feature های زبان سی‌شارپ به شمار می‌رود و معادلی برای ان در زبان هایی مثل java موجود نمی‌باشد.

*هنگامی که در نوشتن کد یک فانکشن به عنوان ورودی به یک فانکشن دیگر داده شود در واقع از `Strategy Pattern` استفاده شده است.

در ابتدا لازم است مواردی برای مقدمه توضیح داده شود .

`Static Class:`

کلاس استاتیک کلاسی است که همه member variable ها و function ها یی که در آن استفاده می‌شود باید استاتیک باشد.

در نمونه کد زیر یک مثال از یک متد استاتیک را داخل کلاس استاتیک Ext مشاهده می کنید که به این متد `extention method` گفته می شود.

```
 static class Ext
{
    public static int Next(this int n, int offset)
    {
        return n+offset;
    }
}
```

نمونه کد ۲۲۴: دراینجا متد استاتیک Next به عنوان یک گزینه برای متغیر n تعریف می شود.

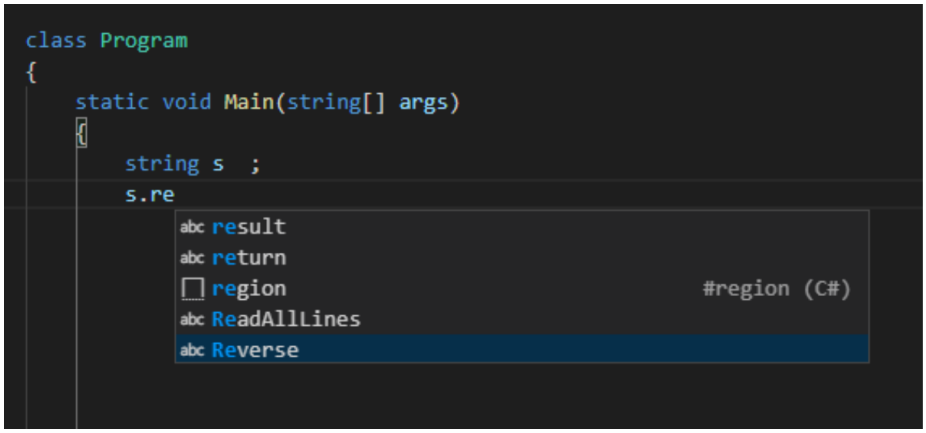
طریقه استفاده از آن به صورت زیر است.

```
static void Main(string[] args)
{
    int w = 5;
    Console.WriteLine(w.Next(4));
}
```

برای مثال دیگر تابع استاتیک Reverse .را به صورت زیر پیاده سازی می کنیم

```
public static string Reverse(this string str)
{
    char[] chs = str.ToCharArray();
    for(int i =0 ; i< chs.Length/2 ; i++)
        (chs[i] , chs[chs.Length-i-1])=(chs[chs.Length-i-1] ,chs[i]);
    return new string(chs);
}
```

حال می بینیم برای هر متغیر از نوع string یک گزینه به نام Reverse موجود می باشد




```
string s = "AliHossein";
Console.WriteLine(s.Reverse());
```

نمونه کد ۲۲۵: طریقه استفاده از `extention method` در کد

در اینجا اگر بخواهیم متد Next را به گونه ای بنویسیم که دو پارامتر از ورودی بگیرد می‌توانیم آن را به صورت زیر بنویسیم و Next را از داخل کلاس Ext صدا بزنیم .

```
int sw = Ext.Next(5, 4);
```

برای اینکه کد ما مرتب تر شود و برای هر نوع از متغیر قابل استفاده باشد می توانیم آن را به صورت زیر نیز بنویسیم

```
public static _Type[] Reverse<_Type>(this _Type[] chs)
{
    for(int i=0; i<chs.Length/2; i++)
        (chs[i], chs[chs.Length-i-1]) = (chs[chs.Length-i-1], chs[i]);
    return chs;
}
public static string Reverse(this string str) =>
    new String(str.ToCharArray().Reverse());
```

- یکی دیگر از مباحث مهمی که در مقدمه `linq` باید گفته شود چگونگی تعریف و استفاده از انواع

مختلف متغیر هایی از نوع `Tuple` می باشد. Tuple درواقع یک class و reference type است.

۱.۲۵ انواع تعریف `Tuple`

- در این روش در واقع یک کلاس جداگانه ایجاد میشود و اطلاعات داخل آن ذخیره می شود .

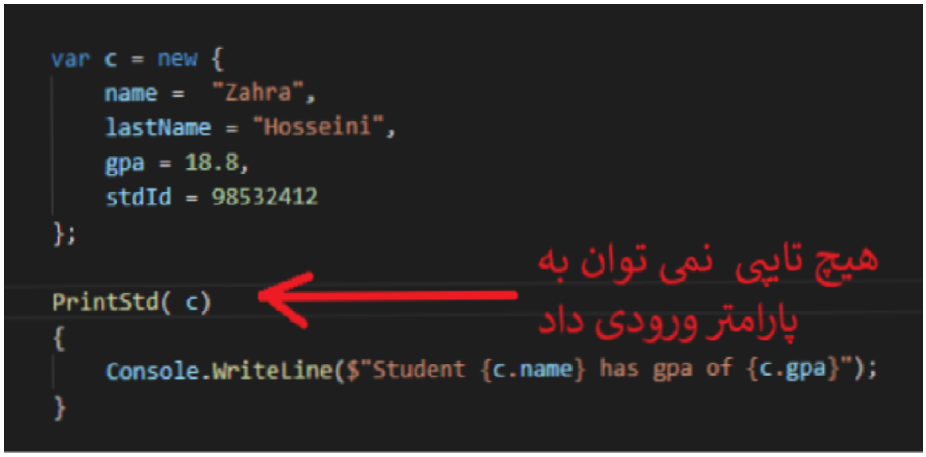
```
۱ var t = Tuple.Create("Zahra", "Hosseini", 8.19, 98521343);
```

از مشکلات استفاده از این تعریف میتوان به این اشاره کرد که اطلاعات داده شده به Tuple موقع استفاده به صورت `item1,item2,....` نشان داده می شوند .

برای حل این مشکل میتوان از `anonymous class` که به صورت زیر تعریف میشود استفاده کرد .

```
۱ var c = new {
۲     name =  "Zahra",
۳     lastName = "Hosseini",
۴     gpa = 8.18,
۵     stdId = 98532412
۶ };
۷ Console.WriteLine($"{c.gpa}" of gpa has {c.name} "Student);
```

حال میتوان برای هر کدام از اطلاعات نام دلخواه انتخاب کرد که با آن شناخته شود.این تعریف نیز محدودیت هایی دارد مثلا اگر بخواهیم این Tuple را به عنوان ورودی یک تابع به کاربریم هیچ تایپی (string ,int ,Tuple,var,...) برای آن قابل تعریف نیست .



- `Value Tuple`

نوع دیگری از Tuple میتوان تعریف کرد که برخلاف حالت قبل یک Value type از نوع `Struct` است .این نوع Tuple محدودیت نوع قبلی را ندارد و هنگام استفاده از آن به عنوان ورودی فانکشن میتوان نوع آن را مشخص کرد .در شکل زیر یک مثال آورده شده است .

```
(string name, string lastName, double gpa, int stdId) student =
("Zahra", "Hosseini", 8.19, 98521343);

static void PrintStd((string name, string lastName, double gpa, int stdId) std)
{
    var (n, l, g, s) = std;
    (string na, string la, double gp, int st) = std;
    Console.WriteLine(n, l, g, s);
}
```

از اطلاعات داخل این نوع Tuple میتوان به صورت item1,item2,... ویا ایجاد نام جداگانه برای هر property استفاده کرد.همچنین میتوان از عملگر های `= =` ویا `=!` برای مقایسه مقادیر دو Tuple استفاده کرد .

```
(string, string, double, int) std2 = student;
string name = "ali";
string lastName = "Zahraei";
double gpa = 9.19;
int stdId = 98521234;

var std3 = (name, lastName, gpa, stdId);

if ( std2 != std3 )
{
    Console.WriteLine("equal");
}
```

۲.۲۵ Deconstruct کردن یک Tuple

deconstruct کردن یک Tuple به این معنی است که هر کدام از `item` های مختلف یک Tuple را در متغیر های محلی تعریف می کنیم این کار این قابلیت را به ما می دهد تا از از مقادیر item ها به صورت جداگانه استفاده کنیم.

```
static void PrintStd((string name, string lastName, double gpa, int stdId) std)
{
    var (n, l, g, s) = std;
    (string na, string la, double gp, int st) = std;
    Console.WriteLine(n, l, g, s);
}
```

۳.۲۵ LINQ

linq کوتاه شده عبارت `Language Integrated Query` به معنای زبان پرس و جوی یکپارچه است.که برای استخراج و استفاده از داده های یک `Database` میتوان از آن کمک گرفت.عمده لینک از extention method هایی روی تایپ `iEnumerable` تشکیل شده است.

