



دانشکده مهندسی کامپیوتر

جزوه درس

برنامه‌سازی پیشرفته

استاد درس: سید صالح اعتمادی*

نیمسال دوم
سال تحصیلی ۹۹-۹۸

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

فهرست مطالب

۸

Functions and Parameters ۲

پریا فضاحت - ۱۳۹۸/۱۱/۱۹

۸ ۱.۲ معرفی زبان‌های Native و Managed

۱۳

Functions ۳

نیوشا بقیعی - ۱۳۹۸/۱۱/۳۰

۱۴ ۱.۳ تعریف Function

۱۵ ۲.۳ انواع Function

۱۵ ۳.۳ انواع ورودی ها

۱۷ ۴.۳ انواع خروجی ها

۱۹

Method/Function Features ۴

محمدمهری جاوید - ۱۳۹۸/۱۱/۲۶

۱۹ ۱.۴ متادولینگ

۲.۴ فرستادن براساس مقدار یا آدرس

۲۷ Pass by reference VS Pass by value

۳۴ : (variadic functions)

۳۹ : (Generic Functions)

۴۴ : (Naming Conventions)

۴۹ Fold Expression :

۵۳ ۷.۴ نحوه صحیح تقسیم کردن دو عدد بر یکدیگر :

۵۶ ۸.۴ تمرین کلاسی :

۶۰

Class and Object ۵

روزیه غزوی - ۱۳۹۸/۱۱/۲۸

۶۱	۱.۵
۶۱	۲.۵
۶۲	۳.۵
۶۳ (Access Modifiers)	۴.۵
۶۴ (Field)	۵.۵
۶۴ (Constructor)	۶.۵
۶۵ (Method)	۷.۵
۶۵ (Property)	۸.۵
۶۶ Auto-implemented	۹.۵
۶۷ Namespace	۱۰.۵

۶۸

۶ آرایه‌ها و کلاس

نیکی نژاکتی - ۱۳۹۸/۱۲/۳

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

۷۶

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

امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

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

۸۶

۸ کار با فایل در سی شارپ

باپک بهکام کیا - ۱۳۹۸/۱۲/۱۰

۸۶	۱.۸ آشنایی با فایل
۸۷	۲.۸ برنامه ثبت نام دانش آموزان
۹۲		۹ شبیه سازی و استاتیک
		محمد جواد مهدی تبار - ۱۳۹۸/۱۲/۱۲
۹۲	۱.۹ اهداف اصلی این جلسه
۹۲	۲.۹ کلمه های کلیدی مهم
۹۹	۳.۹ دستورات مهم فایل
۱۰۰	۴.۹ کد زده شده درون کلاس
۱۰۵		۱۰ Directory/File (دیرکتوری و فایل)
		علی رهنما علمداری - ۱۳۹۸/۱۲/۱۷
۱۰۵	Directoryclass ۱.۱۰
۱۰۷	۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص
۱۱۰	۳.۱۰ منابع
۱۱۱		۱۱ شبیه سازی چیلین وارز
		شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹
۱۱۱	۱.۱۱ حل مسئله
۱۲۳	۲.۱۱ Queue
۱۲۷		۱۲ stacks- objects
		بنشه قلی نژاد - ۱۳۹۸/۱۲/۲۴
۱۲۹	۱.۱۲ استک
۱۳۴	۲.۱۲ objects
۱۴۱		۱۳ Destructor
		یاسمن مدنی - ۱۳۹۹/۱/۱۶
۱۴۱	۱.۱۳ عنوانین کلی جلسه
۱۴۲	۲.۱۳ دیستراکتور و فاینالایزر
۱۴۵	۳.۱۳ Struct

۱۴۷	داده نوع های Reference Type و Value Type ۱۴
	مسعود گلستانه - ۱۳۹۹/۱/۱۸
۱۴۸	۱.۱۴ تفاوت میان داده های value type و reference type در سی شارپ
۱۵۱	۲.۱۴ کپی سطحی (shallow copy)
۱۵۲	۲.۱۴ باکسینگ و آنباکسینگ
۱۵۴	۴.۱۴ Nullable ها در سی شارپ
۱۵۶	Exceptions ۱۵
	یاسمن توکلی - ۱۳۹۹/۱/۲۳
۱۵۸	۱.۱۵ exception چیست؟
۱۵۷	۲.۱۵ رفع exception
۱۵۹	۲.۱۵ نکته ۱
۱۶۱	۴.۱۵ نکته ۲
۱۶۱	۵.۱۵ try/catch vs if/else
۱۶۲	۶.۱۵ throwing using else/if
۱۶۳	۷.۱۵ Call Stack and Re-Throwing Exceptions
۱۶۴	۸.۱۵ throw new exception با تفاوت
۱۶۶	Exceptions & Operators ۱۶
	بیان دیوانی آذر - ۱۳۹۹/۱/۲۵
۱۶۹	۱.۱۶ Exceptions
۱۷۳	۲.۱۶ Indexers
۱۷۵	Operator Overloading ۱۷
	بنیکی مجیدی فرد - ۱۳۹۸/۱/۳۰
۱۷۷	۱.۱۷ Operator Conversion
۱۸۰	۲.۱۷ Pairs Operator
۱۸۴	واسط ها ۱۸
	باوان دیوانی آذر - ۱۳۹۹/۲/۱
۱۸۴	۱.۱۸ چالش ۱
۱۸۹	۲.۱۸ چالش ۲
۱۹۲	۳.۱۸ نکات

۱۹۲ خلاصه بندی ۴.۱۸
۱۹۳ تمرینات اضافی ۵.۱۸
۱۹۴	Interface <code>IEnumerable</code>, <code>IDisposable</code> ۱۹
	آزاده دارابی مقدم - ۱۳۹۹/۲/۶
۱۹۴ Generic Interface ۱.۱۹
۱۹۶ Generic Constraints ۲.۱۹
۱۹۷ <code>IDisposable</code> ۳.۱۹
۱۹۸ <code>StreamReader</code> Class ۴.۱۹
۱۹۸ <code>Stopwatch</code> ۵.۱۹
۱۹۹ <code>IEnumerable</code> ۶.۱۹
۲۰۲	چگونگی کارکرد مموری ۲۰
	سعید شهیب زاده - ۱۳۹۹/۲/۸
۲۰۲ <code>Memorymanager</code> Example ۱.۲۰
۲۱۱	اشاره گر به تابع ۲۱
	مهدیه نادری - ۱۳۹۸/۲/۱۳
۲۱۱ اشاره گر به تابع در زبان پایتون ۱.۲۱
۲۱۲ اشاره گر به تابع در زبان سی شارپ ۲.۲۱
۲۲۱	Closure - Async Pattern ۲۲
	پارمیدا مجتمع صنایع - ۱۳۹۹/۰۲/۲۰
۲۲۲ <code>c++</code> در Closure ۱.۲۲
۲۲۷ <code>c#</code> در Closure ۲.۲۲
۲۲۹ Multi-Threading ۳.۲۲
۲۳۷	Async Pattern و Tasks و Thread ۲۴
	زهرا مومنی نژاد - ۱۳۹۹/۷/۱۸
۲۳۷ <code>Async</code> و <code>Await</code> ۱.۲۴
۲۴۱ Tasks ۲.۲۴
۲۴۲ Thread ۳.۲۴

۲۵۲	LINQ : Language Integrated Query ۲۵ فاطمه میرجلیلی - ۱۳۹۹/۲/۲۷
۲۵۵	۱.۰۵ ا نوع تعریف Tuple
۲۵۷	۲.۰۵ Tuple کردن یک Deconstruct
۲۵۷	۳.۰۵ LINQ
۲۶۲	لينك ۲۶ محمد حسین رجبی - ۱۳۹۹/۳/۱۶
۲۶۲	۱.۰۶ دیزاین پترن (الگوی طراحی) چیست ؟
۲۶۳	۲.۰۶ شکل فایل بدین صورت است :
۲۶۵	۳.۰۶ Ternary operator
۲۶۵	۴.۰۶ Aggregate
۲۶۵	۵.۰۶ GroupBy
۲۶۶	۶.۰۶ شکل فایل بدین صورت است :
۲۶۷	۷.۰۶ Join
۲۶۸	وراثت ۲۷ امیرحسین درخشان - ۱۳۹۹/۳/۳
۲۶۹	۱.۰۷ کاربرد وراثت
۲۷۰	۲.۰۷ ضرورت استفاده از وراثت
۲۷۰	۳.۰۷ توضیح کلی و نحوه استفاده از وراثت
۲۷۲	۴.۰۷ استفاده از Constructor و توابع در کلاس های فرزند
۲۷۴	۵.۰۷ استفاده از protected
۲۷۴	۶.۰۷ استفاده از abstract
۲۷۵	۷.۰۷ Polymorphism
۲۷۷	۸.۰۷ استفاده از Sealed
۲۷۸	Design Patterns - State Pattern ۳۰ پارسا عیسی زاده - ۱۳۹۹/۳/۱۸
۲۷۸	۱.۰۸ کلاس Account
۲۷۹	۲.۰۸ Guard Clause
۲۸۰	۳.۰۸ State Pattern

٤.٣٠ ماشین حساب

جلسه ۲

Functions and Parameters

پریا فصاحت - ۱۳۹۸/۱۱/۱۹

جزوه جلسه ۱۲ مورخ ۱۳۹۸/۱۱/۱۹ درس برنامه‌سازی پیشرفته تهیه شده توسط پریا فصاحت؛ در این جلسه توابع و پارامترها در زبان‌های C++، C# تدریس شدند. امید است جزوی تهیه شده مفید واقع شود.

۱.۲ معرفی زبان‌های Native و Managed

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++، C# نامیده می‌شوند:

کد نوشته شده تنها کدیست که اجرا می‌شود. از قسمت main شروع می‌شود و تنها هر چه در main نوشته شده اجرا می‌شود. و هیچ 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# نامیده می‌شود:

* این زبان‌ها تبدیل به یک زبان میانی می‌شوند. برای مثال در زبان C# تبدیل به ILCode می‌شود. و در زبان Java Byte Code کد میانی است. لازم به ذکر است که کدهای میانی به تنهایی قابل اجرا نیستند؛ و نیاز به یک Run Time Environment نیاز دارند. که برای جاوا DotNet Run Time Environment و برای سی‌شارپ Java Run Time Environment 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 Java, C#، 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.

```

1 int* nums1 = (int*) malloc(sizeof(int) * 5);
2 //Malloc
3
4 int* nums2 = new int[5];
5 //new

```

نمونه کد ۱: تفاوت نوشتاری new و malloc

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:

```

1 public class Program
2 {
3     int[] nums = ReadFromInput();
4     static void Main(string[] args)
5     {
6         SortNumbers();
7     }
8 }

```

نمونه کد ۲: استفاده از متغیر global

intermediate Language*
Garbage Collection†

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

```

1 public class Program
2 {
3     static void Main(string[] args)
4     {
5         int[] nums = ReadFromInput();
6         SortNumbers(nums);
7     }
8 }
```

نمونه کد ۳: استفاده از متغیر local

این نمونه کد برای هر تعدادی 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:

```

1 static void Sort(int[] nums)
2 {
3     for(int i=0; i<nums.Length; i++)
4     {
5         for(int j=i+1; j<nums.Length; j++)
6         {
7             if (nums[i] < nums[j])
8             {
9                 Swap(ref nums[i], ref nums[j]);
10            }
11        }
12    }
13 }
```

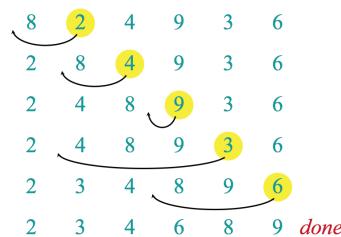
نمونه کد ۴: تابع Sort در زبان C#

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 مستقیماً روی خود متغیر ایجاد می‌شود، نه صرفاً مقدار کپی شده‌ی آن. `[ref]`.

- برای درک بهتر، این روش مرتب سازی 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 یکی‌گر تعویض می‌شود. این روند تا جایی ادامه می‌یابد که تمامی عناصر آرایه باهم مقایسه شده‌باشند و حلقه‌ی اول به پایان برسد. لازم است توجه کنیم که؛ هر بار شمارنده‌ی حلقه‌ی اول یک واحد اضافه می‌شود، حلقه‌ی دوم یک بار کامل اجرا می‌شود.



شكل ۱.۲ : Sort Function Algorithm

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

```

1 void sort(std::vector<int>& nums)
2 {
3     for (size_t i = 0; i < nums.size(); i++)
4     {
5         for (size_t j = i + 1; j < nums.size(); j++)
6         {
7             if (nums[i] < nums[j])
8                 swap(nums[i], nums[j]);
9         }
10    }
11 }
```

نمونه کد ۵: تابع Sort در زبان CPP

در نمونه کد فوقانی `size_t` در واقع برای استفاده از سایز یا شمار[‡] آورده شده است. 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 اندازه 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.

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 `[vector] #include<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 می‌شود. به عنوان مثال برای پخت کیک ابتدا باید به دستورالعمل مراجعه کرد که مراحل را عنوان کرده مخلوط کردن، در قالب ریختن، ... سپس برای هر مرحله توضیحات جداگانه داده می‌شود. در واقع این همان کاری است که function ها انجام می‌دهند؛ آن ها ورودی و خروجی های تعریف شده ای دارند و عملکرد های خاصی را دنبال می‌کنند. اجرا و نوشتן Top-down-approach را در هر مرحله Pseudocode می‌گوییم.

۱.۳ Function تعریف

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

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

نمونه کد ۶: مثالی در زبان

```
1 public void func()
2 {
3     System.Console.WriteLine(world" "Hello);
4 }
```

c#: مثالی در زبان

```
1 #include <iostream>
2 int func()
3 {
4     std::cout << World!" "Hello;
5 }
```

c++: مثالی در زبان

```
1 public class Sample {
2     public void func() {
3         System.out.println(world" "Hello);
4     }
5 }
```

نمونه کد ۹: مثالی در زبان

۲.۳ Function انواع

۴ نوع function قابل تعریف و استفاده کردن است:

- متدهای بدون ورودی و بدون خروجی
- متدهای بدون ورودی و دارای خروجی
- متدهای با ورودی و بدون خروجی
- متدهای 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 object, vector, array, string, list, integer ... 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.

* قابلیت مشابه آرایه دارد با این تفاوت که سایز آنها بصورت dynamically قابل تغییر هم هست. این قابلیت با درج کتابخانه `ash` قابل استفاده است و کاربردهای 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 تغییری روی متغیر مستقیماً 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, java, csharp, cpp ... قابلیت استفاده از این نوع را دارند.

```

1 void swap(int* x, int* y)
2 {
3     int temp;
4     temp = *x; /* x address at value the save */
5     *x = *y; /* x into y put */
6     *y = temp; /* y into temp put */
7 }
8 int a = 1;
9 int b = 2;
10 swap(&a, &b);

```

نمونه کد ۱۰ : مثالی در زبان C

```

1 void swap(int& x, int& y)
2 {
3     int temp;
4     temp = x; /* x address at value the save */
5     x = y; /* x into y put */
6     y = temp; /* y into x put */
7 }
8 int a = 1;
9 int b = 2;
10 swap(a, b);

```

نمونه کد ۱۱ : مثالی در زبان C++

```

1 void swap(ref x, ref y)
2 {
3     int temp = x;
4     x = y;
5     y = temp;
6 }
7 int a = 1;
8 int b = 2;
9 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 دقیقاً نوع متغیر ورودی را مشخص نکنیم می‌توان از معادل‌های کلی استفاده کرد؛

در ... var <= C# ... در ... auto <= C++ ...

: value by pass *

در این حالت مقدار متغیر (یک کپی از متغیر مورد نظر) به تابع فرستاده می‌شود و هر تغییری روی متغیر مستقیماً 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 سنتیت که تابع پس از انجام کار خود پاسخ می‌دهد. مقدار برگشتی می‌تواند هر یک از ۴ نوع متغیر باشد: لیست یا آرایه ، string object integer یا .

تقریباً 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

محمد Mehdi جاوید - ۱۳۹۸/۱۱/۲۶

۱.۴ متدهای اولو دینگ

متدهای اولو دینگ: 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 متدهای اولو دینگ خواهیم داشت.

*** نوع خروجی تابع (ریترن تایپ) تاثیری در متدهای اولو دینگ 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 متدهای اولو دینگ سیگنیچر یک تابع پارامترهای ورودی یک تابع هستند.

قسمت‌هایی از شکل که زیر آن خط قرمز کشیده شده است. پارامترهای تابع هستند.

```
static int add (int x, int y)
```

شکل ۱.۴ : سیگنیچر یک تابع در متاد اولودینگ

۲.۱.۴ متاد اول لو دینگ 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 سیگنچرهای متفاوتی دارند. و همانطورکه می بینید هیچ اوروی 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 parameters
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؟

```

1  using System;
2
3  namespace MethodOverLoading
4  {
5      class Program
6      {
7          static int Multiply (int x, int y, int z, int i, int j) => x * y * z * i * j; function-1 //
8          static int Multiply (int x, int y, int z, int i) => x * y * z * i; function-2 //
9          static int Multiply (int x, int y, int z) => x * y * z; function-3 //
10         static int Multiply (int x, int y) => x * y; function-4 //
11         static void Main(string[] args)
12         {
13             Console.WriteLine(Multiply(1, 2, 3));
14         }
15     }
16 }
```

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 اولودینگ 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 اولودینگ کافی است.

مثال از چند نوع یا تایپ داده‌ای متفاوت :

int - double - string - bool

```
0 references
static string add string name, string lastName) => name + lastName;
0 references
static int add int num1, int num2) => num1 + num2;
0 references
static double add double num1, double num2) => num1 + num2;
```

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

مثال زیر را حل کنید :

- ۱ - کدام تابع صدای خواهد شد؟
- ۲ - خروجی برنامه چه خواهد بود؟

```

1  using System;
2
3  namespace MethodOverLoading
4  {
5      class Program
6      {
7
8          static string add (string name, string lastName) => name + lastName; 1 function //
9          static int add (int num1, int num2) => num1 + num2; 2 function //
10         static double add (double num1, double num2) => num1 + num2; 3 function //
11         static void Main(string[] args)
12         {
13             Console.WriteLine(add("Ali", "Reza"));
14         }
15     }
16 }
```

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

جواب سوال :

- ۱ - تابع شماره یک صدای خواهد شد.
- ۲ - خروجی برنامه AliReza = خواهد بود.

سوال :

چگونه می‌تواند متدهای ارائه دینگ رخ دهد. در صورتی که نوع و تعداد پارامترهای ورودی یکسانی دو تابع داشته باشد؟

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 متاداورلودینگ در بقیه زبان‌ها:

در تمامی زبان‌های برنامه‌نویسی (جاوا - سی‌پلاس‌پلاس - سی‌شارپ) متاداورلودینگ به طور مشابه طبق توضیحات بالا رخ می‌دهد. اما در پایتون اینطور نیست.
علت چیست؟

در پایتون تابع آبجکت (شیء) از کلاس 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. آخرین تابع نوشته شده. یا آخرین آبجکت (شیء) ساخته شده. استفاده خواهد شد. و تعاریف بالا از تابع بدون استفاده خواهد ماند. برای فهم بیشتر این موضوع کد زیر را نگاه کنید.

```

1 def add (x) :
2     return x
3 def add (x, y) :
4     return x + y
5 def add (x, y, z) :
6     return x + y + z
7
8 print(add(1))
9 print(add(1, 2))
10 print(add(1, 2, 3))

```

نمونه کد ۱۵ : متاداورلودینگ 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 تابع را صدا می‌زنیم. می‌توانیم به جای 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 داده‌ای 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 داده‌ای 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.

```

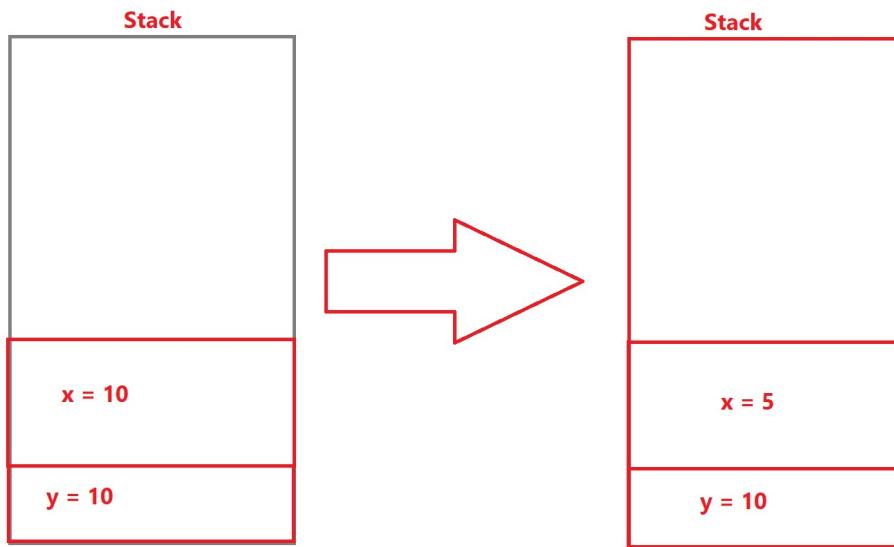
1 static void NewGrade (int x)
2 {
3     x = 5;
4     Console.WriteLine(x);
5 }
6
7 static void Main(string[] args)
8 {
9     int y = 10;
10    NewGrade(y);
11    Console.WriteLine(y);
12 }
```

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 می بینیم. مقدار درون متغیر تغییر نکرده است. هنگامی که ما تابع NewGrade را صدا می زنیم. برنامه مقدار درون متغیر y را نگاه می کند. و آن مقدار را کپی کرده. و در متغیر x که پارامتر ورودی تابع NewGrade است. قرار می دهد. 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 متغیر (pass by value) می گویند.

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 بدھیم. در همه جا اعمال می‌شود. زیرا تنها چیزی را که به ما تابع دیگری می‌دهیم. پوینتر به آدرس هیچ آن است. و اگر ما تغییری در آن آدرس هیچ بدھیم. در همه جا هایی 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 بدھیم. کافی است. که از کلمه کلیدی ref در سیگنیچر تابع و در هنگام دادن آن متغیر به تابع استفاده کنیم.

: (ref keyword) ref

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؟

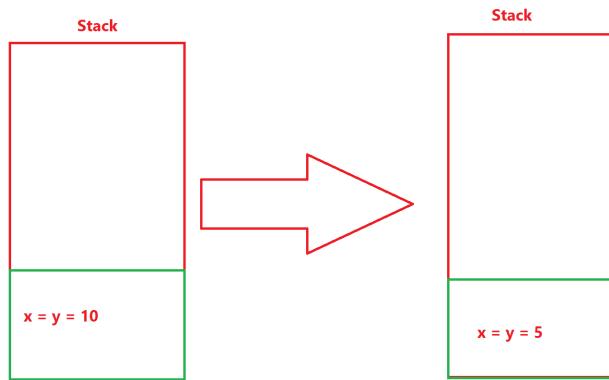
```

1 static void NewGrade (ref int x)
2 {
3     x = 5;
4     Console.WriteLine(x);
5 }
6
7 static void Main(string[] args)
8 {
9     int y = 10;
10    NewGrade(ref y);
11    Console.WriteLine(y);
12 }
```

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 داده‌ای 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. حتماً 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 صدای 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.

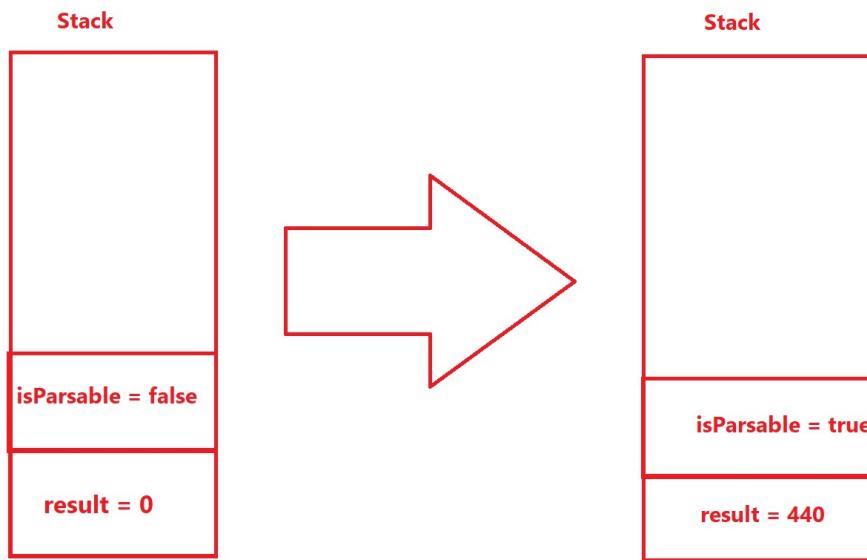
```

1 static void Main(string[] args)
2 {
3     int result;
4     bool isParsable = int.TryParse("440", out result);
5     Console.WriteLine(result + "\t" + isParsable);
6 }
```

نمونه کد : ۱۸

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 `440` 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 :



out keyword : ۷.۴

جرا مقدار اولیه متغیر result مساوی با صفر است؟

دادهایی 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 درتابع دوم که صدا زده شده. متغیر مقدار بگیرد. اما در استفاده از کلمه کلیدی ref حتما باید که متغیر ما درتابع اولیه مقدار دهی اولیه شده باشد. تا بتوان از آن استفاده کرد.

نکته : آیا می‌توان از کلمه کلیدی ref برای نوع داده‌ای از نوع رفونس نیز استفاده کرد؟

بله اما بودن یا نبودن آن تاثیری نخواهد داشت.

۳.۴ توابع با پارامترهای ورودی متغیر (variadic functions)

C# : ۱.۳.۴

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

برای ساخت تابع 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.