در ابتدا برای استفاده از linq در کد خود باید از `using System.linq ;` در ابتدای صفحه استفاده شود.

```
using System.Linq;
```

در نمونه کد زیر یک مثال از یک extention method و معادل آن در linq آورده شده است.

```
public static double[] ToDouble(this string[] list)
{
    double[] result = new double[list.Length];
    for(int i=0; i<list.Length; i++)
        result[i] = double.Parse(list[i]);
    return result;
}

string[] nums = {"6.15", "9.19", "2.17", "4.13", "6.15"};
double[] numsParsed = nums.ToDouble();

var numParsedWithLinq = nums.Select(s => double.Parse(s)) ;
```

`Select():`

متد Select() یکی از متد های پرکاربرد linq است که کاربرد آن دقیقا معادل کلمه `select` می‌باشد. این متد یک `Lambda expression` به عنوان پارامتر ورودی می پذیرد. که عموما نوع برگشتی آن یک iEnumerable شامل اطلاعات درخواستی است.

در این مثال Select یک iEnumerable ویک function به عنوان ورودی گرفته و پس از اجرای function روی تک تک اعضا یک iEnumerable به عنوان خروجی تحویل می دهد.

```
public static IEnumerable<_OutType> MySelect<_OutType, _InType>(
    this IEnumerable<_InType> list, Func<_InType, _OutType> fn)
{
    foreach(var input in list)
        yield return fn(input);
}
```

نمونه کد ۲۲۶: شرح متد Select به صورت یک function

`OrderBy():`

متد OrderBy() اطلاعات موجود را بر اساس `key selector` که در بطن به آن داده می شود به صورت صعودی مرتب می کند .

```
        .OrderBy(n => (n))
```

نمونه کد ۲۲۷: در این جا key selector همان object n است.

`OrderByDescending():`

این متد همانند متد OrderBy() است با این تفاوت که اطلاعات به صورت نزولی مرتب میشوند.

```
.OrderByDescending(n => n)
```

`Tolist():`

این متد داده های موجود را به صورت یک لیست به خروجی تحویل می دهد.

`ForEach():`

زمانی که داده ها به صورت یک لیست موجود باشند متد ForEach براساس Action داده شده به آن عمل موردنظر را روی تک تک داده های لیست انجام می‌دهد. تفاوت این متد با متد های گفته شده در این است که ورودی آن به جای `Func` یک `Action` می باشد.

```
.ToList()
.ForEach(n => WriteLine(n));
```

`Take(int x);`

این متد به اندازه x تا از داده های موجود را برمیگرداند.

`Where()`

این متد برای فیلتر کردن داده ها براساس دارا بودن یک ویژگی یا شرط ورودی اش به کار می رود.

```
.Where(t => t.country != "INVALID")
```

`Skip(int x);`

این متد x داده اول را از iEnumerable شده به آن حذف میکند.

`Trim()`

این متد می تواند روی یک پارامتر از نوع `string` صدا زده شود و میتواند چند کاراکتر را به عنوان ورودی بپذیرد .درصورت وجود کاراکتر ها در ابتدا یا انتهای string موجود حذف میشوند.

```
Trim('"', ' ');
```

`SelectMany():`

این متد کاربردی شبیه به `Select` دارد با این تفاوت که میتواند یک عنصر را به بیش از یک عنصر تبدیل کند و در خروجی بازگرداند . برای فهم بهتر عملکرد این متد می توانید به مثال زیر توجه کنید.

```
.SelectMany(t => {
    return new (string country, double percent)[]{
    (t.country, t.male),
    (t.country, t.female)};
})
```

نمونه کد ۲۲۸: در این جا ورودی یک Tuple t و خروجی دو Tuple با item های دلخواه از Tuple اولیه می باشد.

`Average():`

این متد میانگین داده هارا بر اساس پارامتر ورودی اش برمی گرداند.

```
    .Average(t => t.percent);
```

نمونه کد ۲۲۹: در این مثال میانگین داده ها براساس percent بدست می‌اید

```
static void Main(string[] args)
{

    File.ReadAllLines(address" @"Database)
        .Skip(2)
        .Select(l => {
            var toks = l.Split(',').Select(t => t.Trim('"', ' ')).ToArray();
            try
            {
                return (
                        country:toks[0],
                        year:int.Parse(toks[1]),
                        both:double.Parse(toks[2]),
                        male:double.Parse(toks[3]),
                        female:double.Parse(toks[4])
                        );
            }
            catch
            {
                return ("INVALID",0, 0, 0, 0);
            }
        })
        .Where(t => t.country != "INVALID")
        .SelectMany(t => {
            return new (string country, double percent)[]{
                (t.country, t.male),
                (t.country, t.female)};
        })
        .OrderBy(t => t.percent)
        .OrderByDescending(t => t.female - t.male)
        .Take(20)
        .ToList()
        .ForEach(t => WriteLine(t));
}
```

نمونه کد ۲۳۰: نمونه استفاده از تمام متد های ذکر شده

جلسه ۲۶

لینک

محمد حسین رجبی - ۱۳۹۹/۳/۱۶

۱.۲۶ دیزاین پترن (الگوی طراحی) چیست ؟

- دیزاین پترن ها راه حل های رایج و قابل استفاده برای مشکلات رایج در طراحی نرم افزار هستند
- دیزاین پترن ها راه کارهایی بهینه و با قابلیت استفاده مجدد برای مشکلات برنامه نویسی هستند.

دیزاین پترن ها سرعت کد نویسی رو بالا میبرند چونکه الگوهایی هستند از پیش آماده و تست شده که در اختیار برنامه نویس ها برای انواع موقعیت ها استفاده میشوند.

دیزاین پترن ها به خودی خود مشکلات رو حل نمیکنند بلکه ابزار مناسبی هستن که به ما در حل مشکلات در برنامه نویسی کمک میکنند .

در این جلسه حول فایل `gdp.csv` که مخفف `gross Domestic product` یا همان فعالیت اقتصادی مطالب جدیدی در مورد لینک یاد بگیریم .

۲.۲۶ شکل فایل بدین صورت است :

- خط اول : `Country Name,Country Code,Year,Value`
- ما بقی خط ها : `Arab World,ARB,1968,25760683041.0857`

۰

در نمونه کد ۱ ابتدا کل خط های فایل را خوانده سپس با توجه به این که خط اول فقط الگو را مشخص میکند با `skip(1)` از آن گذر میکنیم. در ادامه با `where` خط هایی که دارای کلمه ی `iran` هستند را جدا میکنیم سپس با متد `properSplit` هر خط را به ۴ قسمت تبدیل کرده و به عنوان خروجی یک `tuple` که به ترتیب متشکل از نام کشور و سال و فعالیت اقتصادی است را بر میگردانیم.

توجه شود که این قسمت مانند یک تابع فقط تعریف شده است و تا وقتی که صدا زده نشود اجرا نمیشود. پس تا وقتی در خط ۱۰ صدا زده نشود اگر اشتباهی در آن باشد مشخص نمیشود . اصطلاحا به این حالت `Lazy Evaluation` میگویند که مربوط به `IEnumerable` است.

```
۱    var iranLines = File.ReadAllLines(@"path")
۲                        .skip(1)
۳                        .where(l => l.Tolower().Contains("iran"))
۴                        .select(l => {
۵                            var toks = properSplit(l).ToArray();
۶                            return (
۷                            Country : toks[0],
۸                            year: int.Parse(toks[2]),
۹                            gdp : double.Parse(toks[3])
۱۰                           );
۱۱                       });
```

نمونه کد ۲۳۱: تعداد خط های شروع شده با ایران

تابع `properSplit` به عنوان ورودی یک خط را گرفته .

سپس با `Split('"', StringSplitOptions.RemoveEmptyEntries)` بر اساس دبل کوتشن قسمت کرده و قسمت های خالی را حذف میکند و قسمت اول که نام کشور است را بر میگرداند.

در مرحله بعد قسمت دوم را بر اساس `,` تقسیم کرده و بخش های بعدی را برمیگرداند .

```
private static IEnumerable<string> properSplit(string line)
{
    var toks = line.Split('"', StringSplitOptions.RemoveEmptyEntries);
    yield return toks[0];
    foreach (var t in toks[1].Split(',', StringSplitOptions.RemoveEmptyEntries))
    {
        yield return t;
    }
}
```

نمونه کد ۲۳۲: متد `properSplit`

یکی دیگر از توابع قابل استفاده `OrderBy()` است که به صورت دیفالت از کم به زیاد بر اساس متغیر خواسته شده `sort` میکند. در این مثال میتوان با `OrderBy(t => t.gdp)` تمامی `tuple` ها را بر اساس فعالیت اقتصادی مرتب کرد .

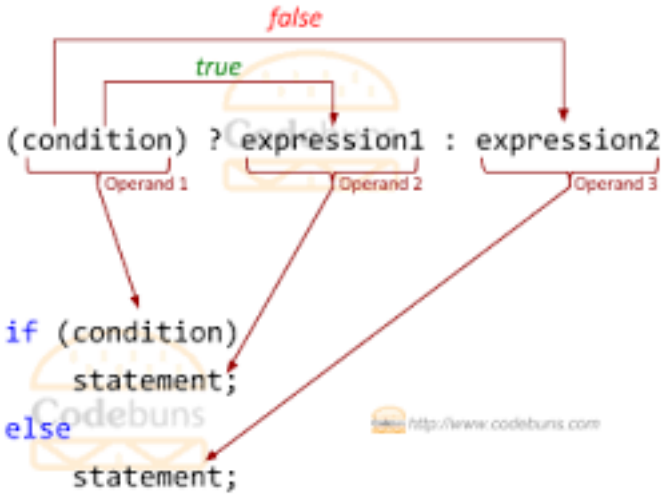
در ادامه شما میتوانید با `First()` و `Last()` به اولین و آخرین عضو دسترسی داشته باشید .

تابع `Average()` نیز مانند `OrderBy()` یک `selector` میگیرد و بر اساس آن میانگین را محاسبه میکند. در مثال حاضر با `Average(t => t.gdp)` میتوان میانگین فعالیت های اقتصادی را محاسبه کرد.

دو تابع `Max() , Min()` نیز یک `selector` گرفته و بدون مرتب کردن بزرگترین و کوچک ترین مقادیر را بر اساس `selector` داده شده پیدا میکنند .

در این مثال با `Max(t => t.gdp)` یا `Min(t => t.gdp)` می توان این دو مقدار را پیدا کرد .

۳.۲۶ Ternary operator



۴.۲۶ Aggregate

روشی برای جمع بندی است که دو ورودی گرفته و با الگوریتم داده شده فقط یه خروجی میدهد .

```
result.Aggregate( (t1, t2) => t1.gdp > t2.gdp ? t1 : t2).ToString()
result.Aggregate( (t1, t2) => t1.gdp < t2.gdp ? t1 : t2).ToString()
```

نمونه کد ۲۳۳: Aggregate

در مثال بالا Aggregate دو t2 tuple , t1 را گرفته و در هر مرحله tuple ای که gdp بیشتری دارد را پس میدهد و این کار ادامه پیدا میکند تا در نهایت tuple ای که بیشترین gdp را داراست را پیدا کند. توجه شود که result مجموعه ی تمام tuple هاست .

۵.۲۶ GroupBy

یکی دیگر از موارد قابل استفاده در این بخش GroupBy است که شما میتوانید یک `key selector` به آن دهید این متد اعضایی که key selector یکسان دارند را در یک گروه قرار دهد . البته به صورت دستی میتوان این کار را با دیکشنری انجام داد که نیازمند وقت و توجه بیشتری است.

برای مثال میتوان به کد زیر اشاره کرد :

```
result.where(t => t.year == 2012)
           .GroupBy(t => (int) (t.gdp / 100))
            .OrderByDescending(g => g.Key)
            .ToList()
            .ForEach(g => {
                g.Key.ToString().Dump($"  ,{g.Count()} "Key:);
                g.ToList().ForEach(t => t.Dump());
            });
```

نمونه کد ۲۳۴: GroupBy

در این مثال ابتدا tuple هایی که سالشان برابر ۲۰۱۲ است را جدا کرده سپس آن هایی که حاصل تقسیم فعالیت اقتصادیشان بر صد برابر یک مقدار است را در یک گروه قرار میدهیم . و در نهایت با الگو دلخواه در خروجی نمایش میدهیم .

در ادامه با استفاده از فایل `population.csv` جمعیت هر کشور را در هر سال در tuple قرار میدهیم .

۶.۲۶ شکل فایل بدین صورت است :

- خط اول : `Country Name,Country Code,Year,Value`
- ما بقی خط ها : `Arab World,ARB,1960,92490932`

با کد زیر میتوان به این نتیجه رسید :

```
File.ReadAllLines(@"path")
        .skip(1)
        .select(l => {
            var toks = properSplit(l).ToArray();
            return (
                    Country : toks[0],
                    Code : toks[1],
                    year: int.Parse(toks[2]),
                    pop : int.Parse(toks[3])
                );
            });
```

نمونه کد ۲۳۵: population

که روند مشابهی با کد قبل داد .

Join ۷.۲۶

برای ترکیب دو لیست به کار میرود . بطوریکه به ازای عضوی که در هر دو لیست وجود دارد یک `Func` خاصی را روی آنها انجام دهد .

برای درک بهنر به مثال زیر توجه کنید:

```
gdpResult.Join(popResult,
    t => (code: t.code, year: t.year),
    t => (code: t.code, year: t.year),
    (t1, t2) => (t1.country, t1.year, normgdp:t1.gdp / t2.pop)
)
.Where(t => t.year == 2012)
.OrderBy(t => t.normgdp)
.ToList();
```

نمونه کد ۲۳۶: join

در این مسئله gdpResult (لیستی از تاپل های gdp) را میخواهیم با popResult (لیستی از تاپل های pop) ترکیب کنیم . بدین منظور در خط های ۲ و ۳ میگوییم tuple هایی را میخواهیم ترکیب کنیم که year , code یکسان داشته باشند (`key selector`) و در خط ۴ چگونگی ترکیب این دو tuple را بیان کردیم که به عنوان خروجی یک tuple به ترتیب با مقادیر نام کشور و سال و حاصل تقسیم فعالیت اقتصادی بر جمعیت را برگرداند .

و در نهایت توابع زیر را نیز در نظر بگیرید :

- `Distinct` برای حذف عناصر تکراری به کار میرود .
- `concat` دو لیست را پشت هم میاورد یا میچسباند.
- `Contains` چک کردن وجود یک عضو در لیست
- `Any` آیا عضوی هست که یک شرط را برقرار کند .
- `All` آیا همه ی اعضا شرطی را برقرار میکنند.

جلسه ۲۷

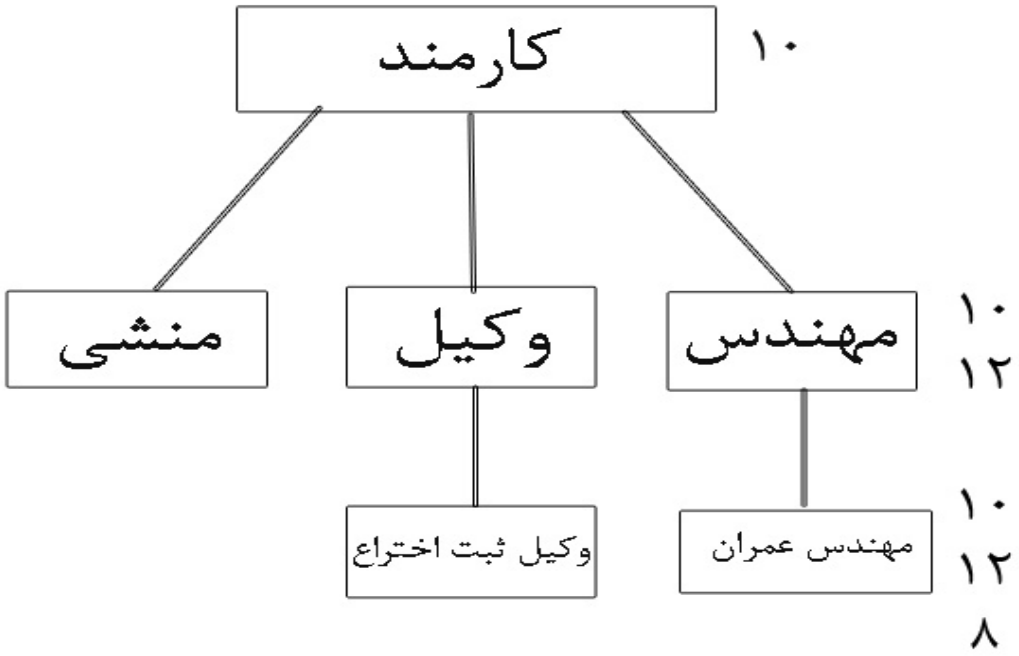
وراثت

امیرحسین درخشان - ۱۳۹۹/۳/۳

جزوه جلسه ۲۷ام مورخ ۱۳۹۹/۳/۳ درس برنامه‌سازی پیشرفته تهیه شده توسط امیرحسین درخشان. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