```

1 public static int Sum (params int[] numbers)
2 {
3     int result = 0;
4     foreach (int number in numbers)
5         result += number;
6     return result;
7 }
8 static void Main(string[] args)
9 {
10    Console.WriteLine(Sum(1, 2, 3, 4, 5, 6));
11    int[] numbers = {3, 4, 5, 6};
12    Console.WriteLine(Sum(numbers));
13 }
```

نمونه کد ۱۹ : variadic functions

خروجی برنامه بالا ابتدا عدد ۲۱ و سپس عدد ۱۸ خواهد بود.
همانطور که می‌بینید تعداد ورودی‌های 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.

```

1  public static int HowManyMatches (params int[] numbers, int matchNumber)
2  {
3      int totalMatch = 0;
4      foreach (int number in numbers)
5          if (number == matchNumber)
6              totalMatch++;
7      return totalMatch;
8  }
9  static void Main(string[] args)
10 {
11     Console.WriteLine(HowManyMatches(1, 2, 3, 4, 5, 2, 3, 1, 2));
12 }
```

نمونه کد : ۲۰

مشکل برنامه بالا در کجاست؟

همانطور که در بالاتر گفته شد. اگر قرار است. که توابع تعداد پارامترهای ورودی متفاوتی داشته باشند. الزاماً باید به عنوان آخرین پارامتر به تابع داده شوند.
پس کد زیر نحوه صحیح کد بالا را نمایش می‌دهد.

```

1  public static int HowManyMatches (int matchNumber, params int[] numbers)
2  {
3      int totalMatch = 0;
4      foreach (int number in numbers)
5          if (number == matchNumber)
6              totalMatch++;
7      return totalMatch;
8  }
9  static void Main(string[] args)
10 {
11     int[] numbers = {1, 2, 3, 4, 5, 2, 3, 1,};
12     int match = 2;
13
14     Console.WriteLine(HowManyMatches(match, numbers));
15     Console.WriteLine(HowManyMatches(2, 1, 2, 3, 4, 5, 2, 3, 1));
16 }
```

نمونه کد : ۲۱

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

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.

- پارامتر ورودی که چند مقدار می‌گیرد. از نوع داده‌ای توپل (tuple) خواهد بود.
- در پایتون می‌توانیم. که علاوه بر اینکه پارامترهای ورودیتابع 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.

```

1 languages_and_their_grades = ("Python", "Java", "C++", "C#", 10, 9, 8, 7)
2 item1 = languages_and_their_grades[0]
3 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.

```

1 languages_and_their_grades = { "Python" : 100 ,
2                               "C++" : 99 ,
3                               "C#" : 80 ,
4                               "Java" : 90 }
5 grade1 = languages_and_their_grades["Python"]
6 print(grade1) #100
```

نمونه کد ۲۳ : Python Dictionary

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

```

1 def sum_of_numbers (*numbers):
2     result = 0
3     for number in numbers :
4         result += number
5     return result
6
7 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.

numbers = (1, 2, 3, 4, 5, 6)

```

1 def print_max_grade_with_course (**languages_with_grades):
2     max_grade = 0
3     course_with_max_grade = None
4
5     for course, grade in languages_with_grades.items() :
6         if grade > max_grade :
7             max_grade = grade
8             course_with_max_grade = course
9
10    print(course_with_max_grade)
11    print(max_grade)
12
13 print_max_grade_with_course( Python = 100 ,
14                             Cpp = 99 ,
15                             JAVA = 95 ,
16                             Cs = 90 )

```

نمونه کد ۲۵ : Python Variable-length KeywordArguments

خروجی برنامه بالا و 100 خواهد بود.

برای تفهیم بیشتر سوال بالا ما دیشکتری به نام languages_with_grades داریم. که کلیدهای آن نام دروس و مقدارهای آن نمرات آن ها 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:

```

1 public static int Sum (int... numbers) {
2     int result = 0;
3     for (int number : numbers)
4         result += number;
5     return result;
6 }
7 public static void main(String[] args) {
8     int[] numbers = {1, 2, 3, 4, 5};
9     System.out.println(Sum(1, 2, 3, 4, 5));
10    System.out.println(Sum(numbers));
11 }
```

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.

تابع عمومی :

```

1  public static void print <_Type> (params _Type[] collection)
2  {
3      foreach (_Type element in collection)
4          Console.WriteLine(element);
5  }
6  static void Main(string[] args)
7  {
8      print<string>("Python", "C++", "C#", "Java");
9      print("Python", "C++", "C#", "Java");
10     print<object>(Reza" "Ali, 22, true, 'M');
11 }
```

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.

```

1 static void Swap <_Type> (ref _Type element1, ref _Type element2)
2 {
3     _Type hold = element1;
4     element1 = element2;
5     element2 = hold;
6 }
7 static void Main(string[] args)
8 {
9     int num1 = 1, num2 = 2;
10    Swap(ref num1, ref num2);
11    Console.WriteLine(num1);
12    Console.WriteLine(num2);
13
14    double dnum1 = 5.1, dnum2 = 5.3;
15    Swap(ref dnum1, ref dnum2);
16    Console.WriteLine(dnum1);
17    Console.WriteLine(dnum2);
18 }
```

نمونه کد ۲۸ Generic Functions :

خروجی برنامه بالا چه خواهد بود؟

جواب :

- 2
- 1
- 3.5
- 1.5

: (Generic classes and interfaces) کلاس‌ها یا اینترفیس‌های عمومی

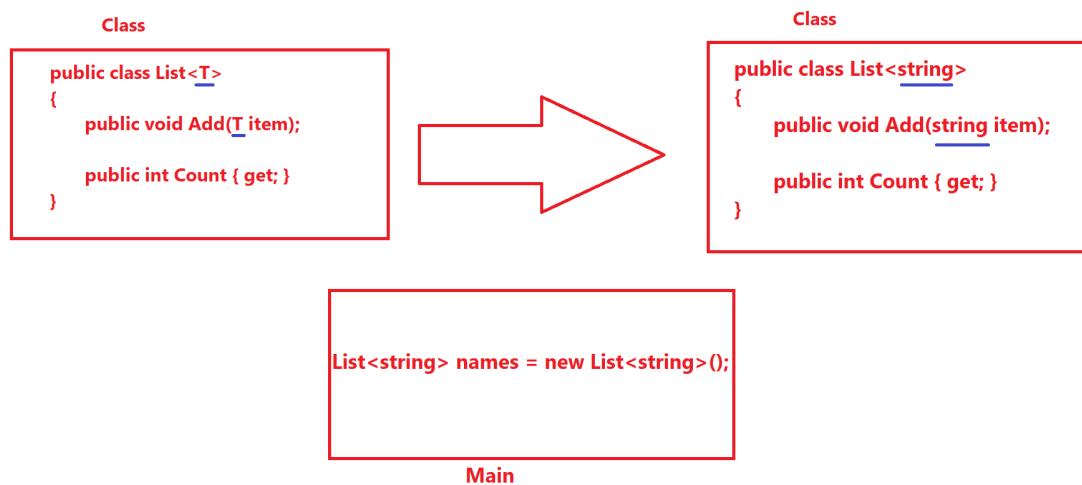
مثال خوبی که از کلاس‌های عمومی وجود دارد. لیست است. که در System.Collections.Generic قرار دارد.

```

1  public class List<T>
2  {
3      public void Add(T item);
4      public int Count { get; }
5  }
6  static void Main(string[] args)
7  {
8      List<string> names = new List<string>();
9      List<int> grades = new List<int>();
10 }
```

نمونه کد : ۲۹

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



generic class : ۸.۴

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

```
¹ public interface IEnumerable<_Type>
```

نمونه کد : ۳۰

: 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 تفہیم بیشتر مطالب نگاه کنید.

```
¹ public static <_Type> void print (_Type... collection) {
²   for (_Type element : collection)
³     System.out.println(element);
⁴ }
⁵ public static void main(String[] args) {
⁶   print("Reza", true, 123, 'M');
⁷   print(140, 70, 2);
⁸ }
```

نمونه کد : ۳۱

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 نام‌گذاری (Naming Conventions)

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

: Famous Naming Conventions

Pascal Casing : هرگاه 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 : هرگاه 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

هرگاه که نام یک معرف یا معین‌کننده هويت (Identifier) به اين صورت نوشته شود که کلمه اول نام متغير بيانگر نوع داده‌اي آن باشد. و ادامه کلمات نام متغير به صورت پاسکال كيس نوشته شوند.
به مثال‌های زير برای تفهيم بيشتر مطالب نگاه کنيد :

- 1- strFirstName (string)
- 2- strLastName (string)
- 3- iStudentId (integer)
- 4- bNameOfVariable (boolean)

هرگاه که نام یک معرف یا معین‌کننده هويت (Identifier) همگي حروف آن به صورت بزرگ نوشته شوند. Upper-case یا به طور ديگر تمامي حروف با Caps Lock نوشته شده باشند. طوري که All Caps نام بگيرند.

به مثال‌های زير برای تفهيم بيشتر مطالب نگاه کنيد :

- 1- FIRSTNAME
- 2- LASTNAME
- 3- STUDENTID
- 4- NAMEOFVARIABLE

هرگاه که نام یک معرف یا معین‌کننده هويت (Identifier) بين کلمات آن از under-line استفاده شود. و حروف کلمات به صورت کوچک نوشته شده باشند.

به مثال‌های زير برای تفهيم بيشتر مطالب نگاه کنيد :

- 1- first_name
- 2- last_name
- 3- student_id
- 4- name_of_variable

: Csharp

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

به مثال زیر برای تفہیم بیشتر مطالب نگاه کنید:

```

1 public class ClientActivity
2 {
3     public void ClearStatistics()
4     {
5         .....
6     }
7 }
```

نمونه کد :۳۲ Method and Classes Naming convention :

: در زبان سی‌شارپ 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 تفہیم بیشتر مطالب نگاه کنید:

```

1 public void Func (int studentId)
2 {
3     string name = "Ali";
4 }
```

نمونه کد :۳۳ Method Arguments and Local 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 تفہیم بیشتر مطالب نگاه کنید :

```

1 public static const Id = 98521000;
```

نمونه کد :۳۴ Constants or Readonly Variables

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

```

1 public interface IComparable
2 {
3 }
```

نمونه کد ۳۵ : Interface

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

```

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

Curly brackets : ۳۶

: در زبان سی‌شارپ، در کلاس‌ها بهتر است. که نام فیلدها را به صورت کمل کیس و نام پر اپرتی‌ها(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 تفہیم 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 }
```

Fields and Properties : ۳۷

: Python

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

```
1 class Student:
```

نمونه کد : ۳۸

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.

```
1 def add_two_numbers (number1, number2):
2     return number1 + number2
```

نمونه کد : ۳۹

: Java

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

```
1 class ImageSprite;
2 interface Sport
```

نمونه کد : ۴۰

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

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

نمونه کد : ۴۱

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

```
1 int studentId;
```

نمونه کد : ۴۲

Fold Expression : ۶.۴

از بی‌سادی‌ام (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 متده 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 می‌توانید به جای کلمه typename از کلمه class نیز استفاده کنید.
 به مثال زیر نگاه کنید:

```

1 template <typename T>
2 auto add (T num1, T num2)
3 {
4     return num1 + num2;
5 }
6 int main(int argc, char const *argv[])
7 {
8     cout << add(1, 2) << endl;
9     cout << add(1.1, 6.1) << endl;
10    cout << add(true, true) << endl;
11 }
```

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 داده‌ای ورودی آن تعداد زیادی ورودی بتواند بگیرد؟

```

1 template <typename ..._Types>
2 auto sum (_Types ...numbers)
3 {
4     return (numbers + ...);
5 }
6 int main(int argc, char const *argv[])
7 {
8     cout << sum(1, 2, 67.5, true, 0001.0) << endl;
9 }
```

نمونه کد ۴۴ Fold Expression :

خروجی برنامه بالا 71.5001 است.

توضیح : پارامتر پک بالا به صورت زیر باز می‌شوند.

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

برای تفهیم بیشتر به مثال زیر نیز نگاه کنید :

```

1 template <typename T>
2 void print(T arg)
3 {
4     cout << arg << endl;
5 }
6 template <typename T, typename ..._Types>
7 void print(T first, _Types ...args)
8 {
9     cout << first << endl;
10    print(args...);
11 }
12 int main(int argc, char const *argv[])
13 {
14     print("Mahdi", 17, 023.0, true, 'J');
15 }
```

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 هر بار 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 انتهای 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. قطعاً با ارور مواجه می‌شویم.

در زیر راهی برای ذخیره ورودی‌های تابع در یک ساختمان داده را بررسی می‌کنیم.
*** از آنجا که ساختمان داده فقط یک نوع می‌تواند داشته باشد. پس حتماً 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.

```

1 template <typename T, typename... _Types>
2 T Sum(_Types ...args)
3 {
4     vector<T> numbers = {args ...};
5     T result = 0.0;
6     for(T number : numbers)
7         result += number;
8     return result;
9 }
10 int main(int argc, char const *argv[])
11 {
12     cout << Sum<int>(1, 3, 10) << endl;
13     cout << Sum<double>(1.1, 2.001, 0.3) << endl;
14 }
```

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. الزاماً 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. الزاماً بخش اعشاری جدا خواهد شد.
برای تفہیم بیشتر مطالب به مثال‌های زیر نگاه کنید :

```

1 int main(int argc, char const *argv[])
2 {
3     int num1 = 7;
4     int num2 = 5;
5     cout << num1/num2 << endl;
6     cout << (double)num1/num2 << endl;
7     cout << (float)num1/num2 << endl; ^I
8 }
```

نمونه کد ۴۷ : Division

خروجی برنامه بالا به ترتیب :

1	1.4
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 تفہیم بیشتر به مثال زیر نگاه کنید:

```

1 int main(int argc, char const *argv[])
2 {
3     int num1 = -7;
4     int num2 = 5;
5     cout << num1/num2 << endl;
6     cout << (float)num1/num2 << endl;
7 }
```

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.

-1

خروجی برنامه بالا :

-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.

```

1 num1 = int(2)
2 num2 = int(3)
3 print(type(num1))
4 print(type(num2))
5
6 result = num1/num2
7 print(type(result))
8 print(result)
9 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

```

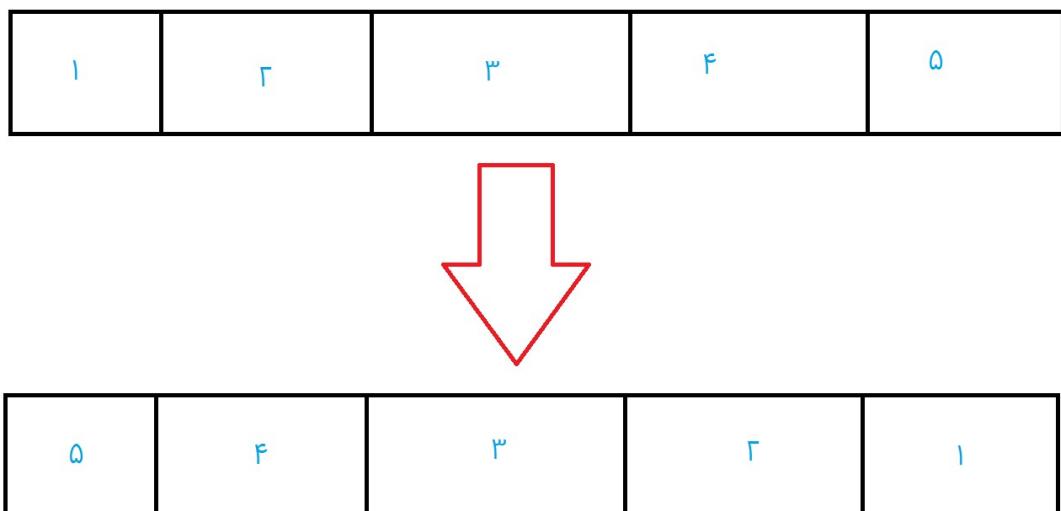
۸.۴ تمرین کلاسی :

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.

Algorithm ۱۸.۴

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

در اینجا ما عضو اول را پنجم و عضوم دوم را با چهارم جابجا می‌کنیم.

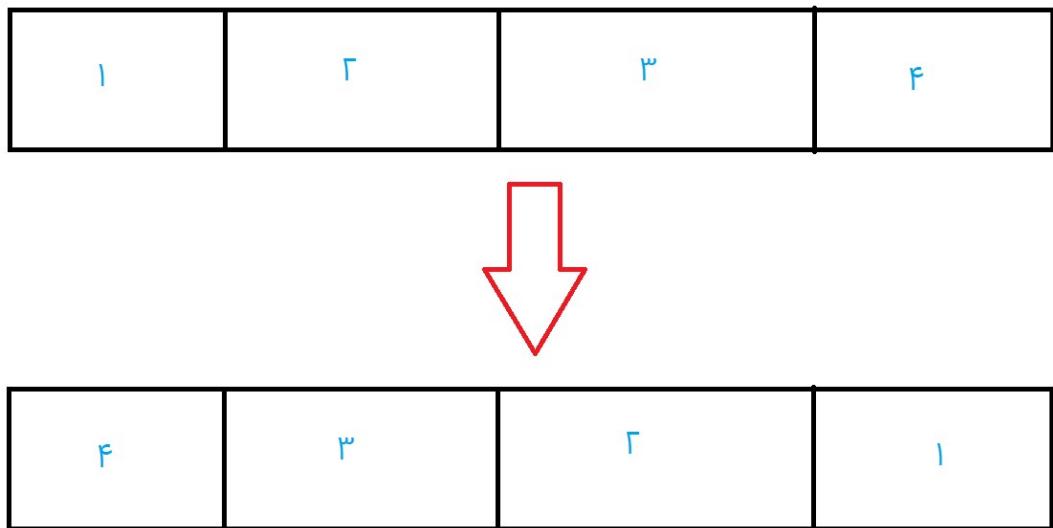


نتیجه برعکس شده

شكل 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.

شكل 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 قبلیش گرد کنیم. تعداد بارهای جابجا کردن لازم را به ما خواهد داد.

Python ۲.۸.۴

```

1 def reverse(l1):
2     for idx in range(int(len(l1)/2)):
3         l1[idx], l1[len(l1)-idx-1] = l1[len(l1)-idx-1], l1[idx]
4
5 def output_reverse(*args):
6     args = list(args)
7     Reverse(args)
8     print(args)

```

نمونه کد : ۵۰ Reverse and Print

۴ باعث می شود. 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 ورودی های تابع output_reverse را به لیست تبدیل کردیم؟ زیرا tuple ها غیرقابل تغییر هستند.

: C# ۳.۸.۴

```

1 static void Reverse<_Type>(_Type[] list)
2 {
3     for (int i=0; i<(list.Count()/2); i++)
4         (list[i], list[list.Count()-1-i]) = (list[list.Count()-1-i], list[i]);
5 }
6 static void OutputReverse<_Type>(params _Type[] args)
7 {
8     Reverse(args);
9     System.Console.WriteLine(string.Join("", args));
10}

```

نمونه کد : ۵۱ Reverse and Print

توضیحات :

اندازه آرایه را برگردانده که خروجی آن عددی از نوع عدد صحیح است. و با تقسیم بر ۲ اعداد مورد نظر را به ما می دهد.

: C++ ۴.۸.۴

```

1  template<typename _Type>
2  void Reverse(vector<_Type>& v)
3  {
4      for(int i=0; i<v.size()/2; i++)
5      {
6          auto hold = v[i];
7          v[i] = v[v.size()-i-1];
8          v[v.size()-i-1] = hold;
9      }
10 }
11
12 template<typename _T, typename... _Type>
13 void OutputReverse(_Type... args)
14 {
15     vector<_T> v1 = {args ...};
16     Reverse(v1);
17     for(string n : v1)
18         cout << n << " ";
19     cout << endl;
20 }
```

نمونہ کد : ۵۲ Reverse and Print

جلسه ۵

Class and Object

روزبه غزوی - ۱۳۹۸/۱۱/۲۸

یک کلاس مانند نقشه‌ای کامل از یک شی مشخص است. در جهان واقعی هر شی ای بی 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 ای بی از یک اتومبیل را ساخته است. با این اوصاف اتومبیل های فراری ، لامبورگینی و کادیلاک همگی شی ای 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 `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` از کلمات رزرو شده `ی` سطوح دسترسی استفاده شده است. و چون در این مورد از کلمه `ی` استفاده شده، هر کسی میتواند شی ای از این کلاس را ایجاد کند. به دنبال کلمه `ی` `class` نام دلخواه کلاس (Circle) قرار گرفته است. باقی مانده `ی` تعریف یک کلاس بدنی `ی` آن است که داده ها و رفتار های کلاس در آن تعریف میشود. فیلد ها، صفات، متدها و رویدادها در مجموع اعضای کلاس را تشکیل میدهند.

۲.۵ ایجاد یک شیء از کلاس

اگرچه یک شیء (`object`) و کلاس در موقعيت به عنوان جایگزینی برای هم دیگر استفاده میشوند، در واقع آنها دو چیز متفاوت هستند. یک کلاس نوع یک شیء را مشخص میکند. گاهی اوقات از شیء به عنوان نمونه ای از یک کلاس یاد میشود. اشیا میتوانند با استفاده از کلمه `ی` رزرو شده `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 `object1` با مقداردهی به یک شیء از نوع `Circle` ارجاع پیدا کرده است.

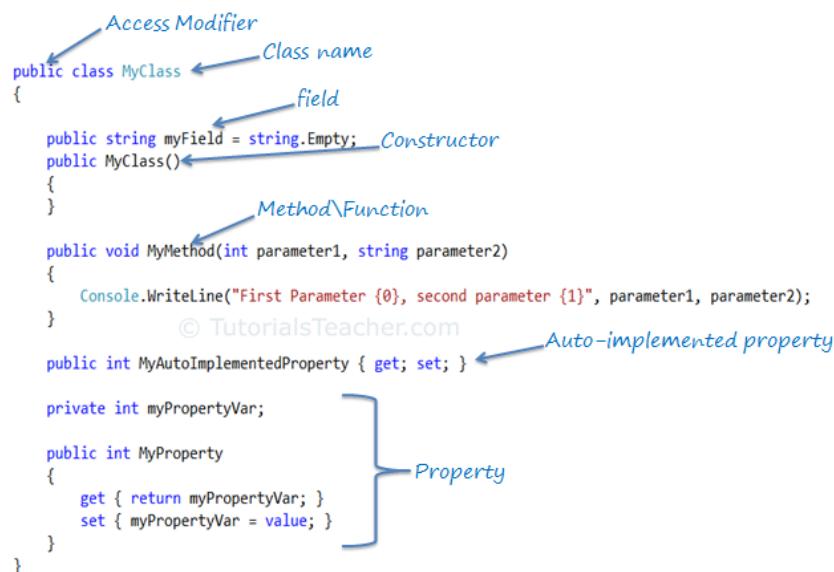
٣.٥ بررسی یک نمونه مثال از کلاس

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