۱.۲۷ کاربرد وراثت

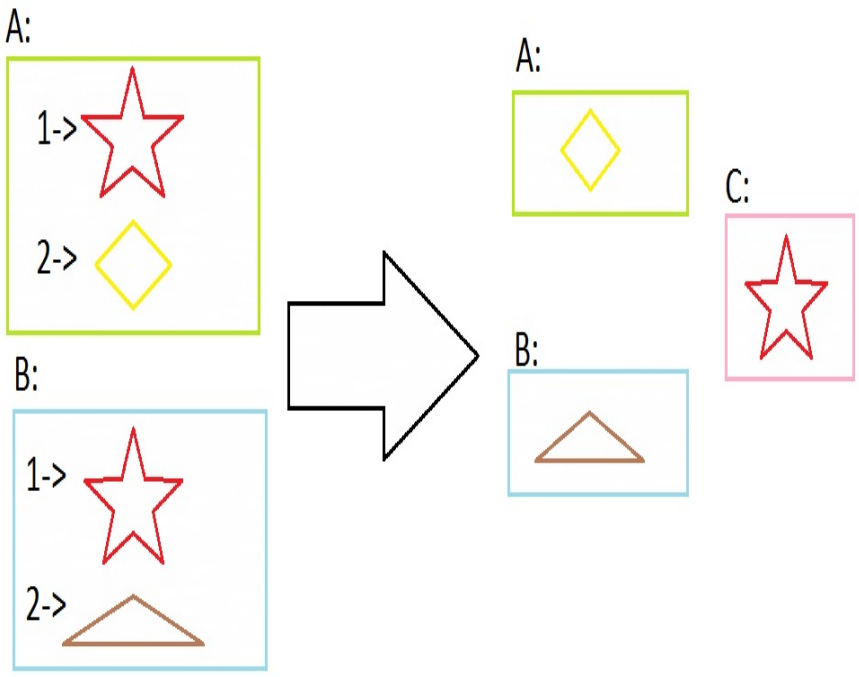
وراثت در برنامه نویسی برای اضافه کردن ویژگی های یک کلاس به کلاسی دیگر هست یعنی وقتی می گوییم کلاس B از کلاس A ارث بری میکند یعنی B تمام ویژگی های A را دارد برای روشن تر شدن موضوع به عکس و مثال زیر توجه کنید.



فرض کنیم برای کارمندان در یک شرکت دفترچه راهنمایی وجود دارد که ۱۰ صفحه دارد حال برای مهندسین ۱۲ صفحه دیگر و مخصوص خود مهندسین وجود دارد و از انجا که هر مهندسی خود کارمند هست پس ان ۱۰ صفحه مربوط به کارمندان هم در دفترچه انها وجود خواهد داشت و حال مهندسین عمران نیز مثلا ۸ صفحه مخصوص خود را دارند و از انجا که خود مهندس نیز هستند ان ۱۲ صفحه مربوط به مهندسین را هم دارند و از انجا که هر مهندسی خود کارمند هست پست ان ۱۰ صفحه را هم دارند در تصویر بالا اگر مهندس عمران را یک کلاس در نظر بگیریم از کلاس مهندس ارث بری میکند و کلاس مهندس هم از کلاس کارمند .(این مطلب برای وکیل و منشی نیز به همین طور تعمیم داده میشود)

۲.۲۷ ضرورت استفاده از وراثت

دو کلاس A و B را در نظر بگیرید که در بخشی از کد مشترک هستند حال برای جلوگیری از تکرار این بخش میتوان ان را جدا کرد و در موارد لزوم از ان استفاده کرد(نمای کلی این کار در شکل زیر امده است) در وراثت نیز چنین عملی صورت میگیرد.



۳.۲۷ توضیح کلی و نحوه استفاده از وراثت

در یک مثال واقعی دو کلاس Security و Engineer را در نظر بگیرید

```
namespace note
{
    class Security
    {
        string Name;
        long Salary;
        public string DoType()
        {
            return $Type" "{this.Name};
        }
    }
}
```

نمونه کد ۲۳۷: class security

```
namespace note
{
    class Engineer
    {
        string Name;
        long Salary;
        public string DoBuild()
        {
            return $Build" "{this.Name};
        }
    }
}
```

نمونه کد ۲۳۸: class Engineer

این دو کلاس در دو خط اولشان با هم یکسان هستند پس میتوان این دو خط را در کلاس دیگری به نام Employee قرار داد

```
namespace note
{
    class Employee
    {
        public string Name;
        public long Salry;
    }
}
```

نمونه کد ۲۳۹: class Employee

حال با توجه به توضیحات گفته شده کلاس های Security و Engineer میتوانند از کلاس Employee ارث ببرند نحوه این کار با استفاده از `:` میباشد

```
namespace note
{
    class Security:Employee
    {
        public string DoType()
        {
            return $Type" "{this.Name};
        }
    }
}
```

نمونه کد ۲۴۰: class Security

```
namespace note
{
    class Engineer:Employee
    {
    public string DoBuild()
    {
        return $Build" "{this.Name};
    }

    }
}
```

نمونه کد ۲۴۱: class Engineer

در این حالت کلاس Employee را کلاس پدر یا base class و به کلاس های Security و Engineer که از آن ارث برده اند کلاس فرزند یا derived class گویند. این نکته را هم در نظر داشته باشید که در زبان برنامه نویسی C# یک کلاس تنها از یک کلاس میتواند ارث ببرد .

۴.۲۷ استفاده از Constructor و توابع در کلاس های فرزند

کلاس Employee را به گونه زیر در نظر بگیرید

```
public class Employee
{
    public string Name;
    public long Salary;
    public Employee(string name,long salary)
    {
        this.Name=name;
        this.Salary=salary;
    }
    public void AddSalary(long adding)
    {
        this.Salary+=adding;
    }
}
```

نمونه کد ۲۴۲: class Employee

حال کلاس Engineer را هم در نظر بگیرید که از کلاس Employee ارث میبرد و علاوه بر ان دارای یک شی به نام Field از نوع string می باشد حال در constructor این کلاس باید از `:base` استفاده کرد

```
using System;
namespace note
{
    public class Engineer : Employee
    {
        public string Field;
        public Engineer(string name,long salary,string field)
            : base(name, salary)
        {
            this.Field=field;
        }
    }
}
```

نمونه کد ۲۴۳: class Engineer

در حال حاضر میتوان از متد AddSalary در Engineer استفاده کرد اما فرض کنیم بخواهیم این متد فرق داشته باشد مثلا Salary را در عدد داده شده ضرب کند برای این کار بایستی در کلاس Employee برای این متد از virtual استفاده کرد.

```
public virtual AddSalary(long adding)
{
    this.Salary+=adding;
}
```

نمونه کد ۲۴۴: addsalary virtual

و در کلاس Engineer باید برای این متد از override استفاده کنیم.

```
public override void AddSalary(long multing)
{
    this.Salary*=multing;
}
```

نمونه کد ۲۴۵: addsalary override

یا اگر بخواهیم در کلاس Engineer به همان صورت باشد اما به شرط خاصی مثلا اگر خقوق فعلی اش زیر ۱۰۰۰۰۰۰ بود به ان اضافه شود باید این گونه کار کرد.

```
public override void AddSalary(long adding)
{
    if(this.Salary<1_000_000)
        base.AddSalary(adding);
}
```

نمونه کد ۲۴۶: addsalary if

یعنی به طور کلی با base.method به متد virtual خواهیم رسید.

۵.۲۷ استفاده از protected

اگر یک شی از کلاس پدر private باشد در ان صورت در کلاس های دیگر قابل دسترسی نیست مثلا در مثال Employee و Engineer فرض کنیم به جای public string Name از private string Name استفاده میکردیم در این صورت this.Name در کلاس Engineer معنا نخواهد داشت پس در چنین حالتی اگر بخواهیم تنها کلاس های فرزند یا derived class ها بتوانند به یک شی دسترسی داشته باشند به جای private باید از `protected` استفاده شود به عبارت دیگر زمانی که از protected برای یک شی استفاده می‌کنیم تنها در کلاسهای فرزند ان شی قابل دسترسی است و در کلاس های دیگر قابل دسترسی نیست.

۶.۲۷ استفاده از abstract

وقتی برای یک کلاس مثلا کلاس Employee از `abstract` استفاده میکنیم به این صورت که abstract class Employeeبه این معناست که تنها در کلاس هایی که از ان ارث برده اند میتوان از ان استفاده کرد یعنی چنین کلاسی تنها برای ارث بری کلاس های دیگر ساخته شده است و در کلاس هایی که از ان ارث نبرده اند نیمتوان از new Employee() استفاده کرد.