```

1  public class MyClass
2  {
3      public string myField = string.Empty;
4
5      public MyClass()
6      {
7      }
8
9      public void MyMethod(int parameter1, string parameter2)
10     {
11         Console.WriteLine(" First Parameter {0} , Second Parameter {1} ",
12             parameter1, parameter2);
13     }
14
15     public int MyAutoImplementedProperty { get; set; }
16
17     private int myPropertyVar;
18
19     public int MyProperty
20     {
21         get { return myPropertyVar; }
22         set { myPropertyVar = value; }
23     }
24 }
```

نمونه کد ٥٥: 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 رزو شده ای هستند که بر روی اعلان یک کلاس ، متاد، صفت ، فیلد و دیگر اعضای یک کلاس میتوانند اعمال شوند.

کلمات رزو 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.

عملکرد	کلمه های کلیدی
کلمه های public به هر قسمت از برنامه در همان اسمبلی و یا اسمبلی دیگر اجازه میدهد که به نوع و اعضای آن دسترسی پیدا کند.	public
کلمه های 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.	private
کلمه های internal به هر قسمت از برنامه در همان اسمبلی اجازه میدهد که به نوع و اعضای آن دسترسی پیدا کند	internal
کلمه های 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.	protected

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. عموماً فیلد ها در کلاس دارای سطح دسترسی private (فقط قابل دسترسی در محدوده 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 همانم با نام کلاس باشند تعریف میشوند :

```

1 class MyClass
2 {
3     public MyClass()
4     {
5     }
6 }
7

```

نمونه کد ۵۶: مثالی از کانسٹراکتور بدون پارامتر

```

1 class Product
2 {
3     public ID Id;
4     public string Name;
5     public int Price;
6     public double Rate;
7     public Product(ID id , string name, int price, double rate)
8     {
9
10         this.Id=id;
11         this.Name=name;
12         this.Price=price;
13         this.Rate=rate;
14     }
15 }

```

نمونه کد ۵۷: مثالی از کانسٹراکتور پارامتر دار

در بدنه سازنده قصد داریم مقدار متغیری که از پارامتر ورودی سازنده دریافت کرده ایم درون متغیر نمونه کلاس برویزیم! 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 `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`، هرگاه 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.

عموماً صفات در زبان سی شارپ برای سطح دسترسی `public` (قابل دسترسی در خارج از محدوده 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 کلمه‌ی رزرو شده‌ی `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 قسمت‌های `get` و `set` برای یک صفت به کار برد.

```

1  private int myPropertyVar;
2
3  public int MyProperty
4  {
5      get {
6          return myPropertyVar / 2;
7      }
8
9      set {
10
11         if (value > 100)
12             myPropertyVar = 100;
13         else
14             myPropertyVar = value;
15     }
16 }
```

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 ساده‌تر 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:

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

دقت داشته باشید که هیچ فیلدی برای این صفت تعریف نشده است. یک فیلد به صورت ضمنی بعداً 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 `namespace` تعریف شود :

```

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

نمونه کد ۶۱: نمونه ای از فضای نام

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

جلسه ۶

آرایه‌ها و کلاس

نیکی نژاکتی - ۱۳۹۸/۱۲/۳

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

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 جهت static و از جهت 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، عضو ۲۱ 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 ۵ 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 ۲۱ امین عضو به آن اضافه می‌شود، یک vector جدید به اندازه ۲۰×۲ ساخته می‌شود که ۲۰ عضو vector قبلی را کپی کرده و در آن می‌ریزد، vector قبلی را پاک کرده و عضو ۲۱ ام را به این vector جدید اضافه می‌کند.

متغیر vGrades روی stack قرار دارد ولی مقادیر آن در Heap ذخیره شده است. در نتیجه برای استفاده از آن در متدها باید از 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 می‌کنیم:

```
1 Void Print(const std::vector<int> &vGrades)
2 {
3     for(int n:vGrades)
4         std::cout<<n;
5 }
```

نمونه کد ۶۲: تابع چاپ همه اعضای یک vector

۲.۶ آرایه‌ها در Java

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

Dynamic and Static Array •

```
int count=5;
```

```
int[] gradeList = new int[count];
```

آرایه‌ی بالا معادل `int* c++` 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 آن ۳ است:

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

در نتیجه مقدار `a` برابر $4/8$ 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 $6/0$ 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];
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 کلاس‌های کوچک‌تر مانند این مثال، تعریف و پیاده‌سازی کلاس هر دو در یک فایل با پسوند .h انجام شده است. نام فایل‌ها باید با نام کلاس یکی باشد.

برای تعریف کلاس مانند زیر ابتدا class را نوشته و سپس نام آن را می‌نویسیم. کدهای مربوط به کلاس داخل {} نوشته می‌شود و در آخر آن؛ قرار می‌گیرد. در C++ کدها به صورت default به حالت قرار دارند و بیرون از کلاس نمی‌توان به آن‌ها دسترسی داشت. برای قابل دسترس بودن کدها باید در قسمت public نوشته شود:

```
class Course
{
};
```

در کلاس متغیر‌های کلاس و constructor آن را تعریف می‌کنیم:

```
1 #include<string>
2 #include "Instructor.hpp"
3
4 using namespace std;
5
6
7 class Course
8 {
9     string m_Title;
10    string m_InstructorName;
11    string m_InstructorDegree
12    int m_Credits;
13
14 public:
15     Course(string title, string instName, string instDegree, int credits)
16         : m_Title(title)
17             , m_InstructorName(instName)
18                 , m_InstructorDegree(instDegree)
19                     , m_Credits(credits)
20     {}
21 };
```

نمونه کد ۶۳: کلاس در C++

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

یک pointer است، تعریف کرد:

```

1 Course(string title, string instName, string instDegree, int credits)
2 {
3     this->m_Title = title;
4     this->m_InstructornName = instName;
5     this->m_InstructorDegree = instDegree;
6     this->m_Credits = credits;
7 }
```

برای دسترسی به کلاس در فایل های دیگر باید از "#include"ClassName استفاده کرد.

کلاس می‌تواند دارای یک object از یک کلاس دیگر باشد:

```

1 #include <string>
2 using namespace std;
3
4 class Instructor
5 {
6     string m_Name;
7     string m_Degree;
8     double m_Rating;
9
10 public:
11     Instructor(string name, string degree, double rating)
12         : m_Name(name)
13         , m_Degree(degree)
14         , m_Rating(rating)
15     {}
16 };
```

متدهای کلاس مانند دیگر متدها تعریف می‌شوند:

```

1     string GetInfo()
2     {
3         return m_Name + " " + m_Degree;
4     }
```

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

```

1 #include<string>
2 #include "Instructor.hpp"
3
4 using namespace std;
5
6
7 class Course
8 {
9     string m_Title;
10    Instructor m_Instructor;
11    int m_Credits;
12
13 public:
14     Course(string title, string instName, string instDegree, int credits)
15         : m_Title(title)
16             , m_Instructor(instName, instDegree, 5.3)
17             , m_Credits(credits)
18     {}
19
20     string GetCourseInfo()
21     {
22         string result;
23         result += m_Title;
24         result += "\n";
25         result += m_Instructor.GetInfo();
26         return result;
27     }
28 };

```

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:

```

1     string GetCourseInfo()
2     {
3         string result;
4         result += m_Title;
5         result += "\n";
6         result += m_Instructor.GetInfo();
7         return result;
8     }

```

جلسه ۷

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

امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

جزوه جلسه ۱۷م مورخ ۱۳۹۸/۱۲/۰۵ درس برنامه‌سازی پیشرفته تهیه شده توسط امیرحسین سماوات. در جهت
مستند کردن مطالب درس برنامه‌سازی پیشرفته

Value Type و Reference Type ۱.۷

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

Value Type •

Reference Type •

Value Type ۲.۷

به نوعی از داده Value Type گفته میشود که یک مقدار را در فضای حافظه 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.

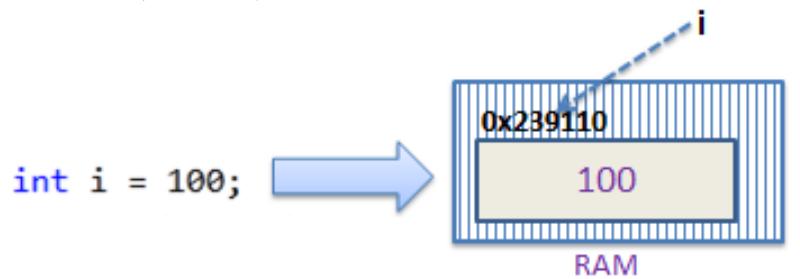
نکته : تمام Type Value ها از فضای نام System.ValueType استفاده میکنند که آن فضای نام هم در فضای نام System.Object قرار دارد.

برای مثال متغیری از نوع int را در نظر بگیرید :

```
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 ذخیره سازی مقدار ۱۰۰ را در حافظه به آدرس (۰x۲۳۹۱۱۰) برای متغیر "i" نشان میدهد



همه‌ی داده‌نوع‌هایی که در زیر آورده شده است از نوع value type هستند :

double	decimal	char	byte	bool
sbyte	long	int	float	enum
ushort	ulong	uint	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 می‌کنید ، 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 نحوه‌ی عملکرد روش ارسال با مقدار را نشان میدهد :

```

1 static void ValueChange(int x)
2 {
3     x = 200;
4
5     System.Console.WriteLine(x);
6 }
7
8 static void Main(string[] args)
9 {
10    int i = 300;
11
12    System.Console.WriteLine(i);
13
14    ValueChange(i);
15
16    System.Console.WriteLine(i);
17 }
```

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 اولیه‌ی متغیر "i" ندارد. با چاپ مقدار

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

```

1 100
2 200
3 100

```

Reference Type ۴.۷

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 قرار گرفته 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.

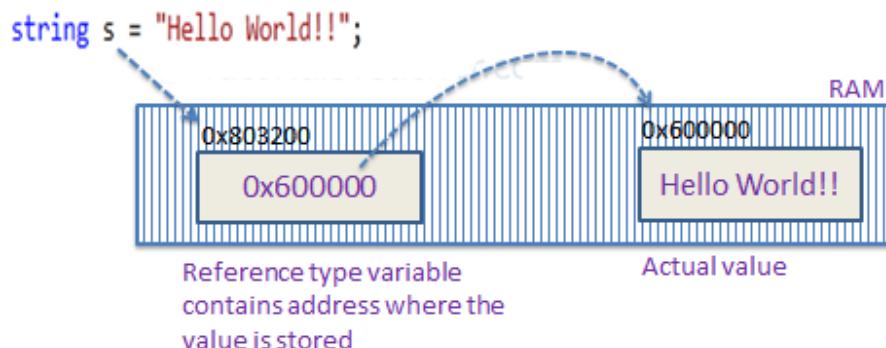
برای این حالت یک متغیر رشته ای را میتوان مثال برد :

```

1 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 ۰x۶۰۰۰۰۰۰ است که آدرس خانه ای از حافظه است که مقدار اصلی یعنی !! Hello World در آن قرار گرفته است.

داده نوع های زیر همگی Reference Type هستند :

String •

• تمام آرایه ، حتی اگر مقادیر آنها از نوع Value Type باشد

Classes •

Delegates •

ارسال با ارجاع Pass by Reference ۵.۷

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

```

1 static void ChangeReferenceType(Student std2)
2 {
3     std2.StudentName = "Steve";
4 }
5
6 static void Main(string[] args)
7 {
8     Student std1 = new Student();
9     std1.StudentName = "Bill";
10
11     ChangeReferenceType(std1);
12
13     Console.WriteLine(std1.StudentName);
14 }
```

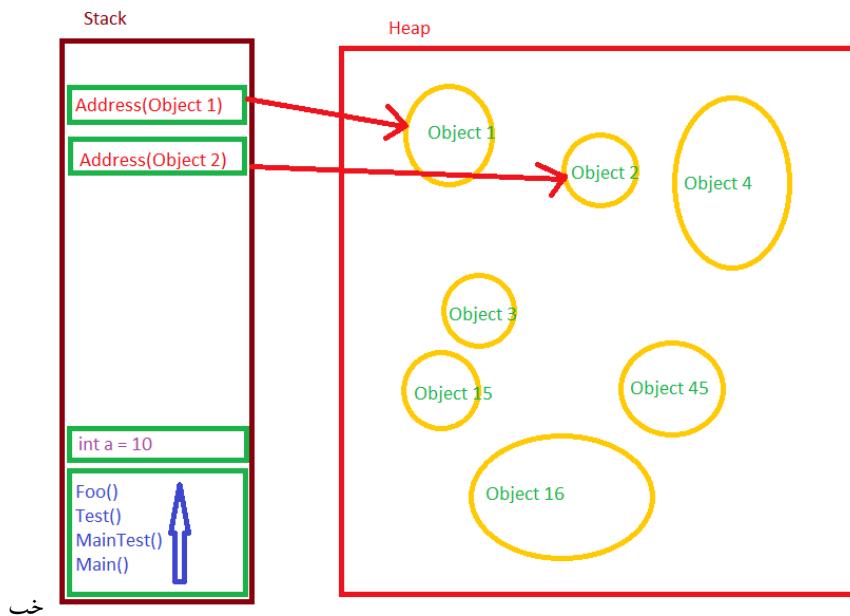
در نمونه مثال فوق ، از آنجایی که Student یک کلاس (class) است هنگامی که شی ایی از کلاس Student به نام std۱ را به عنوان آرگومان به متده ChangeReferenceType() ارسال می کنیم ، آن چیزی که در عمل ارسال میشود آدرس حافظه ای شی std۱ است. بنابراین وقتی متده ChangeReferenceType() فیلد StudentName را تغییر میدهد ، 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 ای "steve" است

Heap و Stack ۶.۷

وقتی که یک متده را صدای زنیم، پارامترها و متغیرهای محلی آن، نیاز به حافظه دارند و این حافظه همیشه از Stack تأمین می‌شود. سپس وقتی کار به متده تمام شد (یا به خاطر Exception) این حافظه به Stack به صورت خودکار بازگردانده می‌شود. وقتی یک نمونه‌ای از یک کلاس را ایجاد می‌کنیم (که از کلمه new کردن استفاده می‌کنیم) برای این نیز یک حافظه نیاز است که از Heap استفاده می‌شود. وقتی کار تمام می‌شود به Heap بازگردانده نمی‌شود.

```

1 public void Main()
2 {
3     Stack // 
4     int x = 2;
5     Heap // 
6     MyClass ob = new MyClass();
7 }
```



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 ۱۶.۷

```

1 #include <iostream>
2 using namespace std;
3 void func(int a)
4 {
5     a++;
6 }
7 int main()
8 {
9     int a=2;
10    func(a);
11    cout<<a;
12 }
```

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

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

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

```

1 #include <iostream>
2 using namespace std;
3 void func(int *a)
4 {
5     (*a)++;
6 }
7 int main()
8 {
9     int a=2;
10    func(&a);
11    cout<<a;
12 }
```

اینجا 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
```

يعني ادرس b رو بزار در a

```
int &b=a
```

يعني با رفنس اينجا كار داريم.

پس مثال بالا رو ميشه به اين شكل هم نوشته بهتر هم هست به همين شكل نوشته بشه چون اشتباها ناخواسته رو کم مي کنه .

```

1 #include <iostream>
2 using namespace std;
3 void func(int& a)
4 {
5     a++;
6 }
7 int main()
8 {
9     int a=2;
10    func(a);
11    cout<<a;
12 }
```

حالا سوالی که پيش مياد اينه که چرا Call کنيم اصلا مگه نميشه کد رو به اين شكل هم نوشت ؟ چرا ولی مشکلش اينه که از نظر سرعت اجرا و مصرفه حافظه اصلا بهينه نيست چون موقع ارسال متغير به تابع يك بار متغير کپي ميشه و همين طور موقع return دن متغير باز يك بار ديگه مقدار برگشتی کپي ميشه تو a اين کپي شدن ممکنه برای int زياد فرقی نکنه ولی برای يك class که مثلا ۱۰۰ تا فيلد داره خيلي به چشم مياد . از اون طرف موقع ارسال پaramter به متغير و زمانی که برنامه داخل تابع هستش ۲ تا کپي از متغير تو حافظه داريم که اينم يعني مصرف حافظه ۲ برابر بيشتر از اونی که نيازه !

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 اشتباہ 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 کرد رو میده .

Java in Class ۷.۷

مفهوم کلاس:

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

شکل کلی کلاس:

```

1 Public class Classname{
2
3     protected int id;
4     public String text;
5     private double spt;
6
7
8     public Classname(int id, String text) {
9         this.id = id;
10        this.text = text;
11    }
12
13     public String functionName(){
14         String data = AP" "Learning;
15         return data;
16     }
17     public class Main {
18         public static void main(String[] args) {
19             Classname c = new Classname(4, "AP");
20             c.functionName();
21         }
22     }
23 }
```

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

مرسی که برای خواندن این جزو و 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 می‌شویم (نمونه کد ۶۴).

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         string stdid = Console.ReadLine();
6         File.WriteAllText("stdid.txt", stdid + "\n")
7     }
8 }
```

نمونه کد ۶۴: مثالی برای `File.WriteAllText`

(نمونه کد ۶۵)

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         string fileName = "stdid.txt";
6         string content = File.ReadAllText(fileName);
7         System.Console.WriteLine(content);
8     }
9 }
```

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 می‌کنیم. (نمونه کد ۶۶).

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

نمونه کد ۶۶: انتخاب اسم فایل

۲.۲.۸ پیاده‌سازی اولیه

حال برنامه‌ای می‌نویسیم که بتواند ورودی ار نوع `string` بگیرد که این ورودی یکی از `add`, `list`, `find` باشد که هر کدام متدى را صدا بزند

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

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         if (args.Length != 1 )
6         {
7             Usage();
8             return;
9         }
10
11         if (args[0] == "add")
12             AddStudent();
13         else if (args[0] == "list" )
14             PrintStudents();
15         else if (args[0] == "find")
16             FindStudent(args);
17         else
18             Usage();
19     }
20 }
```

نمونه کد ۶۷: پیاده سازی add, list, find

(نمونه کد ۶۸)

```

1 private static void Usage()
2 {
3     System.Console.WriteLine(" students. register to used be can program This ");
4     System.Console.WriteLine(" follows: as is syntax usage The ");
5     System.Console.WriteLine(" [searchstring] add|list|find cs.exe ");
6 }
```

نمونه کد ۶۸: پیاده سازی متدها

AddStudent() ۳.۲.۸

پیاده سازی متدهای AddStudent(). این متدهای زمانی صدازده می شود که کلمه add به عنوان ورودی وارد شود.

(نمونه کد ۶۹)

```

1  private static void AddStudent()
2  {
3      Console.Write(" " "Id?");
4      string id = Console.ReadLine();
5      Console.Write(" " "Name?");
6      string name = Console.ReadLine();
7      File.AppendAllText(StorageFileName, " {id} , {name} \n ");
8  }

```

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

PrintStudents() ۴.۲.۸

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

نمونه کد ۷۰.

```

1  private static void PrintStudents()
2  {
3      var lines = File.ReadAllLines(StorageFileName);
4      foreach(var line in lines)
5          System.Console.WriteLine(line);
6  }

```

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.

```

1  class Program
2  {
3      static void Main(string[] args)
4      {
5          if (args.Length != 1 && args.Length != 2)
6          {
7              Usage();
8              return;
9          }
10
11         if (args[0] == "add")
12             AddStudent();
13         else if (args[0] == "list")
14             PrintStudents();
15         else if (args[0] == "find")
16             FindStudent(args);
17         else
18             Usage();
19     }
20 }
```

نمونه کد ۷۱

FindStudent() ۵.۲.۸در ادامه متد `FindStudent()` را پیاده سازی می کنیم

.(۷۲ نمونه کد)

```

1  private static bool FindStudent(string[] args)
2  {
3      if (args.Length != 2)
4          return false;
5
6      string searchKey = args[1].ToLower();
7
8      string[] lines = File.ReadAllLines(StorageFileName);
9      bool found = false;
10     foreach(string line in lines)
11     {
12         string[] tokens = line.ToLower().Split(',');
13         string id = tokens[0];
14         string name = tokens[1];
15         if (name == searchKey)
16         {
17             Console.WriteLine($" {id} : {name} ");
18             found = true;
19         }
20     }
21
22     if (!found)
23         Console.WriteLine(" Found! Not ");
24
25     return true;
26 }
```

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

در کد بالا سطرهای فایل را داخل آرایه‌ای می‌ریزیم. در این صورت آرایه‌ای 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 می‌کنیم سپس اگر اسمی با اسمی که کاربر وارد کرده بود یکی باشد آن اسم و شماره دانشجویی متناظرش در خروجی چاپ می‌شوند 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 `string` با کمک `ToLower()` حروف بزرگ هر دو آن‌ها را به حروف کوچک تبدیل می‌کنیم

جلسه ۹

شی گرایی و استاتیک*

محمدجواد مهدی تبار - 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 متدها 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 •

args ۱.۲.۹

مخفف کلمه arguments می باشد که آرگمن های خط فرمان^۸ را نشان می دهد. که در واقع پارامتر متد اصلی^۹ 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^{۱۰} است . بعد از build کردن برنامه و قبل از اجرا شدن برنامه می توان به عنوان پارامتر به متد اصلی داد. در واقع تنها متدهای که در طول برنامه اجرا می شود متد اصلی است.

نحوه‌ی استفاده از args

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

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

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

برای واضح تر شدن مطلب می توانید از مستندات استفاده کنید .

microsoft doc 1 •

command line[§]
Main[¶]
string array[¤]

microsoft doc 2 •

dotnet perl •

statics ۲.۲.۹

به معنی این است که به `type` متعلق است و نه به یک شی خاص . در `csharp` `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 متدها و متغیرهای آن کلاس باید از دستور، `<class-name.variable>` or `<class-name.method>` استفاده کنیم . در واقع با عملگر `(.)` این کار ممکن است .
اگر از فیلد `static` برای کلاس استفاده کنیم تمام اعضای آن کلاس هم باید `static` باشند .

```

1 static class Example
2 {
3     static int Id;
4     static void Main(string[] args)
5     {
6         Example ex = new Example();
7         above line is incorrect , static class cannot be instantiated //
8         int X = Example.Id;
9     }
10 }
```

نمونه کد ۷۴: کلاس استاتیک

```

1 public class Example
2 {
3     public static int X ;
4     public int Y ;
5     method static // 
6     public static void print()
7     {
8         Console.WriteLine( Example.X );
9     }
10 }
11 public class Program
12 {
13     static void Main(string[] args)
14     {
15         Example.X = 5 ;
16         Example.print();
17     }
18 }
19 
```

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.

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

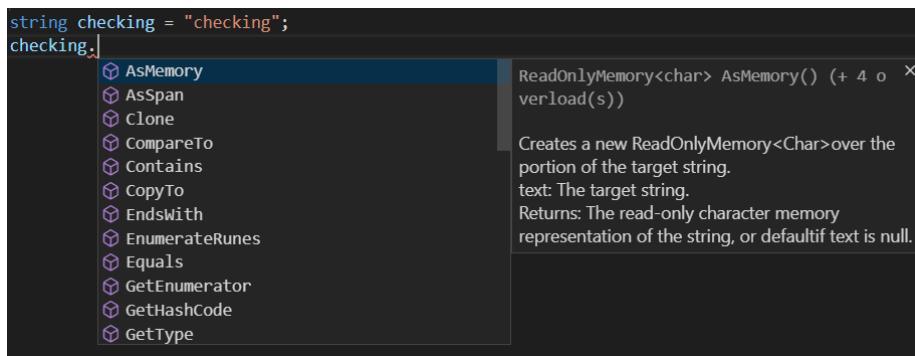
static and non-static method

static method

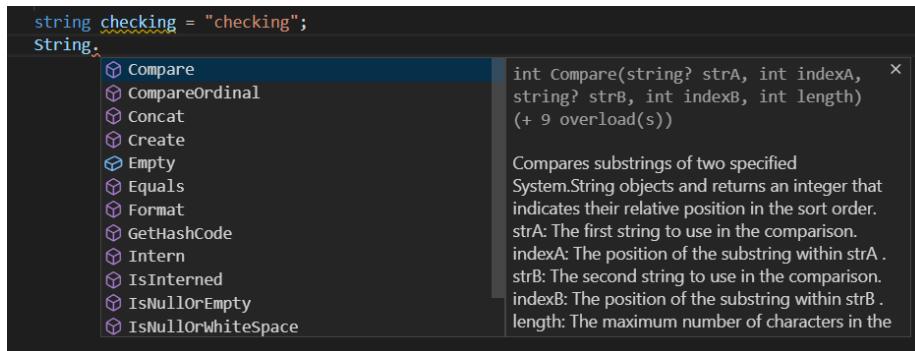
در کل هر وقت از دستور `<class-name.> auto complete` استفاده کنیم پیشنهاد هایی که از `auto complete` می‌آید همه‌ی آنها متد های `static` هستند.

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` می‌آیند `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](#) •



public or private ۳.۲.۹

`public` به معنی قابل دسترس یودن 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 استفاده کرد.

موقعی که باید از `public` استفاده کرد

از `public` موقعی استفاده می‌کنیم که از همه جای برنامه بخواهیم به آن `variable` یا `method` دسترسی داشته باشیم. 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 تغیید دهد.

موقعی 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 `variable` یا `method` 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 و کارایی آن وبسایت‌های زیر مفید هستند `public or private`

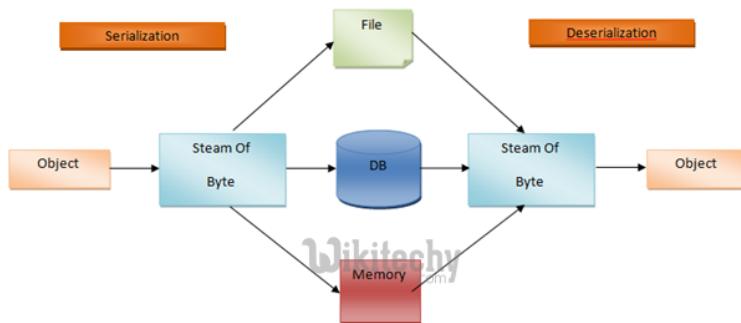
[microsoft doc 1](#) •

[microsoft doc 2](#) •

[dotnetperl](#) •

serialize and deserialize ۴.۲.۹

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` معمولاً شی های یک کلاس رو به صورت رشته `*:` در یک فایل می‌ریزند. برای `deserialize` معمولاً هر خط فایل تشکیل دهنده یک شی است و با زدن حلقه `for` روی هر خط آن می‌توان شی ها را بدهست آورد. و معمولاً آنها را در آرایه یا لیستی از نوع آن شی می‌ریزند.



شكل ۳.۹ : serialization

```

1 class Student
2 {
3     private int Id;
4     private string Name;
5     public string Serialize()
6     {
7         return this.Id + " " + this.Name;
8     }
9     public void AddToFile()
10    {
11        File.AppendAllText(string path , this.Serialize());
12    }
13 }

```

نمونه کد ۷۶ : serialize

string**

```

1  private List<Student> Students;
2  private void deserialize(string file)
3  {
4      string[] lines = File.ReadAllLines(file);
5      foreach(string line in lines)
6      {
7          Student s = Student.Parse(line);
8          Students.Add(s);
9      }
10 }
11 public static Student Parse(string line)
12 {
13     string[] tokens = line.Split(',');
14     int id = int.Parse(tokens[0]);
15     string name = tokens[1];
16     return new Student(id, name);
17 }
18 }
```

نمونه کد :۷۷

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، کارایی آن وسایت های زیر مفید هستند.

[microsoft doc](#) •[csharpcorner](#) •

۳.۹ دستورات مهم فایل

:`File.ReadAllText(string path)` محتويات درون فایل را می‌خواند و خروجی آن از نوع رشته است.

:`File.ReadAllLines(string path)` محتويات درون فایل را می‌خواند و خروجی آن از نوع آرایه ای از رشته است که اندیس i ام آن خط z 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 ۰ ام آن خط نام 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 ۰ ام آن خط نام فایل را تشکیل دهد و اگر فایل مورد نظر موجود نباشد آن را ایجاد می‌کند.

۴.۹ کد زده شده درون کلاس

```

1  private static void Usage()
2  {
3      Console.WriteLine(" this program can be used to reigster students ");
4      Console.WriteLine(" the usage syntax is as follows: ");
5      Console.WriteLine(" cs.exe find|list|add [seachstring] ");
6 }
```

نمونه کد ۷۸: نحوه‌ی 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

`find [wanted]` ، `list` ، `add` ، دانشجویی را یا ثبت نام بکند یا لیست دانشجو‌ها را بگیرد و یا دانشجوی خاصی را پیدا کند. پس طبیعتاً 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^{§§}

```

1 public class Student
2 {
3     private int Id;
4     public string Name;
5     public Student(int id, string name)
6     {
7         this.Id = id;
8         this.Name = name;
9     }
10 }
```

نمونه کد :۷۹ Student member variables and constructor

```

1 public class Course
2 {
3     private string Name;
4     private int Code;
5     private string StorageFileName;
6     private List<Student> Students;
7     public Course(string name, int code, string storageFileName)
8     {
9         this.Name = name;
10        this.Code = code;
11        this.StorageFileName = storageFileName;
12        this.Students = new List<Student>();
13        LoadFile(storageFileName);
14    }
15    private void LoadFile(string file)
16    {
17        string[] lines = File.ReadAllLines(file);
18        foreach(string line in lines)
19        {
20            Student s = Student.Parse(line);
21            this.RegisterStudent(s);
22        }
23    }
24    public void RegisterStudent(Student s)
25    {
26        Students.Add(s);
27    }
28 }
```

نمونه کد :۸۰ Course member variables and constructor

در نمونه کد **۸۰** هر بار که یک شی جدید ساخته می‌شود در سازنده **۲۹ آن متده LoadFile** صدا زده می‌شود تا شی‌های ساخته شده در فایل در لیست دانشجوها ریخته شود. که همان مفهوم **deserialize** که در نمونه کد **۷۷** به آن اشاره شده است.

Constructor^{۳۰}

دستور add

```

1 Program class //
2 Course math = new Course(name: "Math", code: 101, "Students.csv");
3 if (args[0] == "add")
4 {
5     Student s = Student.GetStudentFromConsole();
6     math.RegisterStudent(s);
7     math.StoreInFile();
8 }
9 Student class //
10 public static Student GetStudentFromConsole()
11 {
12     System.Console.Write(" Id? ");
13     int id = int.Parse(Console.ReadLine());
14     System.Console.Write(" Name? ");
15     string name = Console.ReadLine();
16     return new Student(id, name);
17 }
18 Course class //
19 internal void StoreInFile()
20 {
21     List<string> lines = new List<string>();
22     foreach(Student s in this.Students)
23     {
24         lines.Add(s.Serialize());
25     }
26     File.WriteAllLines(StorageFileName, lines);
27 }
```

نمونه کد ۸۱ add student

در نمونه کد ۸۱ دستور `add` دانشجو را از ورودی می‌گیرد که متده 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**

```

1 Program class //
2 else if (args[0] == "find" && args.Length == 2)
3 {
4     string searchKey = args[1];
5     string result = "Found" "Not";
6     Student s = math.FindStudent(searchKey);
7     if (null != s)
8         result = s.Serialize();
9     Console.WriteLine(result);
10 }
11 Course class //
12 public Student FindStudent(string searchKey)
13 {
14     Student foundStudent = null;
15     searchKey = searchKey.ToLower();
16     foreach(Student s in Students)
17     {
18         if (s.Name.ToLower() == searchKey)
19         {
20             foundStudent = s;
21         }
22     }
23     return foundStudent;
24 }
```

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**

```

1 Program class //
2 else if (args[0] == "list" && args.Length == 1)
3     math.PrintStudents();
4 Course class //
5 internal void PrintStudents()
6 {
7     foreach(Student s in this.Students)
8     {
9         Console.WriteLine(s.serialize());
10    }
11 }
```

نمونه کد :۸۳ list student

دستور **list** تمام دانشجو های آن فایل را چاپ می‌کند.

جلسه ۹. شبگرایی و استاتیک

۱۰۴

و بسایت های GeeksforGeeks [dotnetperls] dotnet perl [csharp] microsoft doc
کمک شایانی در یادگیری csharp [GeeksforGeeks] می کنند.

جلسه ۱۰

دیرکتوری و فایل (Directory/File)

علی رهنما علمداری - ۱۳۹۸/۱۲/۱۷

جزوه جلسه ۱۰ ام مورخ ۱۳۹۸/۱۲/۱۷ درس برنامه‌سازی پیشرفته تهیه شده توسط علی رهنما علمداری. در زبان سی شارپ می‌توان به وسیله کلاس Directory با پوشش‌ها کارکرد همانند کلاس File کلاس، یکسری متدهای static دارد که به ما اجازه انجام عملیات‌های مختلف بر روی پوشش‌ها را می‌دهند.

Directoryclass ۱.۱۰

۱.۱.۱۰ ایجاد پوشش

به وسیله متدهای CreateDirectory نمونه کد ۸۴ می‌توان پوشش جدید در مسیر مشخص شده ایجاد کرد:

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

نمونه کد ۸۴: ساخت پوشش در سی‌شارپ

این دستور از بالاترین سطح شروع به ایجاد پوشش می‌کند. برای مثال در کد بالا در صورت عدم وجود پوشش TestFolder این پوشش ایجاد شده و سپس پوشش Test داخل ایجاد می‌شود.

۲.۱.۱۰ برسی وجود یک پوشه

بوسیله متده است `Exists` نمونه کد ۸۵ می توان برسی کرد که یک پوشه وجود دارد یا خیر:

```

1  if (!Directory.Exists("D:\\\\Test"))
2  {
3      Directory.CreateDirectory("D:\\\\Test");
4 }
```

نمونه کد ۸۵: برسی وجود یک پوشه در سی شارپ

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:

```

1  var files = Directory.GetFiles("D:\\MyFolder");
2
3  foreach (var file in files)
4  {
5      Console.WriteLine(files);
6 }
```

نمونه کد ۸۶: فایل های یک پوشه در سی شارپ

۴.۱.۱۰ بدست آوردن زیر پوشه های داخل یک پوشه

بوسیله دستور `GetDirectories` نمونه کد ۸۷ می توان لیست پوشه های داخل یک پوشه را بدست آورد:

```

1  var subDirectories = Directory.GetDirectories("D:\\MyFolder");
2
3  foreach (var directory in subDirectories)
4  {
5      Console.WriteLine(directory);
6 }
```

نمونه کد ۸۷: زیر پوشه های یک پوشه در سی شارپ

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 دیرکتوری های آن را بنویسد

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

```

1 static void Main(string[] args)
2 {
3
4     printfileInDir(@"c:\test\...");
5
6 }
7
8     private static void printfileInDir(string v)
9     {
10         var dir=Directory.GetDirectories(v);
11         printfile(v);
12         foreach(var d in dir )
13             printfileInDir(d);
14     }
15
16     private static void printfile(string v)
17     {
18         var files =Directory.GetFiles(v);
19         foreach(var f in files)System.Console.WriteLine(f);
20     }

```

نمونه کد ۸۸: تمام فایل های پوشه در سی شارپ

۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص

در این مرحله می خواهیم برنامه ای 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

```

1 static void Main(string[] args)
2 {
3     DirectoryIndex dirIdx = new DirectoryIndex(dir: @"C:\test\...");
4     dirIdx.CreateIndex("*.cs");
5     while(true)
6     {
7         System.Console.Write(" " "Query?");
8         string q = Console.ReadLine();
9         if (q == "exit")
10             break;
11
12         List<string> result = dirIdx.Query(q);
13         System.Console.WriteLine(" for found "Files + q);
14         foreach(var f in result)
15             System.Console.WriteLine(f);
16     }

```

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 ۹۲ در کلاس `DirectoryIndex` وظیفه تطابق رشته ورودی با اطلاعات

داخل فایل ها را بر عهده دارد

DirectoryIndex ۲.۲.۱۰

```

1 class DirectoryIndex
2 {
3     private string dir;
4     private Dictionary<string, List<string>> Index;
5
6     public DirectoryIndex(string dir)
7     {
8         this.dir = dir;
9         this.Index = new Dictionary<string, List<string>>();
10    }
11 }
```

نمونه کد : ۹۰

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

CreateIndex ۳.۲.۱۰

```

1 internal void CreateIndex(string filter)
2 {
3     List<string> allFiles = new List<string>();
4     GetAllFiles(dir, filter, ref allFiles);
5
6     foreach(var file in allFiles)
7     {
8         AddToIndex(file);
9     }
10 }
```

CreateIndex : ۹۱

* متد `GetAllFiles` نمونه کد ۹۲ با گرفتن آدرس یک پوشه و یک فیلتر از 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 ۹۲ تمام کلمات درون یک فایل را به عنوان key به دیکشنری تعریف شده در کلاس DirectoryIndex داده و لیستی از تمام فایل هایی که آن کلمه در آن وجود دارد را به عنوان value در نظر میگیرد.

GetAllFailes •

```

1  private void GetAllFiles(string subdir, string filter, ref List<string> allFiles)
2  {
3      var files = Directory.GetFiles(subdir, filter);
4      foreach(var file in files)
5          allFiles.Add(file);
6
7      foreach(var d in Directory.GetDirectories(subdir))
8          GetAllFiles(d, filter, ref allFiles);
9  }

```

نمونه کد ۹۲

AddToIndex •

```

1  private void AddToIndex(string file)
2  {
3      string [] tokens =
4          File.ReadAllText(file).Split(' ', ',', '.', '(', ')', '\n', '\r', ';');
5
6      foreach(var tok in uniqueTokens)
7      {
8          if (!string.IsNullOrWhiteSpace(tok))
9          {
10              if (!this.Index.ContainsKey(tok))
11                  this.Index.Add(tok, new List<string>());
12
13              this.Index[tok].Add(file);
14          }
15      }
16  }

```

نمونه کد ۹۳

Query ۴.۲.۱.

```

1  public List<string> Query(string q)
2  {
3      if (this.Index.ContainsKey(q))
4          return this.Index[q];
5
6      return new List<string>();
7  }

```

نمونه کد ۹۴

- * 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 [liberty^{۲۰۰۵}programming] C# Notes for Professionals
[csharp]

جلسه ۱۱

شبیه سازی چیلین وارز

شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹

در این جلسه صفحه‌ی چیلین وارز را شبیه سازی می‌کنیم.

هدف اصلی این جلسه نحوه‌ی فکر کردن به مسئله است و نه جزئیات حل آن. در این جلسه به ارتباط بین کلاس‌ها و آبجکت‌ها دقت کنید و نحوه‌ی استفاده از کیو را تمرین کنید.

در ادامه‌ی این جلسه از مفاهیم زیر استفاده می‌کنیم:

enum •

queue •

switch •

۱.۱۱ حل مسئله

مرحله‌ی اول حل مسئله، استفاده از کلاس‌ها و متود‌های لازم است و سپس پیاده سازی هرکدام از آن‌ها.

```

1 static void Main(string[] args)
2 {
3     Table t = new Table(rows: 10, cols: 8);
4     t.Player1 = new Player(row: 2, col: 3, table: t);
5     t.Player2 = new Player(row: 8, col: 5, table: t);
6     t.Print();
7 }
```

نمونه کد ۹۵: کد اولیه در program.cs

به این منظور، برای شبیه سازی صفحه‌ی چیلین و ارز مشابه نمونه کد ۹۵ ابتدا از کلاس‌های table و player و متود‌های داخل هر یک، مثل print استفاده می‌کنیم و سپس هر یک از این کلاس‌ها و متود‌ها پیاده سازی می‌کنیم.

در قسمت `new Table` روی کلمه‌ی Table کنترل و . را می‌زنیم و گزینه‌ی

`Generate type 'table' -> Generate class 'table' in new file` را انتخاب می‌کنیم.

همین مراحل را برای Player نیز انجام می‌دهیم تا کلاس‌های Table و Player ساخته شوند.

در `Table.cs` بهتر است اسم ممبر و رایبرل‌ها در سی‌شارپ، پاسکال کیس باشد برای همین روی `Rows`، `cols`، `cols` جدایگانه F2 زده و اسم آن‌ها را به `Rows`، `Cols` تغییر می‌دهیم.

برای نمایش صفحه‌ی بازی یک آرایه‌ی دو بعدی از کاراکتر‌ها به عنوان ممبر و رایبل می‌سازیم و آن را در کانسٹراکتور، همانطور که در نمونه کد ۹۶ می‌بینید، نیو می‌کنیم :

```

1 internal class Table
2 {
3     public int Rows;
4     public int Cols;
5     private char[,] Board;
6
7     public Table(int rows, int cols)
8     {
9         this.Rows = rows;
10        this.Cols = cols;
11        this.Board = new char[rows, cols];
12    }
13 }
```

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، سلوول خالی و بازیکن استفاده کنیم. به این منظور از `Enum` استفاده می کنیم.

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

```

1 enum CellType
2 {
3     Empty, Wall, Player
4 }
```

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:

```

1 private void FillBoard(CellType empty)
2 {
3     for(int i=0; i<Rows; i++)
4         for(int j=0; j<Cols; j++)
5             Board[i,j] = CellType.Empty;^^I
6 }
```

نمونه کد ۹۸: متود `FillBoard` در کلاس Table

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

```

1  private void MakeWalls()
2  {
3      for(int i=0; i<Rows; i++)
4      {
5          Board[i, 0] = CellType.Wall;
6          Board[i, Cols-1] = CellType.Wall;
7      }
8      for(int i=0; i<Cols; i++)
9      {
10         Board[0, i] = CellType.Wall;
11         Board[Rows-1, i] = CellType.Wall;
12     }
13 }
```

نمونه کد ۹۹: متود MakeWalls در کلاس Table

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

```
public Player Player2 و public Player Player1
```

دو بازیکن را در کلاس Table تعریف می کنیم.

در این قسمت به دو موضوع باید توجه کرد:

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

برای مسئله‌ی اول باید در کد اولیه‌ی خود، متغیرهای Player را به متغیرهای Table اضافه کنیم و در کلاس Player به منظور استفاده از این متغیر، تغییراتی اعمال می کنیم:

```

1 internal class Player
2 {
3     public int Row;
4     public int Col;
5
6     private Table Table;
7     public Player(int row, int col, Table table)
8     {
9         this.Table = table;
10        if (row < table.Rows && col < table.Cols)
11        {
12            this.Row = row;
13            this.Col = col;
14        }
15        else
16            Console.WriteLine("Error");
17    }
18}

```

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:

```

1 public int Rows {get; private set;}
2 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:

```

1 private Player _Player1;
2
3 public Player Player1
4 {
5     get { return _Player1; }
6     set {
7         this._Player1 = value;
8         Board[value.Row, value.Col] = CellType.Player;
9     }
10}

```

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 Print متود را به صورت نمونه کد

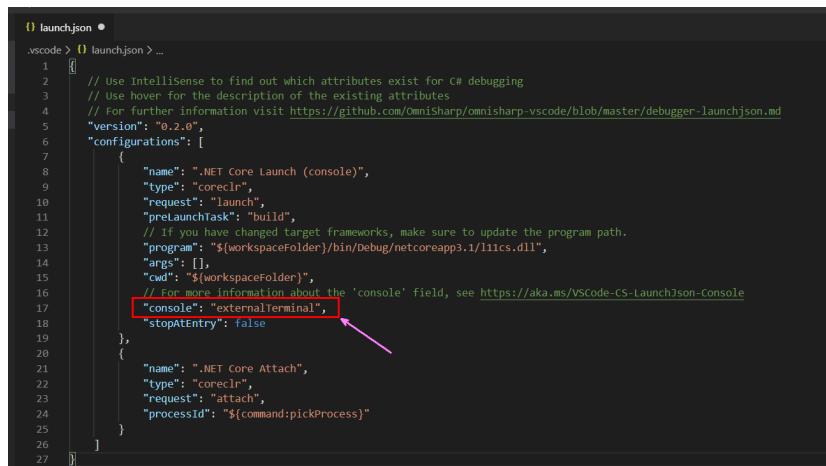
۱۰۳ می نویسیم:

```

1 public void Print()
2 {
3     Console.Clear();
4     PrintFirstLine();
5     for (int i = 0; i < Rows; i++)
6     {
7         Console.Write(i + " ");
8         for (int j = 0; j < Cols; j++)
9             PrintCell(i, j);
10            Console.WriteLine();
11    }
12 }
```

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 ۱۱۱ از اینترنال کنسول به اکسترنال ترمینال تغییر داد.