Polymorphism ۷.۲۷

به مثال زیر توجه کنید

```
using System.Collections.Generic;
using System;

namespace polymorphism
{
    public class Shape
    {
        public int X;
        public int Y;
        public int Height;
        public int Width;
        public Shape(int x,int y,int height,int width)
        {
            (X,Y,Height,Width)=(x,y,height,width);
        }
        public virtual void Draw()
        {
            Console.WriteLine(tasks" drawing class base "Performing);
        }
    }
    public class Triangle : Shape
    {
        public Triangle(int x, int y, int height, int width)
            : base(x, y, height, width)
        { }
        public override Draw()
        {
            Console.WriteLine(Triangle" "Drawing);
            base.Draw();
        }
        public int Area()
        {
            return Height*Width/2;
        }
    }
    public class Circle : Shape
    {
        public Circle(int x, int y, int height, int width)
            : base(x, y, height, width)
        {}
        public override Draw()
        {
            Console.WriteLine(Circle"  "Drawing);
            base.Draw();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Triangle triangle=new Triangle(1,2,3,4);
            Circle circle=new Circle(5,6,7,8);
            Shape shape=triangle;
            List<Shape> shapes=new List<Shape>{circle,triangle};
        }
    }
}
```

نمونه کد ۲۴۷: Polymorphism

همانطور که مشاهده کردید میتوان triangle و یا circle را به عنوان shape معرفی کرد یعنی به طور کلی کلاس های فرزند میتوانند با نام کلاس پدر معرفی شوند مثل خط ۵۱ اما در این حالت تنها از متد هایی که در کلاس پدر هست میتوان استفاده کرد مثلا نمیتوان در shape از ()shape.Area استفاده کرد

۸.۲۷ استفاده از Seald

اگر برای یک کلاس مثلا کلاس Engineer از `seald` استفاده کنیم به این صورت که seald class Engineer یعنی هیچ کلاس دیگری نمیتواند از این کلاس ارث ببرد.

جلسه ۲۹

Patterns Design

زهرا امیری - ۱۳۹۹/۰۵/۱۴

مطالب مطرح شده در این جلسه به شرح زیر است:

- استفاده از Design Patterns

۱.۲۹ Design Patterns

: Design Patterns

Patterns Design در واقع مجموعه ای از بهترین راه حل‌های مشکلات متداول در فرآیند برنامه نویسی است.

Pattern Design ها انواع مختلفی دارند. برای نمونه می توان به استفاده از event ها و delegate ها ، Single Responsibility Principle اشاره کرد.

*برای مشاهده مطالب بیشتر درمورد patterns design می‌توانید به این سایت رجوع کنید. [**designpattern**] [**Design**]

مثال:هدف این تمرین طراحی یک ماشین حساب است که بتواند عبارات ریاضی را با خواندن از فایل محاسبه کند.برای درک بهتر به تصویر زیر مراجعه کنید.

ورودی:

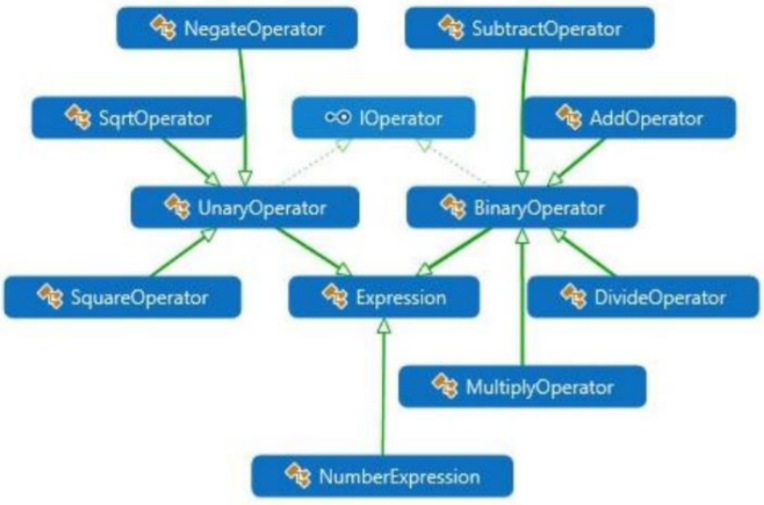
```
Add
Mul
3
4
Sqrt
Sub
5
4
```

خروجی:

```
(3*4) + sqrt(5-4)
13
```

شکل ۱.۲۹: ورودی و خروجی

این دیاگرام رابطه بین کلاس ها و اینترفیس ها را مشخص می کند.



شکل ۲.۲۹: diagram class

بر اساس پیاده سازی زیر اینترفیس IOperator فقط یک ویژگی symbol دارد.مانند: + ، - ، * .
.(نمونه کد ۲۴۸)

```
namespace OOCalculator
{
    public interface IOperator
    {
        string OperatorSymbol { get; }
    }
}
```

نمونه کد ۲۴۸: پیاده سازی IOperator

توضیح: در ادامه بر اساس شکل ۲.۲۹ جهت فلش ها،به شرح رابطه بین کلاس ها می پردازیم.

- کلاس های BinaryOperator و UnaryOperator اینترفیس `IOperator` را پیاده سازی می کنند.
- کلاس های SubtractOperator و AddOperator و DivideOperator هر کدام نوعی `BinaryOperator` یا عملگر دوگانه هستند.
- کلاس های NegativeOperator و SqrtOperator و SquareOperator نیز هر کدام نوعی `UnaryOperator` یا عملگر یگانه هستند.
- کلاس های UnaryOperator و BinaryOperator و NumberExpression هم نوعی `Expression` یا عبارت ریاضی هستند.

برای شروع ابتدا به توضیح کلاس Expression می پردازیم. نحوه کار این کلاس به این صورت است که فایل ورودی را باز کرده و با خواندن خط به خط فایل مشخص می کند که کدام یک از سایرین کلاس ها باید صدا زده شوند.برای تکمیل توضیحات به نمونه کد زیر دقت کنید. .(نمونه کد ۲۴۹)

```
using System;
using System.IO;

namespace OOCalculator
{
    public abstract class Expression
    {
        public abstract double Evaluate();

        public static Expression BuildExpressionTree(StreamReader reader)
        {
            return Expression.GetNextExpression(reader);
        }

        protected static Expression GetNextExpression(TextReader reader)
        {
            string line = reader.ReadLine();
            switch (line)
            {
                case "Add":
                    return new AddOperator(reader);

                case "Subtract":
                    return new SubtractOperator(reader);

                case "Multiply":
                    return new MultiplyOperator(reader);

                case "Negate":
                    return new NegateOperator(reader);

                case "Square":
                    return new SquareOperator(reader);

                case "Divide":
                    return new DivideOperator(reader);

                case "SquareRoot":
                    return new SqrtOperator(reader);

                default:
                    return new NumberExpression(line);

            }
        }
    }
}
```

نمونه کد ۲۴۹: Expression

در ادامه کلاس AddOperator را پیاده سازی می کنیم. با توجه به کانستراکتور این کلاس که ورودی از نوع TextReader دارد،این کلاس می تواند فایل را خط به خط خوانده و عبارت سمت چپ و راست را پیدا کند و سپس حاصل جمع را بدست آورد.

.(نمونه کد ۲۵۰)

```
using System;
using System.IO;

namespace OOCalculator
{
    public class AddOperator : BinaryOperator
    {
        public AddOperator(TextReader reader)
        {
            throw new NotImplementedException();
        }

        public override string OperatorSymbol => throw new NotImplementedException();

        public override double Evaluate() => throw new NotImplementedException();
    }
}
```

نمونه کد ۲۵۰: پیاده سازی AddOperator

نکته: TDD (Test Driven Development) یک روش برای برنامه نویسی است که برنامه نویسی را راحت تر میکند.بدین صورت که شما ابتدا با نوشتن تست خوب انتظار خود را از برنامه مشخص می کنید و سپس سعی می کنید برنامه ای بنویسید که تست مورد نظر کار کند.

حال بر همین اساس تست مناسب را برای این کلاس می نویسیم.

نکته: برای جلوگیری از ایجاد مشکل در خواندن و نوشتن در فایل از Path.GetTempFileName() استفاده می کنیم.این متود فایلی قابل دسترسی به ما می دهد. در ادامه تست مربوط به کلاس را به صورت زیر می نویسیم.

.(نمونه کد ۲۵۱)

```
using OOCalculator;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.IO;

namespace l29cs.tests
{
[TestClass]
    public class OperatorTests
    {
[TestMethod]
        public void AddOperatorTests()
        {
            var tempFile = Path.GetTempFileName();
            File.WriteAllLines(tempFile, new string[]{"3", "4"});
            StreamReader r = new StreamReader(tempFile);
            AddOperator ao = new AddOperator(r);
            double result = ao.Evaluate();
            Assert.AreEqual(7, result);

            string tostrResult = ao.ToString();
            Assert.AreEqual("(3+4)",tostrResult);
        }
     }
}
```

نمونه کد ۲۵۱: Test AddOperator

مراحل بعدی را با دیباگ کردن پیش می بریم.تا در جاهایی که نیاز است کد مربوطه را بنویسیم. برای جلوگیری از Exception تغییرات زیر را در کلاس AddOperator اعمال میکنیم. .(نمونه کد ۲۵۲)

```
using System;
using System.IO;

namespace OOCalculator
{
    public class AddOperator : BinaryOperator
    {
        public AddOperator(TextReader reader)
        {
        }

        public override string OperatorSymbol => "+";
        public override double Evaluate() => 0;
    }
}
```

نمونه کد ۲۵۲: پیاده سازی AddOperator

از آنجایی که کانستراکتور کلاس AddOperator خالی است،کانستراکتور کلاس پدر یعنی BinaryOperator صدا زده می شود پس برای مانع شدن از وقوع Exception برای کلاس پدر نیز تغییرات زیر را اعمال می کنیم.