```

{} launch.json •
vscode > {} launch.json > ...
1  {
2      // Use Intellisense to find out which attributes exist for C# debugging
3      // Use hover for the description of the existing attributes
4      // For further information visit https://github.com/OmniSharp/omnisharp-vscode/blob/master/debugger-launchjson.md
5      "version": "0.2.0"
6      "configurations": [
7          {
8              "name": ".NET Core Launch (console)",
9              "type": "coreclr",
10             "request": "launch",
11             "preLaunchTask": "build",
12             // If you have changed target frameworks, make sure to update the program path.
13             "program": "${workspaceFolder}/bin/Debug/netcoreapp3.1/lilics.dll",
14             "args": [],
15             "cwd": "${workspaceFolder}",
16             // For more information about the 'console' field, see https://aka.ms/VSCode-CS-LaunchJson-Console
17             "console": "externalTerminal",
18             "stopAtEntry": false
19         },
20         {
21             "name": ".NET Core Attach",
22             "type": "coreclr",
23             "request": "attach",
24             "processId": "${command:pickProcess}"
25         }
26     ]
27 }
```

شكل ۱۱۱ : externalTerminal

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

```

1  private void PrintFirstLine()
2  {
3      Console.Write(" ");
4      for (int i = 0; i < Cols; i++)
5          Console.Write(i + " ");
6      Console.WriteLine();
7  }

```

نمونه کد ۱۰۴ : متود PrintFirstLine در کلاس Table

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

```

1  private void PrintCell(int i, int j)
2  {
3      CellType ct = Board[i,j];
4      switch(ct)
5      {
6          case CellType.Empty:
7              Console.Write('-');
8              break;
9
10         case CellType.Player:
11             int playerNumber = GetPlayer(i, j);
12             Console.Write(playerNumber);
13             break;
14
15         case CellType.Wall:
16             Console.Write('w');
17             break;
18     }
19     Console.Write(' ');
20 }

```

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:

```

1  private int GetPlayer(int r, int c)
2  {
3      int playerNumber = -1;
4
5      if (Player1.Col == c && Player1.Row == r)
6          playerNumber = 1;
7      else if (Player2.Col == c && Player2.Row == r)
8          playerNumber = 2;
9      else
10         Console.WriteLine("ERROR");
11
12     return playerNumber;
13 }
```

نمونه کد ۱۰۶: متود GetPlayer در کلاس Table

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

	0	1	2	3	4	5	6	7
0	W	W	W	W	W	W	W	W
1	W	-	-	-	-	-	-	W
2	W	-	-	1	-	-	-	W
3	W	-	-	-	-	-	-	W
4	W	-	-	-	-	-	-	W
5	W	-	-	-	-	-	-	W
6	W	-	-	-	-	-	-	W
7	W	-	-	-	-	-	-	W
8	W	-	-	-	2	-	W	
9	W	W	W	W	W	W	W	W

شکل ۲.۱۱: اجرای Main در Program.cs

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

```

1  private void PrintCell(int i, int j)
2  {
3      var color = Console.ForegroundColor;
4      CellType ct = Board[i,j];
5      switch(ct)
6      {
7          case CellType.Empty:
8              Console.ForegroundColor = ConsoleColor.Green;
9              Console.Write('‐');
10             break;
11
12          case CellType.Player:
13              Console.ForegroundColor = ConsoleColor.Red;
14              int playerNumber = GetPlayer(i, j);
15              Console.Write(playerNumber);
16              break;
17
18          case CellType.Wall:
19              Console.ForegroundColor = ConsoleColor.Yellow;
20              Console.Write('w');
21              break;
22
23          case CellType.Visiting:
24              Console.ForegroundColor = ConsoleColor.Cyan;
25              Console.Write('o');
26              break;
27      }
28      Console.Write(' ');
29      Console.ForegroundColor = color;
30  }

```

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:

```

0 1 2 3 4 5 6 7
0 W W W W W W W W
1 W - - - - - - W
2 W - - 1 - - - W
3 W - - - - - - W
4 W - - - - - - W
5 W - - - - - - W
6 W - - - - - - W
7 W - - - - - - W
8 W - - - 2 - W
9 W W W W W W W W

```

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 بدھیم. بنابراین کلاس وکتور را تعریف می کنیم و به صورت زیر از آن استفاده می کنیم:

```

1 Vector v = new Vector(row: 1, col: 1);
2
3 while ('q' != Console.ReadKey().KeyChar)
4 {
5     t.Player1.Move(v);
6 }
```

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 ای بالا اگر کاربر کلید q را فشار دهد، برنامه ای 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:

```

1 internal void Move(Vector v)
2 {
3     this.Row += v.row;
4     this.Col += v.col;
5     Table.Update();
6 }
```

نمونه کد ۱۰۹ : متود Move در کلاس Player

توجه کنید که Col ، Row 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:

```

1 public void Update()
2 {
3     for(int i=1; i<Rows-1; i++)
4         for(int j=1; j<Cols-1; j++)
5         {
6             if ((Player1.Row == i && Player1.Col == j) ||
7                 (Player2.Row == i && Player2.Col == j) )
8                 Board[i,j] = CellType.Player;
9             else
10                Board[i,j] = CellType.Empty;
11         }
12 }
```

نمونه کد ۱۱۰ : متود Update در کلاس Table

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

```

1 while ('q' != Console.ReadKey().KeyChar)
2 {
3     t.Player1.Move(v);
4     v.Negate();
5     t.Player2.Move(v);
6     t.Print();
7 }
```

نمونه کد ۱۱۱ : تغییر Main در Program.cs

می خواهیم جهت حرکت را بر عکس 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 صفحه‌ی کنسول یا اکسٹرناال ترمینال را پاک می کند. اگر این دستور را اضافه نکنیم، با هر بار اجرا شدن حلقه، یک صفحه‌ی بازی زیر صفحه‌ی کشیده شده‌ی قبلی چاپ می کند.

متود Negate را در کلاس وکتور می نویسیم:

```

1 public void Negate()
2 {
3     row = -1 * row;
4     col = -1 * col;
5 }
```

نمونه کد ۱۱۲: متود Negate در کلاس Vector

حالا می خواهیم نزدیک ترین دیوار به یکی از پلیرها را بیابیم. برای این کار، چهار خانه‌ی 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 خانه‌های چک شده‌ی 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:

```

1 public void BFSVisit(int i, int j)
2 {
3     Queue<Vector> toVisit = new Queue<Vector>();
4     toVisit.Enqueue(new Vector(i, j));
5
6     while (toVisit.Count != 0)
7     {
8         var v = toVisit.Dequeue();
9         Board[v.row, v.col] = CellType.Visiting;
10        foreach (Vector n in GetNeighbors(v))
11        {
12            if (Board[n.row, n.col] != CellType.Visiting)
13                toVisit.Enqueue(n);
14        }
15        Print();
16        Console.ReadKey();
17    }
18 }
```

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:

```

1  private List<Vector> GetNeighbors(Vector v)
2  {
3      List<Vector> neighbors= new List<Vector>();
4      if (v.row-1 > 0)
5          neighbors.Add(new Vector(v.row-1, v.col));
6      if (v.col-1 > 0)
7          neighbors.Add(new Vector(v.row, v.col-1));
8      if (v.row+1 < Rows-1)
9          neighbors.Add(new Vector(v.row+1, v.col));
10     if (v.col+1 < Cols-1)
11         neighbors.Add(new Vector(v.row, v.col+1));
12
13     return neighbors;
14 }
```

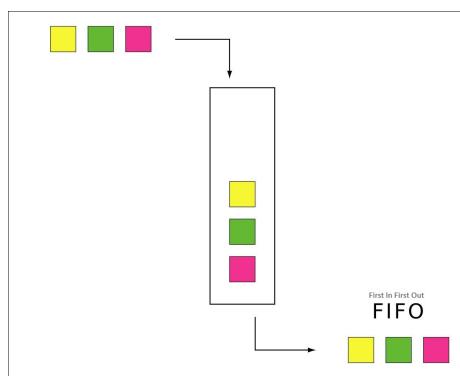
نمونه کد ۱۱۴ : متود GetNeighbors در کلاس Table

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

Queue ۲.۱۱

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.



شكل ۴.۱۱ out first in first

يعنى عضوي که زودتر از همه اضافه شده، زودتر از همه هم خارج می شود. مهم ترین متود های قابل استفاده در کيو، دو چيز است:

- که عضوي را به اول کيو اضافه می کند.
- که آخرین عضو کيو را خارج می کند.

حالا در متود PrintCell کيس ويزيتينگ را اضافه می کنيم:

```

1 case CellType.Visiting:
2     Console.ForegroundColor = ConsoleColor.Cyan;
3     Console.Write('o');
4     break;

```

نمونه کد ۱۱۵: تغيير متود PrintCell در کلاس Table

این قسمت از کد، خانه های ويزيت شده را با آبی رنگ نشان می دهد. خط `t.BFSVisit(5,4);` را به Main در Program.cs اضافه می کنيم تا نحوه ای کار متودی که نوشتم را ببینيم. خروجي ما به شکل زير خواهد بود:

شكل ۱۱.۵: اجرای متود BFSVisit

مطابق انتظار ما، اين متود به ترتيب 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 بروای هر یک از همسایه ها به عنوان خانه ای قبل ذخیره کنیم، تا بعد از پیدا کردن مقصد، با توجه به خانه های قبل که ذخیره

شده اند مسیری از خانه‌ی شروع تا مقصد پیدا کنیم.

برای این‌که مفهوم کیو را بهتر درک کنیم، مثال زیر را پیاده‌سازی می‌کنیم:

```

1 static void Main(string[] args)
2 {
3     Queue<string> myQ = new Queue<string>();
4     myQ.Enqueue("Roohandeh");
5     myQ.Enqueue("Yaghini");
6     myQ.Enqueue("Shahibzadeh");
7     LetGo(myQ);
8     myQ.Enqueue("Azari");
9     myQ.Enqueue("Behkam");
10
11    while (myQ.Count > 0)
12        LetGo(myQ);
13
14
15    private static void LetGo(Queue<string> myQ)
16    {
17        string person = myQ.Dequeue();
18        Console.WriteLine(person);
19        Console.ReadKey();
20    }

```

نمونه کد ۱۱۶: مثالی برای درک بهتر کیو در سی‌شارپ

همانطور که متوجه شده‌اید، با هر بار Dequeue کردن، اولین عضوی که وارد شده بود خارج می‌شود.

```

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);
    }
}

```

شکل ۱۱.۶: خروجی ۱۱۶

برای یادگیری بهتر، می‌توانید داکیومنت ماکروسافت را در زمینه‌ی 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 کیو در پایتون، می توانید داکیومنت پایتون را مطالعه کنید [python].

جلسه ۱۲

stacks- objects

بنفسه 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 نفاوت این که کیو یا صف، First in first out است. 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 اخرين کتابی که روی کتاب دیگر گذاشته میشود، اولین کتاب قابل دسترس است.

استک در csharp اینگونه تعریف میشود:

```

1  using System.Collections.Generic;
2
3  public class Program
4  {
5      static void Main(string[] args)
6      {
7          Stack<string> stack = new Stack<string>();
8      }
9 }
```

نمونه کد ۱۱۷ : تعریف کلاس استک

۲.۱.۱۲ call stack

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

به این نمونه کد توجه کنید:

```

1  using System;
2
3  namespace c16
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              MethodA();
10         }
11         static void MethodA()
12         {
13             Console.WriteLine(Before" A: "In");
14             MethodB();
15             Console.WriteLine(After" A: "In");
16         }
17
18         static void MethodB()
19         {
20             Console.WriteLine(Before" B: "In");
21             MethodC();
22             Console.WriteLine(After" B: "In");
23         }
24
25         static void MethodC()
26         {
27             Console.WriteLine(C" "In);
28         }
29
30     }
31 }

```

نمونه کد ۱۱۸ : call stack

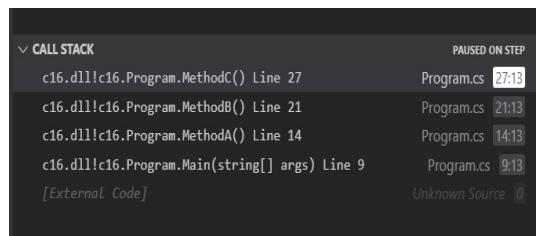
در این مثال، زمانی که چند متادودرتو در یک دیگر صدا زده می شوند، تا زمانی که متاد باز شده بسته نشود، نمی توان تا انتهای برنامه پیش رفت.

پس به ترتیب از اخر به اول، هر متادی 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 متدهای فراخوانی شده، بالا تر از همه قرار دارد.

همان طور که در تصویر ۱.۱۲ می بینید، دقیقاً مانند کتاب هایی که به ترتیب روی یک دیگر اند، این متدهای نیز به ترتیب روی هم فراخوانی شده اند.

stack's API ۳.۱.۱۲

API مخفف Application Programming Interface است. منظور واسطه هایی است که برای هر کلاس در سی شارپ تعریف شده است. درواقع همان استفاده از متدهای پیاده سازی شده، برای آن کلاس بخصوص می باشد.

در API استک، بیشترین و کاربردی ترین متدها 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 بر می گرداند.
برای بهتر متوجه شدن، به این نمونه کد توجه کنید:

```

1  using System;
2  using System.Collections.Generic;
3  class program
4  {
5      static void Main(string[] args)
6      {
7          Stack<string> stack = new Stack<string>();
8          stack.Push("Main");
9          stack.Push("MethodA");
10         stack.Push("MethodB");
11         stack.Push("MethodC");
12
13         while (stack.Count > 0)
14             Console.WriteLine(stack.Pop());
15     }
16 }
```

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()` آخرین داده را از استک حذف ، و اول از همه آن را بر می گرداند و چاپ می کند! و در این حلقه به اندازه استک، به ترتیب این عمل را اجرا می کند تا همه می اعضا چاپ بشوند.

۴.۱.۱۲ کاربرد

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

```

1 program class Program
2     private static bool IsBalanced(string s)
3     {
4         Stack<char> stack = new Stack<char>();
5         foreach(char c in s)
6         {
7             if (c == '(' || c == '[')
8                 stack.Push(c);
9
10            if (c == ')' || c == ']')
11                if ( !stack.TryPop(out char top) || !IsCompatible(c, top))
12                    return false;
13        }
14        return stack.Count == 0;
15    }
16    private static bool IsCompatible(char c, char t)
17    {
18        if (c == ')' && t == '(')
19            || (c == ']' && t == '['));
20    }
21
22    static void Main(string[] args)
23    {
24        string a = ")" + (a+c) / (a+b) * (a-b) [( * "(a+b";
25        string b = ")" (a+c) / (a+b) * (a-b) ( * "(a+b;
26        Console.WriteLine(IsBalanced(a));
27        Console.WriteLine(IsBalanced(b));
28    }
29 }
```

نمونه کد ۱۲۰: پرانتزگذاری صحیح

خروجی:

```
True
False
```

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

و اگر بسته میشود، با کمک متدهای TryPop() و isCompatible() که هردو بولین هستند، پرانتزها را مقایسه میکند و اگر تا زمانی که اندازه استک صفر شود شرط برقرار باشد، نحوه پرانتزگذاری ها صحیح است.

متدهای isCompatible() نیز اخیرین داده استک را با پرانتزبسته شده مقایسه میکند و اگر ترتیب آن صحیح بود، ارزش این متدهای true میباشد.

objects ۲.۱۲

۱.۲.۱۲ تعریف

همان طور که میدانیم، میتوان یک کلاس را پیاده سازی کرده و شیی را جداگانه برایش تعریف کرد. اما زمانی که بخواهیم متدهای 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 متدهای عام دیگر مانند:

lrEquals(), ToString(), GetType(), GetHashCode()

است که چرا برای هر شیی از انواع مختلف که مینویسیم، 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 میتوان آن را به صورت یک شی جدا نوشت، که شامل ویژگی ها و متدهای بالا است. در واقع هر شیی که ما تعریف میکنیم، به نوعی یک نوع object است.

میتوان این طور گفت که تمام ویژگی های آبجکت، در هر شی موجود است.

csharp

```

1 public class Program
2 {
3     Object obj;
4     obj = new Object();
5 }
```

نمونه کد ۱۲۱: تعریف آبجکت در سی شارپ

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

overriding ۲.۲.۱۲

می دانیم به هر حال، هر شی به نوعی از آبجکت ارث بری می کند. زمانی که بتوانیم متدهایی 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 مترا **override** کردیم.

در این جلسه ما **override** کردن سه نوع مترا می آموزیم:

- equals()
- ToString()
- GetHashCode()

equals ۳.۲.۱۲

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:

```

1 static void main
2 {
3     Student s = new Student(name: "Ali", id:98521231);
4     Student s1 = new Student(name: "Zahra", id:77521231);
5     Student s2 = new Student(name: "Ali", id:98521231);
6     s3 = s;
7     Console.WriteLine(s.Equals(s3))
8     Console.WriteLine(s.Equals(s1));
9     Console.WriteLine(s.Equals(s2));
10
11 }
```

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 **s2** از لحاظ ویژگی یکسانند)

برای این که بخواهیم دو شی منحصر 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:

```

1 internal class Student
2 {
3     private string name;
4     private int id;
5
6     public Student(string name, int id)
7     {
8         this.name = name;
9         this.id = id;
10    }
11
12    public override bool Equals(object obj)
13    {
14        if (!(obj is Student))
15            return false;
16        Student other = (Student) obj ;
17        Student other = obj as Student;
18
19        return this.Id == other.Id ;
20    }

```

نمونه کد ۱۲۳ :

در این قسمت ما تعریف کردیم که اگر شی ما دانش آموز نبود، آن را **false** برگرداند و در صورت دانش آموز یودن، شی را به قالب کلاس دانش آموز دربیاورد، و اگر تمام ویژگی ها برابر باشند، مقدار برگردانده **true** باشد. در صورت تعریف این متده، 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 **nullrefer-ence 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:

```

1  public override bool Equals(object obj)
2  {
3      if ((obj is Student))
4          return this.Id == (obj as Student).Id && this.Name == (obj as Student).Name;
5
6      if ((obj is string))
7          return this.Name == (obj as string);
8
9      if ((obj is int))
10         return this.Id == (int) obj;
11
12     return false;
13 }
```

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 رشته‌ی 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`) با تغییر کارکرد تابع Equals به صورت نمونه کد ۱۲۴ حاصل کار به صورت زیر است:

```

1  public static void main
2  {
3      Student s = new Student(name: "Ali", id:98521231);
4      Student s1 = new Student(name: "Zahra", id:77521231);
5      //
6      Console.WriteLine(s.Equals(s1));
7      Console.WriteLine(s.Equals(s2));
8      Console.WriteLine(s.Equals(98521231));
9      Console.WriteLine(s.Equals("Ali"));
10     Console.WriteLine(s.Equals("Zahra"));
11     Console.WriteLine(s.Equals(98989898));
12 }
```

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 رشته‌ی حرفی چاپ کند. برای مثال:

```

1 static void Main(string[] args)
2 {
3     object obj = new object();
4     Console.WriteLine(obj);
5 }
```

نمونه کد ۱۲۶ :

این برنامه در صورت اجرا شدن System.object را چاپ می‌کند. می‌توان با تغییر کارکرد این متد،
شی از نوع کلاس خاص را با فراخوانی متد ToString تبدیل به رشته‌ی حرفی کرد. مثلاً این متد را می‌توان
در کلاس دانش اموز [۱۲۳] override کرد.

```

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

نمونه کد ۱۲۷ :

در متد main:

```

1 static void Main(string[] args)
2 {
3     student stu = new student("Ali", 98532445);
4     Console.Writeline(stu);
5     string s = stu.ToString();
6     Console.WriteLine(s)
7 }
```

نمونه کد ۱۲۸ :

و به ازای نمونه کد ۱۲۸ رشته‌ی حرفی

Ali : 98532445

چاپ می‌شود

GetHashCode ۵.۲.۱۲

متدهای زمانی است که می‌خواهیم 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 نباشد.

به همین علت، معمولاً متدهای Equals و GetHashCode با هم پیاده‌سازی می‌شوند. در این صورت، می‌توان از خود شی از نوع کلاس خاص، به عنوان کلید در دیکشنری استفاده کنیم.

برای مثال این متدهای برای کلاس داشت آموز [۱۲۳] override می‌کیم.

```

1 public override int GetHashCode()
2 {
3     return this.Id.GetHashCode() ^ this.Name.GetHashCode();
4 }
```

نمونه کد ۱۲۹ :

در متدهای Equals تعریف کردیم که در صورت برابر بودن نام و شماره‌ی دانشجویی دو شی یکسان نیز، آنها را برابر در نظر بگیرد. به همین علت در متدهای GetHashCode xor این دو ویژگی برگردانده می‌شود. به این صورت که از نام و یا شماره دانشجویی، می‌توان به عنوان کلید در دیکشنری استفاده کرد.

برای مثال:

```

1 static void main()
2 {
3     Dictionary<Student, List<int>> students=new Dictionary<Student, List<int>>();
4     Student ali = new Student(name:"Ali", id:98521231);
5     Student zahra = new Student(name: "Zahra", id:77521231);
6     students.Add(ali, new List<double>());
7     students.Add(zahra, new List<double>());
8
9     students[ali].Add(19);
10    students[zahra].Add(18);
11
12    Console.WriteLine(students[ali])
13    Console.WriteLine(students[zahra])
14
15 }^^I

```

نمونه کد ۱۳۰ :

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 نوشتتن 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 عنوان 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 Destructor برای یک کلاس خواهیم بود به طور مثال کلاس و توابع زیر در زبان C++ را در نظر بگیرید

```

1 class Test
2 {
3     int *pN;
4
5 public:
6     Test(int n)
7         : pN(new int[n])
8     {
9         cout << "pN=" of size ,Constructor Test "In << n << endl;
10    }
11
12 ~Test()
13 {
14     cout << "pN" of address ,Destructor "In << pN << endl;
15     delete[] pN;
16 }
17 };
18
19 void MethodForStackAllocDemo()
20 {
21     Test a(5);
22 }
23
24 Test *MethodForHeapAllocDemo()
25 {
26     Test *pTest = new Test(6);
27     return pTest;
28 }
```

نمونه کد ۱۳۱ : کلاس تست

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

```
1 Test(){};
```

نمونه کد ۱۳۲ : کانسٹراکتور

از آنجا که در تعریف این کلاس مستقیما از Heap برای ذخیره آرایه مان استفاده کرده ایم نیاز است تا هنگام پایان کار با یک object 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

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

C++ نمونه کد ۱۳۳ : دیستراکتور

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

این نکته شایان توجه است که در ساخت یک object جدید از کلاس تنها زمانی نیاز به استفاده از new داریم که بخواهیم پوینتر آن شی را به عنوان return value داشته باشیم.

اگر یک شی روی Stack تعریف شده باشد دیستراکتور در پایان هر scope 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 زیر اینها در محدوده شرط قابل دسترسی است

```
1 if (true)
2 {
3     Test b(15);
4     cout << statement" If "in << endl;
5 }
```

C++ نمونه کد ۱۳۴ :

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

```
1 int main()
2 {
3     Test *pTest = MethodForHeapAllocDemo();
4     delete pTest;
5 }
```

C++ نمونه کد ۱۳۵ :

در زبان برنامه نویسی C++ بسته به دلخواه برنامه نویس می توان یک کلاس را روی Heap یا Stack تعريف کرد این یکی از تفاوت های این زبان با سی شارپ است در سی شارپ هر کلاس لزوماً روی Heap تعريف می شود که در ادامه بیشتر به آن خواهیم پرداخت.

۲.۲.۱۳ فاینالایزر

سی شارپ دارای ویژگی به نام Garbage Collector است که مدیریت اتمات حافظه را ممکن می سازد. این به آن معناست که با مردمیریت حافظه های اختصاص یافته از دوش برنامه نویس برداشته می شود. برای اطلاعات بیشتر میتوانید به اینجا [visualizationwebsite] مراجعه کنید. تنها متغیر Garbage Collector ها و اشیایی که روی Heap دارند را مورد بررسی قرار میدهد و متغیرهای ذخیره شده روی Stack همانند C++ با پایان یافتن محدوده تعريف از دسترس خارج میشوند.

کلاس زیر در زبان سی شارپ را در نظر بگیرید

```

1   class Test
2   {
3       public int N {get; set;}
4       public Test(int n)
5       {
6           N = n;
7           Console.WriteLine($"N: {N}");
8       }
9
10      ~Test()
11      {
12          Console.WriteLine($"{N} finalizer In");
13      }
14 }
```

نمونه کد ۱۳۶ : کلاس تست

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

```

1   ~Test()
2   {
3       Console.WriteLine($"{N} finalizer In");
4   }
```

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 Private و Public بودن آن هاست اجزای کلاس ها به طور پیش فرض و اجزای 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.

کلاس ها Reference Type و Value Type یک struct

```

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 مفاهیم Value Type و Reference Type بیشتر خواهیم دانست

جلسه ۱۴

داده نوع های و Value Type Reference Type

مسعود گلستانه - ۱۳۹۹/۱/۱۸

جزوه جلسه ۱۱۴ مورخ ۱۳۹۹/۱/۱۸ درس برنامه سازی پیشرفته تهیه شده توسط مسعود گلستانه.

۱.۱۴ تفاوت میان داده های value type و reference 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 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 می شود کامپایلر ۴ بایت حافظه (۳۲ بیت) برای نگه داشتن مقدار عدد صحیح اختصاص دهد. در واقع حکمی که صادر میکند باعث می شود مقدار آن(مانند ۴۲) در این بلوک حافظه کبی 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 میکند). 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 (رشته‌های مختلف می‌توانند 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

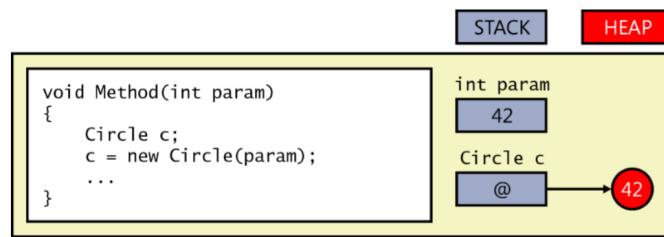
به نمونه کد سی شارپ زیر توجه کنید:

```

1 public class Circle
2 {
3     //some property
4 }
5 void Test(int param)
6 {
7     Circle c;
8     c = new Circle(param);
9     ...
10 }
```

کد نمونه ۱۴۰ : Csharp

فرض کنید مقدار منتقل شده به پارام ۴۲ است. وقتی متندست فراخوانی می‌شود ، یک بلوک حافظه (به اندازه لازم int) از پشتۀ اختصاص داده می‌شود و با مقدار ۴۲ مقدار دهی اولیه می‌شود. با اجرای داخل متند، 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 پشتۀ اختصاص داده می‌شود اما بدون مقدار دهی اولیه باقی می‌ماند. در مرحله بعد ، یک قطعه دیگر از حافظه به اندازه کافی بزرگ برای یک شیء دایره از پشتۀ اختصاص می‌یابد. این همان کاری است که کلمه کلیدی new انجام می‌دهد. کانسٹرکتور دایره تلاش می‌کند تا این حافظه پشتۀ خام را به یک شیء دایره تبدیل کند. رفرنس به این شیء دایره در متغیر c ذخیره می‌شود. تصویر ۱۱۴ این وضعیت را نشان می‌دهد.



شکل ۱۰.۱۴: تخصیص حافظه

- در اینجا باید به دو نکته توجه داشته باشید:

- اگرچه شیء در هیپ ذخیره می شود، اما رفرنس به شیء در پشتہ ذخیره می شود.
- حافظه هیپ بی نهایت نیست. اگر حافظه هیپ پر شود، اپراتور `new` استثناء `OutOfMemoryException` را پرتاپ می کند و شیء ایجاد نمی شود.

مهم: برای بسیاری از توسعه دهنگان (مانند توسعه دهنگان زبان های مدیریت نشده C / C ++، C / C ++ reference type و value type) در ابتدا عجیب به نظر می رسد. در آیا، شما یک تایپ را ایجاد می کنید، و سپس کدی که از آن تایپ استفاده می کند تصمیم می گیرد که آیا متغیر نمونه از این تایپ را باید در پشتہ یا هیپ برنامه ذخیره کند. در حالیکه در زبان های مدیریت شده (managed) مانند سی شارپ برنامه نویسی که تایپ را تعریف می کند، مشخص می کند که نمونه هایی از آن تایپ در کجا ذخیره می شوند. برنامه نویسی که از آن تایپ استفاده می کند، 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، 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 می دارند: یکی بلوک برای `i` و بلوک دیگر برای `Copyi`. اگر مقدار `i` را تغییر دهید، مقدار `Copyi` تغییر نمی کند.

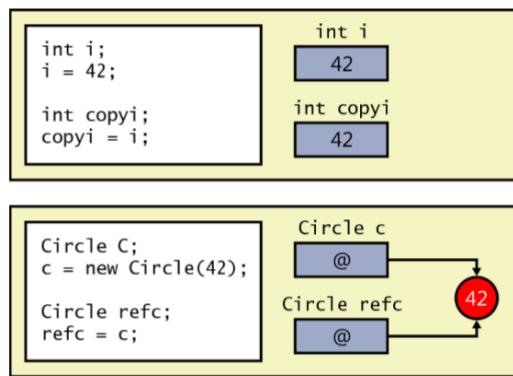
```

1 int i = 42;           // declare and initialize i
2 int copyi = i;        /* copyi contains a copy of the data in i and,
   both contain the value 42 */
3 i++;                 /* incrementing i has no effect on copyi;
   i now contains 43, but copyi still contains 42 */
4
5 Circle c = new Circle(42);
6 Circle refc = c;

```

کد نمونه ۱۴۱ Csharp

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 تگهدارنده `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 ، 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

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 نمونه‌ای از یک value type اشاره کنید.

۱.۳.۱۴ باکسینگ

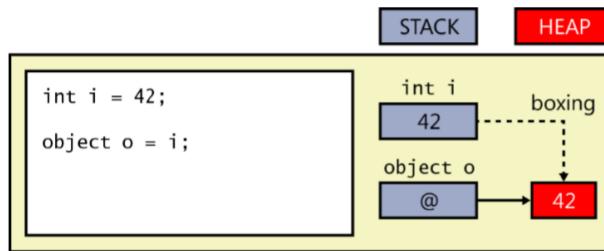
به عنوان مثال، در عبارت زیر متغیر `i` (از نوع `int` ، یک value type) با `42` و سپس متغیر `o` (از نوع `شيء`، یک نوع رفرنس) را با `o` مقداردهی اولیه می‌کنیم:

```
1 int i = 42;
2 object o = i;
```

۱۴۲ : کد نمونه 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 می‌شود. اما، همه رفرنس‌ها باید به اشیاء موجود در هیپ اشاره کنند. ایجاد رفرنس به موارد موجود در پشته می‌تواند استحکام زمان اجرا را به خطر بیندازد و یک نقص امنیتی بالقوه ایجاد کند، بنابراین این کار مجاز 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.

```

1 int i = 42;
2 object o = i; // boxes
3 i = (int)o; // unboxes

```

تأثیر این کست بسیار ظرفی 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. میتوانه 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 نکپی می کندو در غیر این صورت اکسپشن InvalidCastException رخ می دهد.

۴.۱۴ Nullable ها در سی شارپ

مقدار null برای مقداردهی رفرانس تایپ ها به کار می رود . و یک value type نمی تواند مقدار null را در خود نگه دارد. برای مثال `int i = null;` باعث می شود که در زمان اجرا با خطأ رو برو شوید. در سی شارپ نسخه ۲ انواع nullable معرفی شدند که اجازه می دهند مقدار null را به یک متغیر value type 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 :

```
1 Nullable<int> i = null;
```

یک نوع nullable علاوه بر این که دارای محدوده‌ی داده‌ای نوع مورد نظر خود است می تواند مقدار null را هم در خود نگه دارد. برای مثال `Nullable<int> i = null;` علاوه بر اینکه می تواند مقداری بین ۰-۲۱۴۷۴۸۳۶۴۷ را در خود نگه دارد، می تواند مقدار null را نیز در خود ذخیره کند. انواع nullable نمونه‌ای از ساختار `System.Nullable<T>` هستند.

میتوانید از عملگر `?!` به شکل `int? i` و `long? d` به جای `Nullable<T>` برای تعریف متغیر های Nullable استفاده نمایید :

```
1 int? i = null;
2 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; //illegal
```

اما بر عکس آن صحیح است و کاربردهای زیادی در برنامه ها دارد:

```
i = j; //legal
```

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 بر می خورد. لیست exception های متدائل 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.

```
1 public class Program
2 {
3     static void Main(string[] args){
4         string number = Console.ReadLine();
5         int count = int.Parse(number);
6         for (int i = 0; i < count; i++)
7         {
8             System.Console.WriteLine(i);
9         }
10    }
11 }
```

نمونه کد ۱۴۳ : نمونه یک exception

از exception handling error استفاده می‌کنیم. عبارت exception unhandled هنگامی رخ می‌دهد که خطای در قسمتی از برنامه وجود داشته باشد و ما آن را تصحیح نکرده باشیم. هر exception حاوی پیامی است که به طور خلاصه منشاء مشکل را به ما نشان می‌دهد. با جست و جو در کد و اینترنت می‌توانیم خطای رفع کنیم. داکیومنت های سایت مایکروسافت نیز برای پی‌بردن به اینکه هر دستور(مثلاً در اینجا int.Parse) چه ویژگی‌هایی دارد و چه exception‌هایی می‌تواند بدهد، مفید هستند.

۲.۱۵ رفع exception

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

```

1  using System;
2
3  namespace ErrorHandlingApplication {
4      class DivNumbers {
5          int result;
6          DivNumbers(){
7              result = 0;}
8          public void division(int num1, int num2) {
9              try {
10                  result = num1 / num2;}
11              catch (DivideByZeroException e) {
12                  Console.WriteLine("{0}" " caught: "Exception, e);}
13              finally {
14                  Console.WriteLine("{0}" "Result:, result);}
15              }
16          static void Main(string[] args) {
17              DivNumbers d = new DivNumbers();
18              d.division(25, 0);
19              Console.ReadKey();
20          }
21      }
22  }
```

نمونه کد ۱۴۴ : نمونه یک exception

در قطعه کد بالا چون ۲۵ یار تقسیم شده، برنامه به جای crash شدن، ابتدا وارد try شده و بعداز آنکه با خطای مواجه شد وارد catch می‌گردد و ارور DividedByZeroException را چاپ می‌کند.(e) یک متغیر است که به جای DividedByZeroException از آن استفاده می‌شود.)

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

شكل ۱.۱۵ : Exception Zero By Divided

در نهایت کد درون بلوک exception 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 وازه 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 حتی جلوی کلاس اکسپشن، Exception قرار داده شود:

```

1  class Student
2  {
3      public void StudentName(string studentName){
4          if (studentName == null)
5              throw new InvalidStudentNameException(
6                  "Invalid!" "Name");
7      }
8  }
9  class InvalidStudentNameException : Exception
10 {
11     public string StudentName;
12     public InvalidCourseIdException(string msg, string StudentName)
13         : base(msg)
14     {
15         this.StudentName = StudentName;
16     }
17 }
```

نمونه کد ۱۴۵ : Exception Class

```

1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Student newStudent = null;
6
7          try
8          {
9              newStudent = new Student();
10             newStudent.StudentName = "James007";
11
12             ValidateStudent(newStudent);
13         }
14         catch(InvalidStudentNameException ex)
15         {
16             Console.WriteLine(ex.Message );
17         }
18
19
20         Console.ReadKey();
21     }
22     private static void ValidateStudent(Student std)
23     {
24         foreach (char c in std.StudentName)
25         {
26             if (std.StudentName == "0")
27                 throw new InvalidStudentNameException(std.StudentName);
28         }
29     }
30 }
```

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

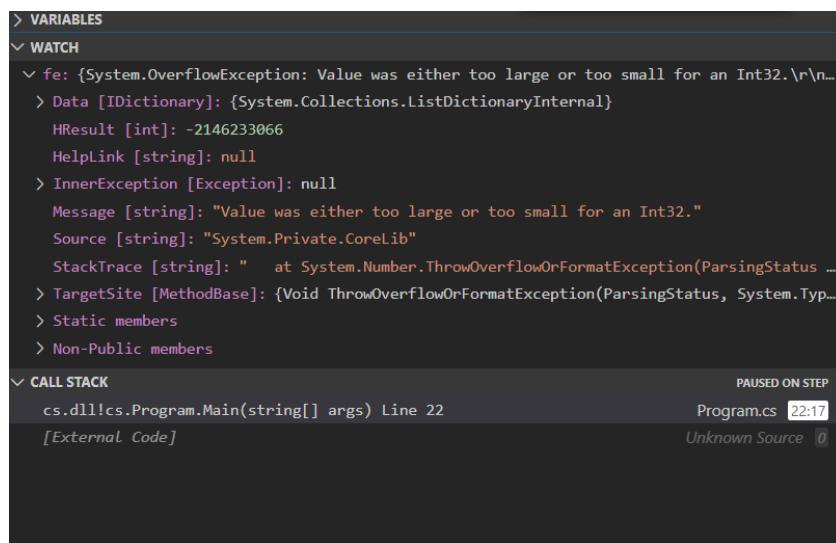
بعضی متغیرها مثل int و double و bool به خودی خود Nullable نیستند. مثلاً `int count = null` بعده ارور برمی خورد. برای همین از nullable value types استفاده می کنیم مثلاً `int? count` یا `double? count`.

```

1  using System;
2  static void Main()
3  {
4      int? count = null;
5      do
6      {
7          try
8          {
9              Console.Write(" integer> "Input);
10             string countString = Console.ReadLine();
11             count = int.Parse(countString);
12         }
13         catch (OverflowException)
14         {
15             Console.WriteLine(again. Try integer. Invalid);
16         }
17     }
18     while(count == null);
19     for (int i=0; i<count; i++)
20         Console.WriteLine(i);
21 }
```

نمونه کد ۱۴۷ throwing user defined exception :

با ورود یک عدد بسیار بزرگ مثل ۹۹۹۹۹۹۹۹۹۹، به ارور overflow exception برمی خوریم. وقتی exception یک پیام حاوی یک خطای debug هنگام کنید. هر exception یک پیام علت خطای را بیان می کند. هر 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"
```

Stack Trace : ۴.۱۵

عبارة 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 مدامی که شرط while برقرار بود exception گرفته می شوند.

```
1  using System;
2  static void Main()
3  {
4      int? count = null;
5      do
6      {
7          try
8          {
9              Console.Write(" integer> "Input);
10             string countString = Console.ReadLine();
11             count = int.Parse(countString);
12         }
13         catch (OverflowException)
14         {
15             Console.WriteLine(again. Try integer. Invalid);
16         }
17     }
18     while(count == null);
19     for (int i=0; i<count; i++)
20         Console.WriteLine(i);
21 }
```

نمونه کد ۱۴۸ While Do Example :

try/catch vs if/else ۵.۱۵

در برخی موارد به جای if else 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:

```

1  try/catch using exception IndexOutOfRangeException //  

2  int[] array = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  

3  int j = Convert.ToInt32(Console.ReadLine());  

4  try  

5  {  

6      int i = array[j];  

7  }  

8  catch (IndexOutOfRangeException)  

9  {  

10     Console.WriteLine("range" of out "Index");  

11 }  

12 if/else using exception IndexOutOfRangeException //  

13 int[] array = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  

14 int j = Convert.ToInt32(Console.ReadLine());  

15 if (array.Length > j && j > -1)  

{  

16     int i = array[j];  

17 }  

18 else  

19 {  

20     Console.WriteLine("range" of out "Index");  

21 }  

22 }
```

نمونه کد ۱۴۹ : catch/try vs else/if

به طور کلی همانطور که از اسم آن مشخص است، exception در شرایط استثنائی مثلاً مواردی که برنامه از حالت عادی خود خارج شده و به مشکل خاصی برمی خورد، ایجاد می شود. معمولاً از if/else برای اجرای یک شرط خاص بهره گرفته میشود که لزوماً 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 ارورها در یک جا، باعث راحتی کار ما در بررسی تمام شرایطی که در آن برنامه به خطأ برمی خورد می شود. نکته دیگری که باید به آن توجه کرد این است که در بلوک try کدی نوشته می شود که احتمال خطأ 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 خطأ در آن رخ داده است می باشد. پرتاب کردن یک exception جزئیات بیشتری درباره آن خطأ به ما ارائه می کند. به علاوه از catch/try برخلاف else/if به طور خاص برای رفع خطأ 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 خطأ استفاده کنیم، در مقیاس بزرگتر تعداد پیام های خطایی که باید بنویسیم بسیار زیاد و پیچیده خواهد شد. مثلاً 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 دقیقاً کدام ارور خواهد بود. به علاوه امکان اینکه بعضی خطاهای را یادمان برود رفع کنیم زیاد است:

```

1 private List<int> Coruses = new List<int>();
2 public string AddCourse(int courseId){
3     if (courseId < 100 || courseId >= 1000)
4     {
5         return Invalid!" Is ID "Course
6     }
7     Coruses.Add(courseId);
8     return null;
9 }
```

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

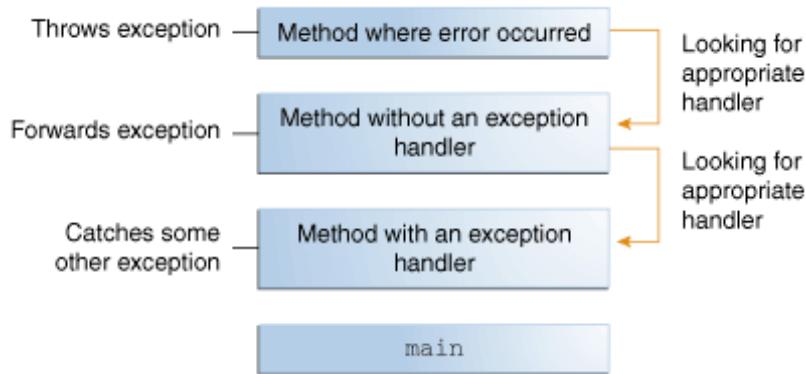
```

1 private List<int> Coruses = new List<int>();
2 public void AddCourse(int courseId)
3 {
4     if (courseId < 100 || courseId >= 1000)
5         throw new InvalidCourseIdException
6         ($"[courseId]" range: of out "courseId", courseId);
7     Coruses.Add(courseId);
8 }
```

نمونه کد ۱۵۱ throwing exception using exception throw :

Call Stack and Re-Throwing Exceptions ۷.۱۵

در برخی موارد وقتی یک exception ظاهر شده یا به اصطلاح پرتاپ می شود و سپس گرفته (catch) می شود، نیاز به دوباره پرتاپ شدن(re-throw) دارد تا با متدهایی که در لایه های زیرین call stack 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 متدهای وجود می آید. اگر در آن متدهای throw قرار داده باشیم، برنامه بعد از رسیدن به آن کد در حافظه استک به لایه پایینی یا به بیانی دیگر، متدهای قبلی در آن رجوع می کند. اگر در این متدهای 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 متدهای قبلی handle نشده باشد و درون بلوك catch کلید واژه throw قرار گیرد برنامه دوباره همین روند را با متدهای پایین تر در حافظه stack انجام می دهد تا زمانیکه همه خطاهای برطرف شده باشند و برنامه به روند عادی خود ادامه می دهد. در واقع اگر بتوانیم کیس را در بلوك catch/try یا else/if برنامه handle می کنیم، اگر نشد آن را پرتاب می کنیم.

۸.۱۵ تفاوت با throw new exception

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

```

1  namespace c10cs
2  {
3      class Student
4      {
5          private List<int> Coruses = new List<int>();
6          public static void RegisterStudent
7              (string name, int id, int[] courseIds)
8          {
9              Student s = new Student(name, id);
10             foreach(var courseId in courseIds)
11             {
12                 try
13                 {
14                     s.AddCourse(courseId);
15                 }
16                 catch(CourseNotOfferedException ide)
17                 {
18                     throw;
19                 }
20             }
21         }
22         public void AddCourse(int courseId)
23         {
24             if (courseId < 100 || courseId >= 1000)
25                 throw new InvalidCourseIdException(
26                     $"{courseId}" range: of out "courseId", courseId);
27
28             EnsureCourseIsOffered(courseId);
29             Coruses.Add(courseId);
30         }
31         void EnsureCourseIsOffered(int courseId)
32         {
33             if (courseId > 500)
34                 throw new CourseNotOfferedException(
35                     $"semester" this offered not is "Course, courseId");
36         }
37     }
38 }
```

نمونه کد ۱۵۲ : throw new exception vs throw

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

جلسه ۱۶

Exceptions & Operators

بیان دیوانی آذر - ۱۳۹۹/۱/۲۵

Exceptions ۱.۱۶

در جلسه‌ی پیش در رابطه با کاربرد Exception ها که چگونه باعث آسان شدن Error handling می‌شوند، گفته شد. این جلسه نحوه‌ی رخ دادن Exception ها را می‌بینیم.

```
1 static void Main(string[] args)
2 {
3     Console.WriteLine("Before" - Main "In");
4     MethodA();
5     Console.WriteLine("After" - Main "In");
6 }
```

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

```

1 private static void MethodA()
2 {
3     Console.WriteLine("Before" - A "In");
4     MethodB();
5     Console.WriteLine("After" - A "In");
6 }
```

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

```

1 private static void MethodB()
2 {
3     Console.WriteLine("Before" - B "In");
4     MethodC();
5     Console.WriteLine("After" - B "In");
6 }
```

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

```

1 private static void MethodC()
2 {
3     Console.WriteLine("Before" - C "In");
4     MethodD();
5     Console.WriteLine("After" - C "In");
6 }
```

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

```

1 private static void MethodD()
2 {
3     Console.WriteLine("Before" - D "In");
4     Console.WriteLine("After" - D "In");
5 }
```

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

```

1 private static void MethodD()
2 {
3     Console.WriteLine("Before" - D "In");
4     throw new InvalidOperationException();
5     Console.WriteLine("After" - D "In");
6 }
```

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 ۴ ام 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 شده است.

همینطور که می‌بینید در اینجا بعد از پرت شدن اکسپشن به خط دهم تابع ۱۰ Main 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 شده است.

```
1 private static void MethodB()
2 {
3     Console.WriteLine(Before" - B "In");
4     try
5     {
6         MethodC();
7     }
8     catch
9     {
10        Console.WriteLine(Catch" - B "In");
11        throw;
12    }
13    Console.WriteLine(After" - B "In");
14 }
```

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

```
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 شده است.

همینطور که می بینید در اینجا بعد از پرتاب شدن اکسپشن به خط دهم تابع MethodB ۱۰ می رویم و بعد دوباره اکسپشن رو پرتاب می کنیم و به اینجا ۱۱ می رویم.

```

1  private static void MethodB()
2  {
3      Console.WriteLine(Before" - B "In");
4      try
5      {
6          MethodC();
7          Console.WriteLine(C" Calling After - B "In");
8      }
9      catch
10     {
11         Console.WriteLine(Catch" - B "In");
12     }
13     finally
14     {
15         Console.WriteLine(Finally" - B "In");
16     }
17     Console.WriteLine(After" - B "In");
18 }
```

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

```

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 شده است. و در MethodD اضافه شده است. (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 نخواهد شد)

نکته‌ی جالب! حتی اگر ما قبل از ورود به بلاک finally برگردیم (return کنیم)، کد این بلاک اجرا خواهد شد.

```
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 مکالمه شده است و در MethodD (finally) اضافه شده است اکسپشن پرتاب شده است

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

```
1  private static void MethodB()
2  {
3      Console.WriteLine("Before" - B "In");
4      try
5      {
6          MethodC();
7          Console.WriteLine(C" Calling After - B "In");
8      }
9      catch (OverflowException)
10     {
11         Console.WriteLine(Catch" - B "In);
12     }
13     finally
14     {
15         Console.WriteLine(Finally" - B "In);
16     }
17     Console.WriteLine(After" - B "In);
18 }
```

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

ممکن! اگر با انواع اکسپشن ها آشنا نیستید، میتوانید با مراجعه به اینجا [stackify] در قسمت با انواع اکسپشن ها در سی شارپ آشنا شوید. Common .NET Exceptions

```
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 می‌شویم و خط **۱۶** اجرا نمی‌شود و بعد وارد بلاک finally می‌شویم و بعد **۱۶** می‌رویم.

```
1 private static void MethodB()
2 {
3     Console.WriteLine(Before" - B "In");
4     StreamReader reader = null;
5     try
6     {
7         reader = new StreamReader("in.txt");
8         string line = reader.ReadLine();
9         MethodC();
10        Console.WriteLine(C" Calling After - B "In");
11    }
12    catch (OverflowException)
13    {
14        Console.WriteLine(Catch" - B "In");
15    }
16    finally
17    {
18        reader.Dispose();
19        Console.WriteLine(Finally" - B "In");
20    }
21    Console.WriteLine(After" - B "In");
22 }
```

نمونه کد ۱۶۳: مثالی از کاربرد بلاک finally

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

Indexers ۲.۱۶

با تعریف Indexer برای کلاس مان 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 محدوده‌ی Indexer تان توجه کنید تا ناخواسته به اکسپشن برخورید.

```
1 class PBEntry
2 {
3     public PBEntry this[string name]
4     {
5         get
6         {
7             return Data[name];
8         }
9         set
10        {
11            this.Data[name] = value;
12        }
13    }
14 }
```

نمونه کد ۱۶۵ : مثالی از Indexer

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

```

1 struct IntBits
2 {
3     public bool this[int index]
4     {
5         get
6         {
7             return (bits & (1 << index)) != 0;
8         }
9         set
10        {
11            if (value)
12                bits |= (1 << index);
13            else
14                bits &= ~(1 << index);
15        }
16    }
17 }
```

نمونه کد ۱۶۶: قطعه کد نمونه برگرفته از کتاب [sharp ۲۰۱۳ microsoft]

۱۷ جلسه

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 یه نوع دیگری که می خواهید تبدیل می کند .