.(نمونه کد ۲۵۳)

```
using System;
using System.IO;

namespace OOCalculator
{
    public abstract class BinaryOperator: Expression, IOperator
    {
        protected Expression LHS;
        protected Expression RHS;

        public BinaryOperator()
        {

        }

        public BinaryOperator(TextReader reader)
        {
        }

        public abstract string OperatorSymbol { get; }

        public sealed override string ToString() => throw new NotImplementedException();

    }
}
```

نمونه کد ۲۵۳: پیاده سازی BinaryOperator

کلاس AddOperator علاوه بر property های خودش، property های کلاس BinaryOperator که از آن ارث میبرد را نیز دارد(LHS وRHS) که مقدار دهی نشده اند. پس باید مقداردهی شوند و عبارت سمت چپ و راست حساب شود.برای این کار از متودهای کلاس Expression استفاده می‌کنیم.به این صورت : .(نمونه کد ۲۵۴)

```
using System;
using System.IO;

namespace OOCalculator
{
    public class AddOperator : BinaryOperator
    {
        public AddOperator(TextReader reader)
        {
            LHS=Expression.GetNextExpression(reader);
            RHS=Expression.GetNextExpression(reader);
        }

        public override string OperatorSymbol => "+";

        public override double Evaluate() => throw new NotImplementedExcwption();
    }
}
```

نمونه کد ۲۵۴: پیاده سازی AddOperator

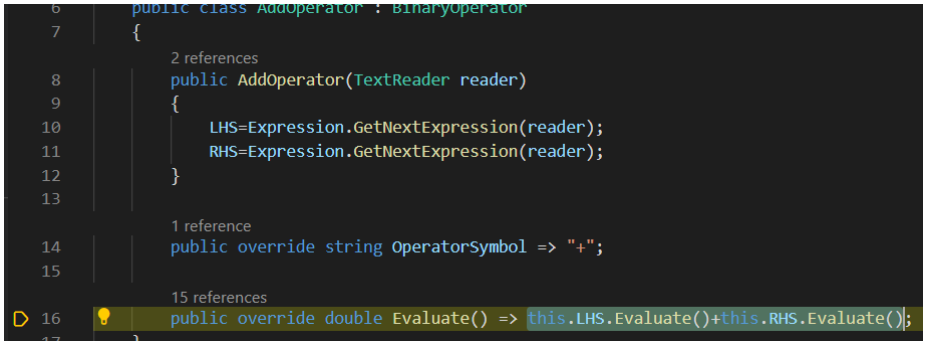
و سپس برای محاسبه مقدار نهایی از تابع Evaluate در کلاس AddOperator که override شده است استفاده می کنیم.به این صورت:

.(نمونه کد ۲۵۵)

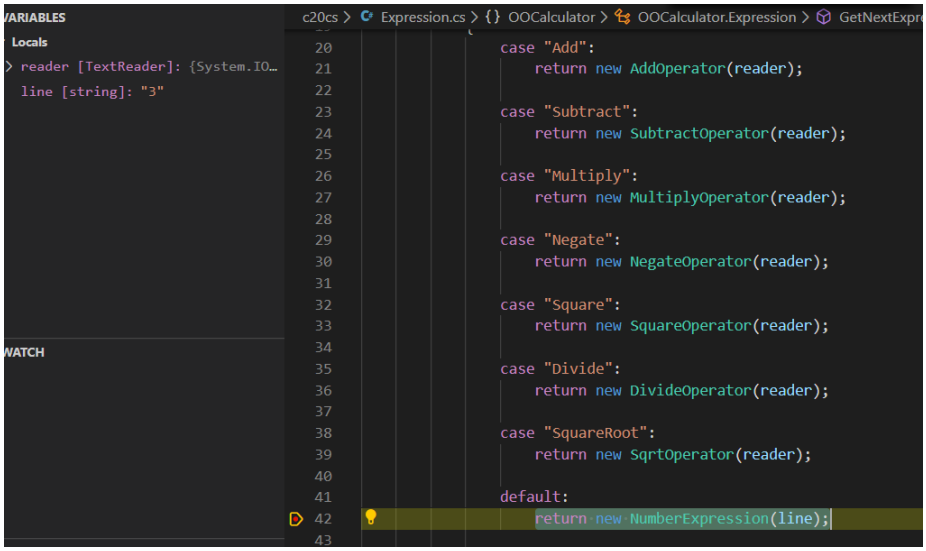
```
using System;
using System.IO;

namespace OOCalculator
{
    public class AddOperator : BinaryOperator
    {
        public AddOperator(TextReader reader)
        {
            LHS=Expression.GetNextExpression(reader);
            RHS=Expression.GetNextExpression(reader);
        }

        public override string OperatorSymbol => "+";

        public override double Evaluate() => this.LHS.Evaluate()+this.RHS.Evaluate();
    }
}
```

نمونه کد ۲۵۵: پیاده سازی AddOperator

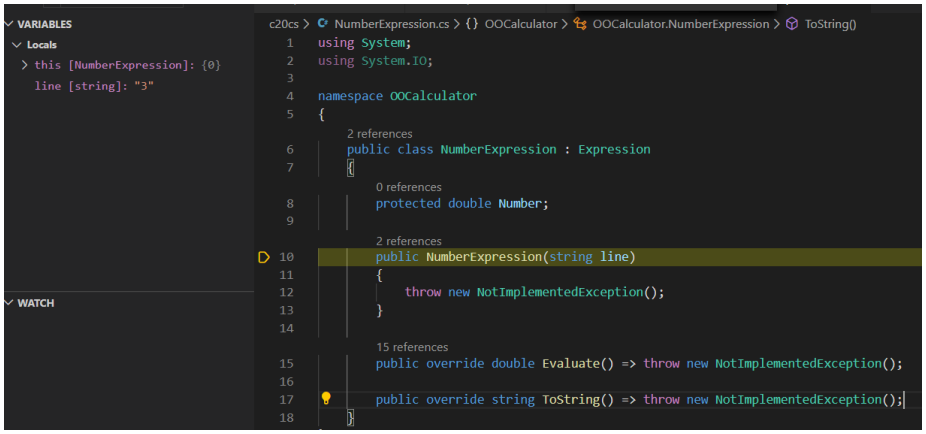
در ادامه مسیر با توجه به تصاویر زیر،نیاز به پیاده سازی کلاس NumberExpression است.



شکل ۳.۲۹: AddOperator

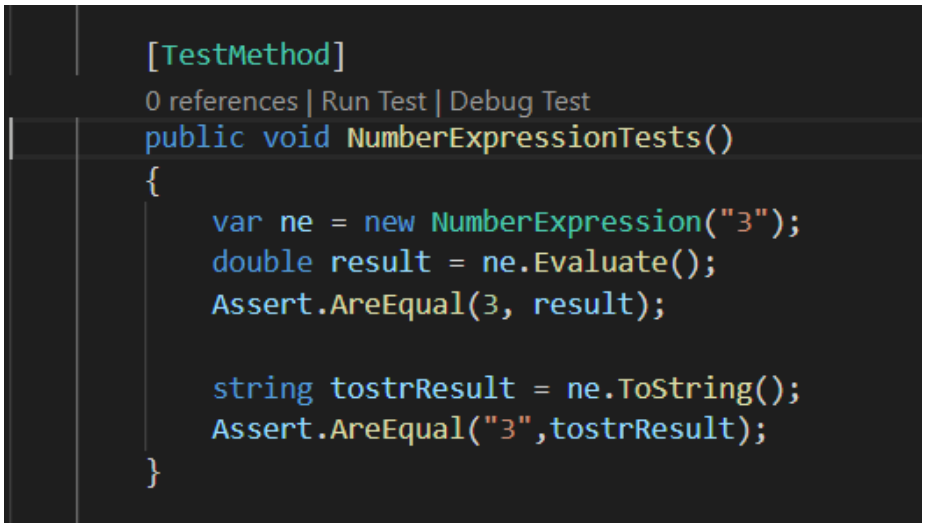


شکل ۴.۲۹: Expression



شکل ۵.۲۹: NumberExpression

برای پیاده این کلاس نیز ابتدا تست مربوطه را به این صورت مینویسیم.



شکل ۶.۲۹: Test NumberExpression

و در نهایت کد ان را به این شکل پیاده سازی می کنیم. .(نمونه کد ۲۵۶)

```
1  using System;
2  using System.IO;
3
4  namespace OOCalculator
5  {
6      public class NumberExpression : Expression
7      {
8          protected double Number;
9
10         public NumberExpression(string line)
11         {
12             Number=double.Parse(line);
13         }
14
15         public override double Evaluate() => this.Number;
16
17         public override string ToString() => this.Number.ToString();
18     }
19 }
```

نمونه کد ۲۵۶: پیاده سازی NumberExpression

جلسه ۳۰

Design Patterns – State Pattern

پارسا عیسی زاده – ۱۳۹۹/۳/۱۸

جزوه جلسه ۳۰ام مورخ ۱۳۹۹/۳/۱۸درس برنامه‌سازی پیشرفته تهیه شده توسط پارسا عیسی زاده . در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته جزوه جلسه ۳۰ ام درس برنامه سازی پیشرفته.