• تفاوت explicit و implicit

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 دارد که در ان اپراتور + را overload کرده ایم به کد زیر جهت بقیه توضیحات توجه کنید :

```

1   class Time
2   {
3       private int h;
4       private int m;
5
6       public Time(int h, int m)
7       {
8           this.h = h;
9           this.m = m;
10      }
11
12      public Time(Time other)
13      {
14          this.h = other.h;
15          this.m = other.m;
16      }
17
18      public void AddTo(Time t)
19      {
20          int newValue = t.m + this.m;
21          this.m = newValue % 60;
22          this.h = t.h + this.h + (newValue / 60);
23      }
24
25      public static Time operator+(Time lhs, Time rhs)
26      {
27          Time t = new Time(lhs);
28          t.AddTo(rhs);
29          return t;
30      }
31  }

```

implicit ۱.۱.۱۷

حال می خواهیم یک ساعت، به ساعت دیگری اضافه کنیم :

```

1     Time t = new Time(12, 30);
2     Time t2 = new Time(1, 0);
3     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 :

```

1     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:

```

1   public static Time operator+(Time lhs, Time rhs)
2   {
3       Time t = new Time(lhs);
4       t.AddTo(rhs);
5       return t;
6   }
7
8   public static implicit operator Time(int fromHour)
9   {
10      return new Time(fromHour, 0);
11  }
```

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:

```
1   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.

```

1   public static Time Parse(string time)
2   {
3       int colPos = time.IndexOf(':');
4       string hour = time.Substring(0, colPos);
5       string minute = time.Substring(colPos+1);
6       return new Time(int.Parse(hour), int.Parse(minute));
7   }
8
9   public static implicit operator Time(string time) => Parse(time);
```

- برای بهتر شدن مفهوم به کد زیر توجه کنید :

```
1 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 با نوع int است به Time تبدیل شده سپس این داده که حالا از نوع Time است با t7 جمع می شود سپس ۴ که از نوع Time 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 :

```
1 Time t9 = (Time) 5.0;
```

در این مثال می خواهیم داده ای از نوع double را مستقیماً به نوع Time تبدیل کنیم پس می توانیم از explicit conversion استفاده کنیم:

```
1 public static explicit operator Time(double time)
2 {
3     int hour = (int) time;
4     int minute = (int) ((time - hour) * 60);
5     return new Time(hour, minute);
6 }
```

Pairs Operator ۲.۱۸

```
== !=  
< >  
<= >=
```

این نوع اپراتور ها باید به طور جفت پیاده سازی شوند و bool برمی گردانند.

=!، == ۱.۲.۱۷

به کد زیر توجه کنید :

```

1  public override bool Equals(object obj)
2  {
3      Time t = obj as Time;
4      if(t != null)
5          return t.h == this.h && t.m == this.m;
6      return false;
7  }
8
9  public static bool operator == (Time lhs,Time rhs)
10 {
11     return !lhs.Equals(rhs);
12 }
13
14 public static bool operator !=(Time lhs,Time rhs)
15 {
16     return !lhs.Equals(rhs);
17 }
```

== پیاده‌سازی کرده این ولی کد بالا StackOverflowError می‌دهد . نحوه اجرا شدن کد بالا به این صورت است که ابتدا Equals در خط ۱۱ صدا می‌شود . در تابع خط ۴ ، t از نوع کلاس Time است پس بعد از این خط ، خط ۱۴ و سپس ۱۶ اجرا می‌شود، بعد از خط ۱۶ که در آن باز Equals صدا زده می‌شود ، خط اول و دوباره خط ۴ اجرا می‌شود و همینگونه ادامه پیدا می‌کند . برای جلوگیری از این مشکل می‌توانیم از ReferenceEquals استفاده کنیم و به جای صدا کردن اپراتور reference شان را با هم مقایسه کنیم :

```
1  public static bool ReferenceEquals (object objA, object objB);
```

همانطور که می‌بینید این Method اگر دو object مثل هم باشند true و در غیر این صورت false برمی‌گرداند .

```

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 می‌توانیم به صورت زیر بنویسیم:

```

1 public static bool operator ==(Time lhs, Time rhs)
2 {
3     if (!ReferenceEquals(lhs, null))
4         return lhs.Equals(rhs);
5     return false;
6 }
7
8 public static bool operator !=(Time lhs, Time rhs) => !(lhs == rhs);

```

=<, == and <, > ۲.۲.۱۷

< بعده > است که برای مثال می‌توانید به کد زیر توجه کنید:

```

1 public static bool operator <(Time lhs, Time rhs)
2 {
3     if (lhs.h == rhs.h)
4         return lhs.m < rhs.m;
5
6     return lhs.h < rhs.h;
7 }

```

در اینجا ساعت این دو شی را با هم مقایسه کردیم و در صورت برابر بودن ساعتشان به مقایسه دقیقه شان می‌پردازیم.

همانطور که گفته‌یم باید pair operator ها را جفت پیاده سازی کنیم، بس اینجا < نیز پیاده سازی کنیم که می‌توانیم خروجی آن را به صورت !(lhs < rhs) بنویسیم که چون حالت lhs == rhs را هم در بر می‌گیرد این حالت را حذف کرده و نهایتاً آن را به صورت زیر می‌توانیم بنویسیم:

```

1 public static bool operator >(Time lhs, Time rhs) =>
2     ! (lhs < rhs) && lhs != rhs;

```

operators true/false ۳.۲.۱۷

برای بعدی می توانیم به `true false` اشاره کنیم و بیشتر برای وقتی استفاده می شود که بخواهیم چک کنیم، مثلا :

```

1 if (t9)
2     t9 = t9 + 1;

```

حال اینجا به عنوان مثال

```

1 public static bool operator true(Time t)
2 {
3     return t.h != 0 || t.m != 0;
4 }
5 public static bool operator false(Time t)
6 {
7     if (t)
8         return false;
9     return true;
10 }

```

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 :

```
 1 public class Student
 2 {
 3     public string Name {get; private set;}
 4     public int Id {get; private set;}
 5     public double GPA {get; private set;}
 6
 7     public Student(string name, int id, double gpa)
 8     {
 9         this.Name = name;
10         this.Id = id;
11         this.GPA = gpa;
12     }
13 }
```

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 بدھیم؟

ابتدا اینکار را با نوشتن دو تابع انجام می دهیم . بنابراین یک تابع برای مرتب کردن دانش آموزان بر حسب شماره دانشجویی و دیگری بر حسب معدل می نویسیم.

```

1 private static void IdSort(Student[] students)
2 {
3     for (int i = 0; i < students.Length; i++)
4         for (int j = i + 1; j < students.Length; j++)
5             if (students[i].Id < students[j].Id)
6                 Swap(students, i, j);
7 }
```

نمونه کد ۱۶۸ : تعریف تابع `IdSort`

```

1 private static void GPASort(Student[] students)
2 {
3     for (int i = 0; i < students.Length; i++)
4         for (int j = i + 1; j < students.Length; j++)
5             if (students[i].GPA < students[j].GPA)
6                 Swap(students, i, j);
7 }
```

نمونه کد ۱۶۹ : تعریف تابع `GPASort`

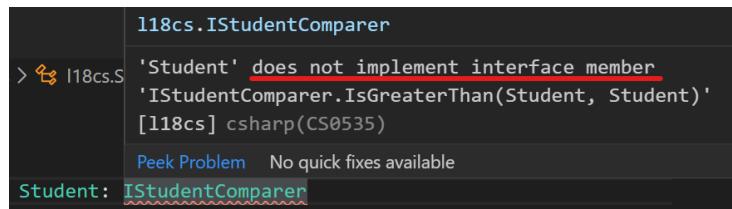
خب همانطور که می بینید این دو تابع تقریباً شبیه هم هستند و فقط در خط هفت با یکدیگر فرق دارند. در اینجا هست که واسط ها `Interfaces` به کمک ما می آیند . واسط ها مزایای زیادی دارند ، مثلاً باعث می شوند که کدها قابلیت بهتری در نگهداری ، انعطاف پذیری و استفاده مجدد داشته باشند . همچنین یک کلاس می تواند همزمان از چند واسط ارث بری کند .

برای حل این چالش ابتدا یک واسطه به نام `IStudentComparer` تعریف میکنیم ، برای تعریف واسط از کلید واژه `Interface` استفاده میکنیم .

```
1 interface IStudentComparer
2 {
3     bool IsGreaterThan(Student s1, Student s2);
4 }
```

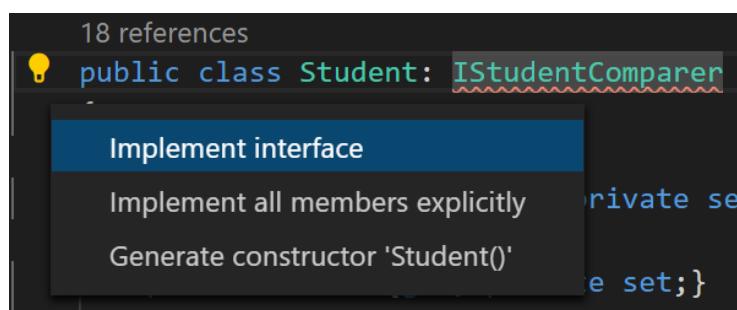
نمونه کد ۱۷۰ : تعریف واسط `IStudentComparer`

سپس در کلاس `Student` از واسطeman ارث بری میکنیم . خب همانطور که میبینید با انجام این کار با یک خطأ روبرو میشویم .



شکل ۱.۱۸ : خطای شماره یک

برای رفع این خطأ از لامپ زردی که در سمت چپ خطی که خطأ را داریم قرار گرفته ، کمک میگیریم و با زدن گرینه `Implement interface` خطأ رفع میشود .



شکل ۲.۱۸ : رفع خطای شماره یک

خب همانطور که می بینید یک تابع به کلاسman اضافه می شود .

```

1 public bool IsGreaterThan(Student other)
2 {
3     throw new System.NotImplementedException();
4 }
```

نمونه کد ۱۷۱ : تعریف تابع IsGreaterThan

اگر دقت کنید ما به یک مشکل بخورد می کنیم . ما برای حل چالشمان می خواستیم 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 واسطه IStudentComparer ارث بری می کنند .

```

1 class StudentIdComparer: IStudentComparer
2 {
3     public bool IsGreaterThan(Student s1, Student s2)
4         => s1.Id < s2.Id;
5 }
```

نمونه کد ۱۷۲ : تعریف کلاس StudentIdComparer

```

1 class StudentGPAComparer: IStudentComparer
2 {
3     public bool IsGreaterThan(Student s1, Student s2)
4         => s1.GPA < s2.GPA;
5 }
```

نمونه کد ۱۷۳ : تعریف کلاس StudentGPAComparer

هم اکنون در کلاس Program ، تابع Sort یک متغیر از جنس IStudentComparer را به عنوان ورودی دریافت می کنیم . سپس از تابع ISGreaterThan واسطه ای استفاده می کنیم .

```

1 private static void Sort(Student[] students, IStudentComparer cmp)
2 {
3     for(int i=0; i<students.Length; i++)
4         for(int j=i+1; j<students.Length; j++)
5             if (cmp.IsGreaterThan(students[i], students[j]))
6                 Swap(students, i, j);
7 }
```

نمونه کد ۱۷۴ : تعریف تابع Sort

برای صدا زدن تابع از دو کلاسی که برای مرتب کردن نوشته ایم ، استفاده می کنیم و یک شیء از کلاس مورد نظر را می سازیم و به تابعمن به عنوان ورودی می دهیم .

```
1 Sort(students, new StudentGPAComparer());
2 Sort(students, new StudentIdComparer());
```

نمونه کد ۱۷۵ : صدا زدن تابع Sort

برای اینکه هر بار یک شیء از کلاس هایمان نسازیم ، می توانیم یک کلاس به نام `StudentComparer` تعریف کنیم و در آن دو ویژگی از جنس کلاس هایمان را تعریف کنیم .

```
1 static class StudentComparer
2 {
3     public static StudentIdComparer StudentIdComparer = new StudentIdComparer();
4     public static StudentGPAComparer StudentGPAComparer = new StudentGPAComparer();
5 }
```

نمونه کد ۱۷۶ : تعریف کلاس StudentComparer

حالا برای اطمینان از اینکه تابعمن درست کار می کند ، آن را تست می کنیم .

```
1 Sort(students, StudentComparer.StudentGPAComparer);
2 PrintStudents(students);
3
4 Sort(students, StudentComparer.StudentIdComparer);
5 PrintStudents(students);
```

نمونه کد ۱۷۷ : تست تابع Sort

```
1 private static void PrintStudents(Student[] students)
2 {
3     foreach(var s in students)
4         Console.WriteLine($" {s.GPA} {s.Id} {s.Name} ");
5 }
```

نمونه کد ۱۷۸ : تعریف تابع PrintStudents

۲.۱۸ چالش ۲

خب ، اگر تست کنید متوجه می شوید که تابع `man` به درستی کار می کند ، حالا فرض کنید که کلاس `Teacher` داریم:

```

1 public class Teacher
2 {
3     string Name;
4     int Id;
5     double Rating;
6
7     public Teacher(string name, int id, double rating)
8     {
9         this.Name = name;
10        this.Id = id;
11        this.Rating = rating;
12    }
13 }
```

نمونه کد ۱۷۹ : تعریف کلاس `Teacher`

فرض کنید که یک آرایه از جنس `Teacher` و یک آرایه دیگر از جنس `Student` داریم و می خواهیم 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 ؟

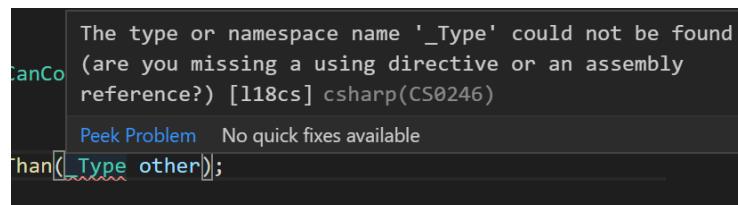
یک واسط به نام `ICanCompare` را تعریف می کنیم .

```

1 public interface ICanCompare
2 {
3     int IsGreaterThan(_Type other);
4 }
```

نمونه کد ۱۸۰ : تعریف واسط `ICanComparer`

همانطور که می بینید به یک خطای بزرگ داریم.



شکل ۳.۱۸: خطای شماره دو

دلیل این خطای این است که هر وقت می خواهیم از `_Type` استفاده کنیم باید کنار نام آن کلاس یا واسط `<_Type>` را اضافه کنیم .

```

1 public interface ICanCompare<_Type>
2 {
3     int IsGreaterThan(_Type other);
4 }
```

نمونه کد ۱۸۱: تعریف واسط `ICanComparer`

خوبیست این خطای رفع شد. الان باید کلاس های `Student` و `Teacher` این واسط را ارث بروی کنند .

```

1 public class Teacher : ICanCompare<Teacher>
2 {
3     string Name;
4     int Id;
5     double Rating;
6
7     public Teacher(string name, int id, double rating)
8     {
9         this.Name = name;
10        this.Id = id;
11        this.Rating = rating;
12    }
13    public int IsGreaterThan(Teacher other) => this.Name.CompareTo(other.Name);
14 }
```

نمونه کد ۱۸۲: تعریف کلاس `Teacher`

```

1  public class Student : ICanCompare<Student>
2  {
3      string Name;
4      int Id;
5      double GPA;
6
7      public Student(string name, int id, double gpa)
8      {
9          this.Name = name;
10         this.Id = id;
11         this.GPA = gpa;
12     }
13     public int IsGreaterThan(Student other) => this.Name.CompareTo(other.Name);
14 }
```

نمونه کد ۱۸۳ : تعریف کلاس Student

اکنون باید تابع Sort را بنویسیم .

```

1  private static void Sort<_Type>(_Type[] students)
2  where _Type:ICanCompare<_Type>
3  {
4      for(int i=0; i<students.Length; i++)
5          for(int j=i+1; j<students.Length; j++)
6              if (students[i].IsGreaterThan(students[j])>0)
7                  Swap(students, i, j);
8  }
```

نمونه کد ۱۸۴ : تعریف تابع Sort

خب حال نوبت به تست کردن تابعمنان می‌رسد .

```

1  Sort(teachers);
2  Sort(students);
```

نمونه کد ۱۸۵ : تست تابع Sort

خوشبختانه تابعمنان 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 می خواهیم شیء هایش 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 باشد ، می توانیم از واسط استفاده کنیم .

مثال ۱ : فرض کنید که می خواهید برای هر یک از اشکال یک کلاس بنویسید . به همین خاطر می توانید یک واسط به نام IShape تعریف کنید و این واسط یه لیستی از جنس رأس بگیرد و تابع هایی برای بدست آوردن محیط و مساحت داشته باشد .

مثال ۲ : فرض کنید که می خواهید 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 می توانیم از دو واسط IDoctor و IPerson استفاده کنیم ، بطوریکه کلاس بیمار از واسط IPerson ارث بری کند و کلاس دکتر از IDoctor و IPerson ارث بری کند .

برای مشاهده مثال های بیشتر به مراجع [۱، ۲] مراجعه کنید .

تعدادی واسط هم خودشان تعریف شده اند و ما می توانیم از آنها ارث بری کنیم . ما در این جلسه با واسط آشنا شدیم و در جلسه بعدی قرار است با واسطه های بیشتری مثل IEnumarator آشنا شویم .

۵.۱۸ تمرینات اضافی

برای تسطیع 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.

۱۹ جلسه

Interface IEnumarable، IDisposable

آزاده دارایی مقدم - ۱۳۹۹/۲/۶

جزوه جلسه ۱۱۹ مورخ ۱۳۹۹/۲/۶ درس برنامه‌سازی پیشرفته تهیه شده توسط آزاده دارایی 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 جزو 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.

Generic Interface ۱.۱۹

یکی از کاربردهای Generic این است که بتوان نوع داده‌ی ساده و پیچیده را مانند عدد، رشته و ... را به عنوان یک پارامتر به متدها، کلاس‌ها و اینترفیس اضافه کرد. برای نوشتن اینترفیس عمومی بهتر است برای

مجموعه کلاس های عمومی یک اینترفیس تعریف کنیم.

```

1 internal interface IShape
2 {
3     void Draw();
4     double GetArea();
5 }
```

نمونه کد ۱۸۶ : اینترفیس عمومی در سی شارپ

```

1 internal class Circle: IShape
2 {
3     private Point point;
4     private int v;
5     public double GetArea() => this.v * this.v * Math.PI;
6     public void Draw()
7     {
8         Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
9     }
10    public Circle(Point point, int v)
11    {
12        this.point = point;
13        this.v = v;
14    }
15 }
```

نمونه کد ۱۸۷ : کلاس circle

```

1 internal class Triangle: IShape
2 {
3     private Point point1;
4     private Point point2;
5     private Point point3;
6     public double GetArea()
7     {
8         return (this.point1.X*(this.point2.Y - this.point3.Y)
9             + this.point2.X*(this.point3.Y - this.point1.Y)
10            + this.point3.X*(this.point1.Y - this.point2.Y))/2;
11     }
12     public void Draw()
13     {
14         Console.WriteLine(" Drawing Triangle at {point1} ,{point2} ,{point3} ");
15     }
16     public Triangle(Point point1, Point point2, Point point3)
17     {
18         this.point1 = point1;
19         this.point2 = point2;
20         this.point3 = point3;
21     }
22 }
```

نمونه کد ۱۸۸ : کلاس Triangle

```

1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  class Program
5  {
6      static void Main(string[] args)
7      {
8          Triangle t = new Triangle(new Point(2,4),
9              new Point(3, -6),
10             new Point(7, 8));
11            Circle c = new Circle(new Point(5, 4), 4);
12            DrawShapesWithStats(t);
13            DrawShapesWithStats(c);
14        }
15        static void DrawShapesWithStats(IShape shape)
16        {
17            shape.Draw();
18            Console.WriteLine(shape.GetArea());
19        }
20    }

```

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

```

Generic Constraints ۲.۱۹

گاهی وقت ها، نیاز داریم که زمان ساخت شی، تنها نوع داده ای که از نوع Type Value و یا فقط reference باشد را قبول کند. برای اینکار، می توانیم از محدودیت ها(Constraint) استفاده کنیم. برای استفاده از از کلمه where استفاده می کنیم.

محدودیت new() مشخص می کند که یک آرگمن 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:

```

1 static void DrawShapesWithStats<T>(T shape) where T: IShape, new()
2 {
3     T s1 = new T();
4     shape.Draw();
5     Console.WriteLine(shape.GetArea());
6 }
```

نمونه کد ۱۹۰ : Generic Constraints

IDisposable ۳.۱۹

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 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 ای اینترفیس IDisposable این اینترفیس متدی با نام Dispose دارد که به شکل () دارد که به شکل صدا زده می شود و بعد از پیاده از این سازی می توان دستورات جهت پاک سازی منابع را داخل آن نوشت.