در جلسات قبل همه مفاهیم برنامه نویسی شی گرا تدریس شد ولی هدف ما از جایی به بعد در برنامه نویسی لزوما این نیست که برنامه درست بنویسیم بلکه باید برنامه تمیز بنویسیم . design pattern های زیادی داریم،لیست pattern design ها را اگر search بکنیم روی اینترنت هست و اگر بخواهیم پیاده سازی شده آن ها با سی–شارپ را هم ببینیم میتوانیم در Github جست و جو بکنیم . Factor pattern ، Facade pattern و ... از design pattern ها هستند . در این جلسه قصد داریم State Pattern را یاد بگیریم .

۱.۳۰ کلاس Account

یک کلاس به نام Account برای مدیریت حساب بانکی میسازیم . این کلاس ۴ property و ۵ متد دارد .

```
internal class Account
    {
        public int Balance {get; private set;} = 0;
        public bool isVerified {get; private set;} = false ;
        public bool IsClosed {get; private set;} = false ;
        public bool IsFrozen {get; private set;} = false ;
        public Account(Action onUnFrozen) {}
        public void Deposit(int value) => this.Balance += value ;
        public void Withdraw(int value) => this.Balance -= value ;
        public void HolderVerified() => isVerified = true ;
        public void Close() => this.IsClosed = true ;
        public void Freeze() => this.IsFrozen = true ;

    }
```

نمونه کد ۲۵۷: کلاس Account

متغیر `Balance` نشان دهنده مقدار دارایی حساب است ، متغیر `IsVerified` نشان دهنده این است که کسی به حساب دسترسی دارد صاحب واقعی آن هست یا خیر . متغیر `IsClosed` بیانگر این است که آیا حساب بسته شده یا نه و متغیر `IsFrozen` بیانگر این است که آیا در نتیجه استفاده نشدن بلند مدت بسته شده یا نه .

همانطور که از کد پیداست متد `Withdraw` برای زمانیست که از موجودی حساب کم شده ، متد `Deposit` برای زمانیست که وجهی به حساب واریز شده ، متد `Close` برای وقتیست که کاربر حسابش را میبندد و متد `Freeze` برای زمانیست که در اثر استفاده نکردن بلند مدت حساب خود به خود بسته میشود.

برای وقتی که `Deposit` و `Withdraw` صدا زده شدند ما یک متغیر از جنس Action به نام `OnUnFrozen` تعریف میکنیم و متغیر `IsFrozen` را false میکنیم .مقدار `OnUnFrozen` را از Constructor کلاس میگیریم .

۲.۳۰ Guard Clause

همانطور که از کد پیداست ما تعداد زیادی متغیر از جنس boolean داریم که برای انجام هر کاری به مقادیرشان نیاز داریم (برای هر کاری باید چک کنیم که آیا حساب کاربری بسته شده یا نه ، آیا کسی که به حساب دسترسی دارد صاحب واقعی آن است یا نه و ...). یک راه این است که تعدادی if تو در تو بنویسیم . این کار از زیبایی کدمان کم میکند و از انجایی که خیلی شلوغ میشود کار با کد و خواندن آن را سخت تر میکند .

برای حل چنین مشکلی از guard clause استفاده میکنیم که راه شناخته شده تری است بین برنامه نویس ها . guard clause به این صورت است که در همان ابتدای متد بررسی میکنیم شرطی را ، اگر شرط

مطلوبمان نبود از متد خارج میشویم .برای مثال در متد `Withdraw` :

```
public void Withdraw(int value)
{
    if(!this.IsVerified)
        return ;
    if(!this.IsClosed)
        return ;

    this.Balance -= value ;
    ManageUnFeezing();
}
```

نمونه کد ۲۵۸: متد `Withdraw` نوشته شده با guard clause

یادآوری : تکرار کردن یک کد از گناهان کبیره (!) ایست که یک برنامه نویس میتواند مرتکب شود . ما گفتیم که در متد های `Withdraw` و `Deposit` نیاز است که حساب از حالت بسته در بیاید . برای این کار باید یک کد را در هر دو متد بنویسیم اینجاست که متد `ManageUnFreezing` را مینویسیم و در هر دو متد استفاد میکنیم .

```
private void ManageUnFreezing()
{
    if(!this.Frozen)
        return ;

    this.IsFrozen = false ;
    this.OnUnFrozen() ;
}
```

نمونه کد ۲۵۹: متدی برای حالتی که حساب بسته شده

State Pattern ۳.۳۰

برای استفاده از if تا الان دو تا راه یاد گرفتیم یکی guard clause بود و دیگری همان if و else عادی که از ابتدا بلد بودیم .اما در این بخش ما قصد داریم به کلی if را از کدی که برای کلاسمان نوشته ایم را حذف کنیم .

در ابتدا سعی داریم در متد `ManageUnFreezing` ، if را حذف کنیم . برای این کار به جای متد `ManageUnfreezing` متغیری با همین نام از جنس action تعریف میکنیم . که در صورت بسته بودن با متد `UnFreeze` برابر شود و در غیر این صورت با متد `DoNothing` . با این روش ما شرطی نمیگذاریم بلکه با استفاده از متد ها تنها از حالتی به حالت دیگری میرویم .

```
public action ManageUnFreezing = DoNothing ;
private void UnFreeze()
{
    this.IsFrozen = false ;
    this.OnUnFrozen();
    this.ManageUnFreezing = DoNothing ;
}

private void Freeze()
{
    this.IsFrozen = true ;
    this.ManageUnFreezing = UnFreeze ;
}
```

نمونه کد ۲۶۰: حذف if از بدنه متد `ManageUnFreezing`

حالا که if از بدنه حذف شد دیگر نیازی به متغیر `IsFrozen` نداریم و میتوانیم آن را حذف بکنیم .

دیدیم که برای حذف این if به یک delegate نیاز داریم . از آنجاییکه یک delegate همانند interface با یک متد است و ما نیاز به چند delegate داریم ، یک interface جدید تعریف میکنیم .

```
internal interface IAccountState
{
    IAccountState Withdraw(Action doWithdraw);
    IAccountState Deposit(Action doDeposit);
    IAccountState Freeze();
    IAccountState HolderVerified();
    IAccountState Close();
}
```

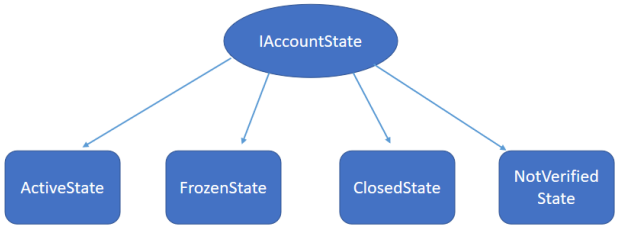
نمونه کد ۲۶۱: `IAccountState`

همانطور که در کد میبینیم خروجی هر متد یک IAccountState است چون که ما بعد از انجام هر عملی وارد حالت جدیدی میشویم .

باید یادآوری کنیم که یکی از اصلی ترین قواعد در برنامه نویسی شی گرا single responsibility principle است که یعنی هر کلاس باید تنها یک وظیفه داشته باشد . اگر دقت کنیم میبینیم که کلاس account در اینجا علاوه بر انجام کارهای مربوط به حساب (واریز و برداشت) ، حالت های مختلف حساب را هم چک میکند .

تا الان از کلمه حالت (state) زیاد استفاده کرده ایم . میدانیم که در هر حالت ، رفتار کلاسمان متفاوت خواهد بود . برای همین از interface تعریف شده استفاده میکنیم . برای اینکه استفاده کنیم باید برای برای

هر state یک کلاس جدید تعریف کنیم که Interface را پیاده سازی کند .



شکل ۱.۳۰: رابطه بین State ها و Interface

حال باید در هر state مشخص کنیم که بعد از ورود به هر متد بسته به state مان چه کاری انجام دهد و سپس وارد چه State ای بشود.به نوعی میتوان state ها را به دومینویی تشبیه کرد که هر state کلاس را به state بعدی میرساند . اگر state امان تغییر نکرد خروجی متد باید this باشد و اگر state تغییر کرد باید شی ای جدید از همان حالت با ورودی `OnUnFrozen` به constructor اش را برگرداند .

نکته : همه state ها به یک `OnUnFrozen` نیاز دارند که به عنوان ورودی در constructor بگیرند .

حال میرویم و ساختار هر یک از state ها را بررسی میکنیم .

Active State ۱.۳.۳۰

Deposit

ابتدا باید متد `OnUnFrozen` صدا زده شود همانطور که در کلاس Account داشتیم .چون در حالت active موجودی حساب میتواند تغییر کند ، پس ابتدا موجودی افزایش می یابد سپس چون حالت بعدی هم دوباره active است همین حالت را باید برگرداند .

Freeze

در این متد تنها کاری که باید انجام شود تغییر حالت به frozen state است .

Withdraw

همانند deposit است تنها با این تفاوت که از موجودی کم میشود .

HolderVerified

چون توی active state هستیم پس هویت تایید شده پس حالت تغییری نمیکند.

Close

تنها کاری که باید در این متد انجام شود این است که حساب به حالت ClosedState برود .

```
internal class ActiveState : IAccountState
{
    private Action OnUnFrozen;

    public ActiveState(Action onUnFrozen)
    {
        this.OnUnFrozen = onUnFrozen;
    }

    public IAccountState Close() => new ClosedState();

    public IAccountState Deposit(Action doDeposit)
    {
        doDeposit();
        return this;
    }

    public IAccountState Freeze() => new FrozenState(OnUnFrozen);

    public IAccountState HolderVerified() => this;

    public IAccountState Withdraw(Action doWithdraw)
    {
        doWithdraw();
        return this;
    }
}
```

نمونه کد ۲۶۲: `ActiveState`

۲.۳.۳۰ Frozen State

Deposit

در اینجا چون با واریزبه حساب ، حساب به نوعی فعال (active) میشود ، در نتیجه ابتدا action ورودی اش را اجرا میکند ، سپس `OnUnFrozen` مربوط به کلاس را و بعد از آن وارد active state میشود .

Freeze

دیدیم که در active state بعد از صدا کردن متد Freeze به State Frozen رفتیم . همین حالتی که الان درآن هستیم در نتیجه در اینجا حالت تغییر نمیکند و همین حالت را به عنوان خروجی میدهیم .

Withdraw

همانند deposit است تنها با این تفاوت که در اینجا از حساب برداشت میشود (این که برداشت میشود یا واریز بستگی به action ای دارد که به عنوان ورودی میگیرد وگرنه syntax مشابه deposit دارد .)

HolderVerified

از آنجاییکه معلوم است حساب freeze شده است یا نه پس هویت تایید شده پس حالت تغییری نمی کند .

Close

تنها کاری که باید در این متد انجام شود این است که حساب به حالت ClosedState برود .

```
internal class FrozenState : IAccountState
{
    private Action OnUnFrozen;
    public FrozenState(Action onUnFrozen)
    {
        this.OnUnFrozen = onUnFrozen;
    }
    public IAccountState Close() => new ClosedState();
    public IAccountState Deposit(Action doDeposit)
    {
        doDeposit();
        this.OnUnFrozen();
        return new ActiveState(OnUnFrozen);
    }
    public IAccountState Freeze() => this;
    public IAccountState HolderVerified() => this;
    public IAccountState Withdraw(Action doWithdraw)
    {
        doWithdraw();
        this.OnUnFrozen();
        return new ActiveState(OnUnFrozen);
    }
}
```

نمونه کد ۲۶۳: `FrozenState`

NotVerified State ۳.۳.۳۰

Deposit

اگر هویت تایید نشده باشد میتوان به حساب وجهی واریز کرد و تنها نمیتوان از آن برداشت کرد . پس action پاس داده شده اجرا میشود ولی چون هنوز هویت تایید نشده پس حالت تغییری نمیکند و متد this را برمیگرداند.

Freeze

هویتی که تایید نشده نمیتواند حسابی را freeze کند (!) در نتیجه این متد کاری انجام نمیدهد و همین حالت را باز میگرداند (چون state تغییری نکرده).

Withdraw

چون هویت تایید نشده در نتیجه نمیتواند از حساب وجهی برداشت کند در نتیجه این متد کاری انجام نمیدهد و از آنجایی که تایید نشده state تغییری نمیکند .

HolderVerified

چون در این حالت هویت تایید میشود، حساب وارد حالت active میشود و `OnUnFrozen` را به عنوان ورودی active state میدهیم .

Close

تنها کاری که باید در این متد انجام شود این است که حساب به حالت ClosedState برود.

```
internal class NotVerfiedState : IAccountState
{
    private Action OnUnFrozen;
    public NotVerfiedState(Action onUnFrozen)
    {
        this.OnUnFrozen = onUnFrozen;
    }
    public IAccountState Close() => new ClosedState();
    public IAccountState Deposit(Action doDeposit)
    {
        doDeposit();
        return this;
    }
    public IAccountState Freeze() => this;
    public IAccountState HolderVerified() => new ActiveState(OnUnFrozen);
    public IAccountState Withdraw(Action doWithdraw) => this;
}
```

نمونه کد ۲۶۴: `NotVerifiedState`

۴.۳.۳۰ Closed State

این حالت از آنجایی که به نوعی بن بست است ، نمی تواند حساب را به حالت دیگری منتقل کند ، در نتیجه هر متد تنها همین حالت (closed) را برمیگرداند و کار دیگری انجام نمی دهد .

```
internal class ClosedState : IAccountState
{
    public IAccountState Close() => this;
    public IAccountState Deposit(Action doDeposit) => this;
    public IAccountState Freeze() => this;
    public IAccountState HolderVerified() => this;
    public IAccountState Withdraw(Action doWithdraw) => this;
}
```

نمونه کد ۲۶۵: `ClosedState`

حال باید کلاس Account را پیاده سازی کنیم . ما در اول جلسه این کلاس را به همراه تعداد زیادی if پیاده سازی کردیم . if هایی برای isVerified و isClosed . هدف ما از ابتدا حذف if بود و اینکه هر کلاس تنها یک مسئولیت داشته باشد . در نتیجه در اینجا تشخیص اینکه حساب در حالت isVerified یا isClosed هست را میگذاریم بر عهده کلاس state . پس تنها متد مورد نظر را برای کلاس state صدا می زنیم .

```
internal class Account
{
    public int Balance {get; private set;} = 0;
    IAccountState State;

    public Account(Action onUnFrozen)
    {
        State = new NotVerfiedState(onUnFrozen);
    }
    public void Deposit(int value)
    {
        this.State = State.Deposit(() => {this.Balance += value;});
    }
    public void Withdraw(int value)
    {
        this.State = this.State.Withdraw(() => {this.Balance -= value;});
    }
    public void HolderVerified()
    {
        this.State = this.State.HolderVerified();
    }
    public void Close()
    {
        this.State = this.State.Close();
    }
    public void Freeze()
    {
        this.State = this.State.Freeze();
    }

}
```

نمونه کد ۲۶۶: `Account`

حال کلاس account تنها مقدار Balance و state را نگه میدارد و بقیه کار ها را بقیه کلاس ها بسته به state حساب انجام میدهند . حال میبینیم که single responsibility principle به خوبی حس میشود . هم اکنون کد تمیز تری داریم که به راحتی مدیریت میشود . این pattern های مختلف برای طراحی برنامه علاوه بر اینکه کاربردی هستند نشان میدهند که ما چقد برنامه نویسی بلدیم و مهم است که pattern های مختلف را یاد بگیریم .

۴.۳۰ ماشین حساب

یک مثال عملی برای state pattern ماشین حساب است .بعضی از state های ماشین حساب کامپیوتر را بررسی میکنیم .

Start state :هنگامیکه عدد روی صفحه ۰ است در state ای هستیم که هر چقدر دکمه ۰ را فشار میدهیم

اتفاقی نمیوفتد .

Accumulate state :اما بعد اگر دکمه عدد دیگری را فشار دهیم حالت عوض شده و در این حالت جدید اگر دکمه ۰ را فشار دهیم عدد نمایشگر تغییر میکند .

Point state :اگر دکمه ممیز فشرده شود برنامه وارد حالت اعشاری میشود که باز هربار که دکمه صفر فشرده شود عدد نمایشگر تغییر میکند و در این حالت جدید اگر دوباره دکمه ممیز را زدیم عدد نمایشگر تغییری نمیکند.

Compute state :اگر دکمه یک عملگر فشرده شود عدد صفحه صفر میشود و باز به start state بازگردانده میشود .

در برنامه این ماشین حساب یک قرارداد IState ای داریم . برای نوشتن آن میبینیم که چه چیزی state برنامه را تغییر میدهد ، و برای هر کدام از آنها یک delegate تعریف میکنیم .

در تمرین شماره ۱۱ ، IState تنها یکبار آن هم توسط کلاس Calculator state پیاده سازی شده (که شبیهش را در مثال حساب بانکی نداشتیم) . در این کلاس برای همه توابع یک حالت پیش فرض در نظر گرفته شده که بعضی null اند و برخی دیگر که میخواهیم برای همه state ها یکسان باشند پیاده سازی شده اند .

برای دیدن design pattern های بیشتر میتوانید به لینک های زیر رجوع کنید :

https: https://refactoring.guru/design-patterns/behavioral-patterns
//www.tutorialspoint.com/design_pattern/state_pattern.htm

همچنین برای دیدن پیاده سازی شده state pattern ها برای سی-شارپ می توانید به لینک زیر رجوع کنید :

https://github.com/RefactoringGuru/design-patterns-csharp