به عنوان مثال در کلاس Circle داریم:

```

1 internal class Circle: IShape, IDisposable
2 {
3     private Point point;
4     private int v;
5     public void Dispose()
6     {
7         Console.WriteLine(" Clearing Circle ");
8     }
9     public double GetArea()
10    {
11        return this.v * this.v * Math.PI;
12    }
13    public void Draw()
14    {
15        Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
16    }
17    public Circle(){}
18    public Circle(Point point, int v)
19    {
20        this.point = point;
21        this.v = v;
22    }
23 }
```

نمونه کد ۱۹۱ : main

StreamReader Class ۴.۱۹

برای خواندن فایل های متنی (txt) از کلاس StreamReader استفاده می کنیم. اغلب با استفاده از جمله using نوشته می شود. این ساختار (using) به پاکسازی منابع سیستم کمک می کند.

```

1 static void Main()
2 {
3     string line;
4     using (StreamReader reader = new StreamReader("file.txt"))
5     {
6         line = reader.ReadLine();
7     }
8     Console.WriteLine(line);
9 }
```

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.

```

1 using System;
2 using System.Diagnostics;
3 internal class Timer: IDisposable
4 {
5     private Stopwatch s = new Stopwatch();
6     private string Name;
7     public Timer(string name)
8     {
9         this.Name = name;
10        s.Start();
11    }
12    public void Dispose()
13    {
14        s.Stop();
15        Console.WriteLine(" Elapsed Time({this.Name}): {s.Elapsed.ToString()}");
16    }
17 }
```

نمونه کد ۱۹۳: کلاس Timer

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         using (Timer t = new Timer("ReadAllLines"))
6         {
7             string[] lines = File.ReadAllLines("file.txt");
8         }
9         using (Timer t2 = new Timer("StreamReader"))
10        {
11             StreamReader reader;
12             using (reader = new StreamReader("file.txt"))
13             {
14                 string line;
15                 while (null != (line = reader.ReadLine()))
16                 {
17                     if (line.Length > 5)
18                         break;
19                 }
20             }
21         }
22     }
23 }
```

نمونه کد ۱۹۴: اندازه‌گیری زمان اجرای برنامه خواندن فایل

```

Elapsed Time(ReadAllLines): 00:00:00.1567477
Elapsed Time(StreamReader): 00:00:00.0001820
```

IEnumerator ۶.۱۹

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 می‌کند که کجا رشتہ قرار دارد و بعد از آن مقدار مورد جاری را بنا به درخواست باز می‌گرداند.

اینترفیس IEnumerable یک کلاس توسط یک پیاده سازی می‌شود.Enumerable یک نوع است که یک متد به نام GetEnumerator دارد که این متد یک enumerator بر می‌گرداند. مقدار جایگاه جاری در رشتہ را بر می‌گرداند. Current

MoveNext متدی است که enumerator را به موقعیت بعدی مکانی پیش می‌برد. و یک مقدار بولین بر می‌گرداند که نشان دهنده معتبر بودن موقعیت بعدی و یا انتهای رشتہ می‌باشد.

موقعیت ابتدایی enumerator ، قبل از اولین مورد در رشتہ است. پس تابع MoveNext باید قبل از اولین دسترسی به Current فراخوانی شود.

Yield یک عنصر مجموعه را بر می‌گرداند و موقعیت مکان نما را به عنصر بعدی هدایت می‌کند.

در نمونه کد زیر کاربردی از این اینترفیس را میبینم.

```

1  using System.Collections.Generic;
2  public class PowersOf2
3  {
4      static void Main()
5      {
6          foreach (int i in Power(2, 8))
7          {
8              Console.Write(" {0} ", i);
9          }
10     }
11     public static IEnumerable<int> Power(int number, int exponent)
12     {
13         int result = 1;
14
15         for (int i = 0; i < exponent; i++)
16         {
17             result = result * number;
18             yield return result;
19         }
20     }
21 }
```

نمونه کد ۱۹۵ : کاربرد yield در اینترفیس IEnumerable

Output: 2 4 8 16 32 64 128 256

```

1  using System;
2  using System.Collections.Generic;
3  class Program
4  {
5      static void Main()
6      {
7          List<int> list = new List<int>();
8          list.Add(1);
9          list.Add(5);
10         list.Add(9);
11         List<int>.Enumerator e = list.GetEnumerator();
12         Write(e);
13     }
14     static void Write(IEnumerator<int> e)
15     {
16         while (e.MoveNext())
17         {
18             int value = e.Current;
19             Console.WriteLine(value);
20         }
21     }
22 }
```

نمونه کد ۱۹۶ : کاربرد GetEnumerator در اینترفیس IEnumerable

۲۰۱

INTERFACE IENUMERABLE, IDISPOSABLE . ۱۹ جلسه

Outut: 1 5 9

[DotNetPerls] [csharpdocumentation] [sharp ۱۰۰ microsoft]

جلسه ۲۰

چگونگی کارکرد مموری

سعید شهیب زاده - ۱۳۹۹/۲/۸

جزوه جلسه ۲۰ ام مورخ ۱۳۹۹/۲/۸ درس برنامه‌سازی پیشرفته تهیه شده توسط سعید شهیب زاده. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

Memorymanager Example ۱.۲۰

در این جلسه مثالی درباره چگونگی کارکرد مموری در سیستم عامل زده شد که در مورد آن صحبت می‌شود.

در ابتدا کلاسی به نام Memorymanager تعریف می‌کنیم که دارای ویژگی‌های زیر است:

```

1  public class Memorymanager
2  {
3      private int size;
4      private int[] Memory;
5      private int FirstFree;
6      private const Int16 NoMoreValues = Int16.MaxValue;
7      public MemoryManager(int size)
8      {
9          this.size = size;
10         this.Memory = new int[size];
11         this.FirstFree = 0;
12         this.Memory[0] = (int) new IntBlock((Int16)size, NoMoreValues);
13     }
14 }
```

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 آدرس 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:

```

1 internal struct IntBlock
2 {
3     private Int32 @value;
4
5     public Int16 size {
6         get => SplitNumbers(@value).size;
7         set => CombineNumbers(value, this.next);
8     }
9
10    public override string ToString() => ${, ${next}}""${size}";
11
12    public Int16 next
13    {
14        get => SplitNumbers(@value).next;
15        set => CombineNumbers(this.size, value);
16    }
17
18    public static explicit operator int(IntBlock b) => b.value;
19
20    public static implicit operator IntBlock(int b) => new IntBlock(b);
21
22    public IntBlock(Int32 num) => this.value = num;
23
24    public IntBlock(Int16 size, Int16 next)
25    {
26        @value = CombineNumbers(size, next);
27    }
28    private static int CombineNumbers(Int16 size, Int16 next)
29    {
30        int result = size;
31        result = result << 16;
32        result = result | (UInt16)next;
33        return result;
34    }
35
36    private static (Int16 size, Int16 next) SplitNumbers(int num)
37    {
38        int size = num;
39        size = size >> 16;
40        int next = ((int)Math.Pow(2, 16) - 1) & num;
41        return (Int16)size, (Int16)next;
42    }
43}

```

نمونه کد ۱۹۸: کلاس IntBlock

متدهای اینترفیس 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 آدرس بلاک خالی بعدی است در یک عدد اینتجر هست که بتوانیم در یک خانه ارایه مموری دو عدد داشته باشیم.

متدهای SplitNumbers و CombineNumbers هستند و میتوانند یک عدد اینتجر ۳۲ بیتی را به دو عدد درون یک تاپل برگردانند.

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:

```

1 class MemoryBlock
2 {
3     private int[] Data;
4     public int Start;
5     public int End;
6
7     public override string ToString()
8     {
9         return $"{this.Start}-{this.End}";
10    }
11
12    public int Size => End - Start + 1;
13    public MemoryBlock(int[] data, int start, int end) =>
14        (Data, Start, End) = (data, start, end);
15 }
```

نمونه کد ۱۹۹: کلاس MemoryBlock

حالا که کلاس های Memoryman و IntBlock را دیدیم میتوانیم به سراغ کلاس MemoryBlock برویم و متدهای آن را بررسی کنیم.

این کلاس یک متدهای AcquireMemory دارد که در واقع باگرفتن یک سایز، یک بلاک از مموری با همان سایز را تحويل میدهد.

میتوانیم این متدهای را ببینیم:

```

1 public MemoryBlock AcquireMemory(int size)
2 {
3     int loc = FirstAvailable(size);
4     return new MemoryBlock(data:this.Memory, start:loc, end:loc+size-1);
5 }
```

نمونه کد ۲۰۰: متدهای AcquireMemory

علاوه بر تخصیص بلاک در مموری یک متدهای حذف بلاک هایی که نمیخواهیم هم نیاز داریم این متدهای DeleteMemory نامگذاری میکنیم و پیاده سازی آن به صورت زیر است:

```

1  public void DeleteMemory(MemoryBlock mb)
2  {
3      var current = this.FirstFree;
4      while (true)
5      {
6          IntBlock bi = this.Memory[current];
7          if (bi.next == NoMoreValues)
8          {
9              this.Memory[current] = (int) new IntBlock(bi.size, (Int16) mb.Start);
10             this.Memory[mb.Start] = (int) new IntBlock((Int16) mb.Size, NoMoreValues);
11             break;
12         }
13     }
14 }
```

نمونه کد ۲۰۱: متدهای DeleteMemory

این کلاس همچنین متدهای SplitNumbers و CombineNumbers را دارد که نحوه پیاده سازی آنها را پیشتر دیدم.

ما همچنین به یک متدهای خالی مموری نیاز داریم ، این متدهای FreeBlocks نام دارد و پیاده سازی آن به صورت زیر است.

```

1  public IEnumerable<IntBlock> FreeBlocks()
2  {
3      int current = this.FirstFree;
4      while (current != NoMoreValues)
5      {
6          IntBlock cb = this.Memory[current];
7          yield return cb;
8          current = cb.next;
9      }
10 }
```

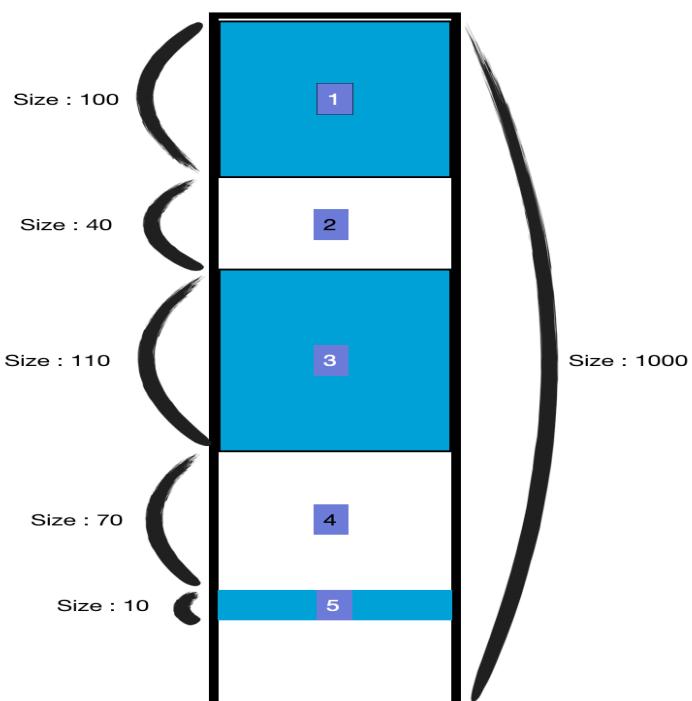
نمونه کد ۲۰۲: متدهای FreeBlocks

این متدهای `IEnumerable` است به این معنی که میتوانیم بر روی آن `foreach` بزنیم و عناصر را یکی با یکی با دستور `yield return` برگردانیم.

دستور `yield return` به این صورت عمل میکند که درون حلقه قرار میگیرد و وقتی متدهای آن رسید یک

عنصر از آن ارایه را برمیگرداند و از متدهای خارج میشود و بعد دوباره به متدهای برمیگردید از ادامه دستور `return` شروع میکند تا دوباره به آن برسد و عنصر بعدی را برمیگرداند

متدهایی که کلاس `MemoryManager` داراست متدهای `FirstAvailable` است که به عنوان ورودی یک سایز میگیرید و در مموری اولین بلاک خالی که ظرفیت لازم یا همان سایز مورد نظر را دارد پیدا میکند و آدرس آن را برمیگرداند



برای مثال در تصویر بالا اگر یک بلاک با اندازه ۵۰ بخواهیم متدهای `FirstAvailable` ادرس بلاک ۱۴ که عدد ۲۵۰ است را برمیگرداند

حالا می توانیم نحوه پیاده سازی متده را ببینیم :

```

1  public int FirstAvailable(int size)
2  {
3      int current = this.FirstFree;
4      int last = this.FirstFree;
5      int location = NoMoreValues;
6      while(current != NoMoreValues)
7      {
8          IntBlock cb = this.Memory[current];
9          IntBlock lb = this.Memory[last];
10         if (cb.size >= size)
11         {
12             location = current;
13             if (last == current)
14                 this.FirstFree = current + size;
15             else
16                 this.Memory[last] = (int) new IntBlock( lb.size,
17                     cb.size > size ?
18                         (Int16)(current + size) :
19                         cb.next);
20
21             if (cb.size > size)
22                 this.Memory[current + size] = (int) new IntBlock(
23                     (Int16)(cb.size - size), cb.next
24                 );
25
26             break;
27         }
28         last = current;
29         current = cb.next;
30     }
31
32     if (current == NoMoreValues)
33         throw new InvalidOperationException();
34
35     return location;
36 }
```

نمونه کد ۲۰۳ : متده FirstAvailable

و در آخر یک متده برای چاپ کردن بلاک های خالی در کنسول داریم که میتوانیم پیاده سازی آن را

مشاهده کنیم:

```

1  public void PrintBlocks()
2  {
3      foreach(var info in this.FreeBlocks())
4          Console.WriteLine(info);
5  }
```

نمونه کد ۲۰۴ : متده PrintBlocks

حالا که با نحوه کار مموری اشنا شدیم میتوانیم استفاده از آن را هم ببینیم برای مثال یک مموری با اندازه ۱۰۰۰ درست کرده و از متدهای AcquireMemory و DeleteMemory استفاده میکنیم و بلاک هایی از انرا اشغال میکنیم و با استفاده از PrintBlocks میبینیم که به درستی کار میکند:

```

1 static void Main(string[] args)
2 {
3     MemoryManager mm = new MemoryManager(size:100);
4     mm.PrintBlocks();
5
6     var m10 = mm.AcquireMemory(10);
7     mm.PrintBlocks();
8
9     var m20 = mm.AcquireMemory(85);
10    mm.PrintBlocks();
11
12    mm.DeleteMemory(m10);
13    mm.PrintBlocks();
14
15    var m3 = mm.AcquireMemory(8);
16    mm.PrintBlocks();
17
18    IntBlock b = new IntBlock(5, 10);
19    int[] memory = new int[10];
20    memory[5] = (int) b;
21 }
```

نمونه کد ۲۰۵: تابع Main

Output:

(۱۰۰,۳۲۷۶۷)

(۹۰,۳۲۷۶۷)

(۵,۳۲۷۶۷)

(۵,۰)

(۱۰,۳۲۷۶۷)

(۵,۸)

(۱۰,۳۲۷۶۷) (عدد ۳۲۷۶۷ به معنی آن است که بلاکی بعد از آن بلاک وجود ندارد)

جلسه ۲۰. چگونگی کارکرد مموری

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

۲۱ جلسه

اشاره گر به تابع

مهدیه نادری - ۱۳۹۸/۲/۱۳

۱.۲۱ اشاره گر به تابع* در زبان پایتون

همان طور که 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*

```

1  def add(a, b):
2      return a+b
3
4  def mul(a, b):
5      return a*b
6
7  def sub(a, b):
8      return a-b
9
10 def combine(list_a, list_b, fn):
11     list_result = []
12     for i in range(0, len(list_a)):
13         result = fn(list_a[i], list_b[i])
14         list_result.append(result)
15     return list_result
16
17 def print_pretty(list):
18     for i in range(0, len(list)):
19         print(f"[{i}]=[{list[i]}]")
20
21 def main():
22     a = [1, 2, 3, 4, 5]
23     b = [2, 3, 4, 5, 7]
24     c = combine(a, b, add)
25     print_pretty(c)
26
27 if __name__ == "__main__":
28     main()

```

نمونه کد ۲۰۶: ارسال یک تابع به عنوان ورودی تابع دیگر در پایتون

به این کد تابع دیگری مثل negative() به کد اضافه می‌کنیم و در main به جای add در تابع combine از آن استفاده می‌کنیم . در پایتون negative() را با هر تعداد ورودی صدا بزنیم، پردازشگر ایرادی نمیگیرد ولی موقع اجرای برنامه با خطا ی کامپایل روبرو می‌شویم. بررسی نکردن نوع متغیرها از ویژگی‌های این زبان است و در زمان run time اگر درست نبود خطای میدهد و در غیر این صورت اجرا میکند.

```

1  def negative(a):
2      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 متغیرهای ورودی و نوع آنها و نوع خروجی مشخص شود. استفاده از کلید واژه‌ی

جلسه ۲۱. اشاره گر به تابع

۲۱۳

delegate خصامت میکند آنچه در ادامه می آید ، یک نوع تابع است نه خود تابع و کسی نمیتواند تابعی از نوع دیگر به تابع ثانی بدهد.

```
1  using System;
2  using System.Diagnostics;
3  using System.IO;
4
5  namespace c14cs
6  {
7      class Program
8      {
9          delegate int binary_op_int(int a,int b);
10         static int mul(int a, int b) => a*b;
11         static int add(int a, int b) => a+b;
12         static int[] combine(int[] a, int[] b, binary_op_int fn)
13         {
14             int[] result = new int[a.Length];
15             for (int i = 0; i < result.Length; i++)
16             {
17                 result[i] = fn(a[i], b[i]);
18             }
19             return result;
20         }
21         static void Main(string[] args)
22         {
23             int[] list_a = new int[] {3, 2, 3, 1, 5};
24             int[] list_b = new int[] {1, -2, 2, 4, 5};
25             var c = combine(list_a, list_b, add);
26         }
27     }
28 }
```

```
1  using System;
2  using System.Diagnostics.CodeAnalysis;
3
4  namespace l21cs
5  {
6      public class Student
7      {
8          public int id;
9          public double GPA;
10
11         public Student(int id, double GPA)
12         {
13             this.id = id;
14             this.GPA = GPA;
15         }
16     }
17 }
```

در کد بالا پیاده سازی همان مثال پایتون را در سی شارپ مشاهده می کنید. برای مرتب کردن آرایه ها

تابع جدیدی به نام Sort() می سازیم.

```

1 static void Sort(int[] list)
2 {
3     for (int i = 0; i < list.Length; i++)
4     {
5         for (int j = i+1; j < list.Length; j++)
6         {
7             if(list[i] < list[j])
8                 (list[i], list[j]) = (list[j], list[i]);
9         }
10    }
11 }
```

یک روش برای Swap() هست که با جابجایی دو قسمت (list[i], list[j]) = (list[j], list[i]) 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 بنویسیم. حال برای اینکه تابع Sort() برای آن بنویسیم باید :

```

1 static void Sort(Student[] list, IStudentComparer cmp)
2 {
3     for (int i = 0; i < list.Length; i++)
4     {
5         for (int j = i+1; j < list.Length; j++)
6         {
7             if(cmp.IsGreater(list[i], list[j]))
8                 (list[i], list[j]) = (list[j], list[i]);
9         }
10    }
11 }
```

اینترفیس IStudentComparer پیاده سازی بشد که میخواهیم برای مقایسه دو Student تابعی داشته باشد. و از آن در کلاس ها استفاده کنیم. برای پیاده سازی آن میتوانیم از کلاس جدیدی در Main استفاده کنیم. و کلاس StudentComparer که اینترفیس مورد نظر را دارد، بنویسیم. به طور دلخواه ویژگی مورد مقایسه در GPA را IsGreater() 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])

```

1  namespace l21cs
2  {
3      internal interface IStudentComparer
4      {
5          bool IsGreater(Student s1, Student s2);
6      }
7 }
```

```

1  namespace l21cs
2  {
3      public class StudentComparer : IStudentComparer<Student>
4      {
5          public bool IsGreater(Student s1, Student s2)
6          {
7              return s1.GPA > s2.GPA;
8          }
9      }
10 }
```

اگر بخواهیم Sort() برای هر نوع داده‌ای، اجرا شود باید تغییراتی در کدهای قبل اعمال کنیم.(نمونه کد ۲۳۰.)

```

1  static void sort<Type>(Type[] list, IStudentComparer<Type> cmp)
```

```

1  internal interface IStudentComparer<Type>
2  {
3      bool IsGreater(Type s1, Type s2);
4  }
```

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.

```

1  delegate bool student_comparer(Student s1, Student s2);
```

```

1 static void sort<_Type>(Student[] list, student_comparer stdcmp)
2 {
3     for(int i=0; i<list.Length; i++)
4         for(int j=i+1; j<list.Length; j++)
5             if (stdcmp(list[i], list[j]))
6                 (list[i], list[j]) = (list[j], list[i]);
7 }
```

دو تابع جدید برای مقایسه‌ی اعضای مجموعه نوشته‌ایم، یکی بر اساس GPA و دیگری بر اساس Id.

```

1 static bool StdCmpGPA(Student s1, Student s2) => s1.GPA > s2.GPA;
2 static bool StdCmpId(Student s1, Student s2) => s1.id > s2.id;
```

وقتی دو تابع همنام باشند، یکی از آنها overload های Sort() 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 مقایسه‌ی اینکه نوشتن این برنامه با اینترفیسی بیشتر است یا دلگیت می‌توان اشاره کرد که اگه اینترفیسی لازم داشتیم که چند متود داشت، استفاده از دلگیت خیلی جالب نبود. برای دیدن نتیجه مرتب کردن لیست، لازم است تابعی برای چاپ کردن بنویسیم.

```

1 private static void print<_Type>(_Type[] stdlist)
2 {
3     Console.WriteLine("-----");
4     foreach(var s in stdlist)
5         Console.WriteLine(s);
6 }
7 );
```

برای بهتر نوشتن و مدیریت آن در کلاس Student تابع ToString() را Override() می‌کنیم.

```

1 public override string ToString() => $"{id} - {GPA};
```

تتابع را صدا می‌زنیم و برنامه را اجرا می‌کنیم.

```

1 static void Main(string[] args)
2 {
3     Student[] stdlist = new Student[]{
4         new Student(98521234, 8.12),
5         new Student(97532412, 8.14),
6         new Student(98234324, 8.13),
7         new Student(97989899, 8.11),
8     };
9
10    sort(stdlist, new StudentComparer());
11    sort(stdlist, new StudentComparer());
12    ^~I print(stdlist);
13    sort(stdlist, new StudentComparer().IsGreater);
14    print(stdlist);
15    sort(stdlist, StdCmpGPA);
16    print(stdlist);
17    sort(stdlist, StdCmpId);
18    print(stdlist);
19    ^~I}

```

بعد از چاپ کردن معلوم نمی شود همه ی توابع درست کار میکنند یا نه. چون یکبار لیست موردنظر مرتب شده و هر بار 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 بر میگرداند تصادفی باشد. و با ویژگی های GPA و Id ارتباطی نداشته باشد.

```

1 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 بودن این عدد، ۵۰ درصد است و کاملاً تصادفی است. ۵۰٪

```

1   Student[] stdlist = new Student[]{
2       new Student(98521234, 8.12),
3       new Student(97532412, 8.14),
4       new Student(98234324, 8.13),
5       new Student(97989899, 8.11),
6   };
7
8   sort(stdlist, new StudentComparer());
9   print(stdlist);
10  sort(stdlist, RndCmp);
11
12  sort(stdlist, new StudentComparer().IsGreater());
13  print(stdlist);
14  sort(stdlist, RndCmp);
15
16  sort(stdlist, StdCmpGPA);
17  print(stdlist);
18  sort(stdlist, RndCmp);
19
20  sort(stdlist, StdCmpId);
21  print(stdlist);

```

همانطور که می بینید توابع صدای زده شده در () Main هم از الگوی خاصی میکند. یک لیست مشخص را مرتب میکنند، آن را چاپ کرده و بعد دوباره بهم میریزند. البته با یک تغییر کوچک میتوان گفت طرح اصلی آن است که اول بهم بریزد بعد سورت و سپس چاپ کند. درنتیجه میتوانیم تابعی بنویسیم که این سه کار را پشت سر هم انجام دهد. پس تابع () RunSortExperiment را طوری مینویسیم که یک String برای عنوان کاری که انجام میدهد ، یک لیست و آبجکتی از کلاس StudentComparer و دو تابع برای Sort() و print() به عنوان ورودی میگیرد. و delegate های زیر

```

1 delegate void sort_delegate<_Type>(_Type[] list, student_comparer stdcmp);
2 delegate void print_delegate<_Type>(_Type[] stdlist);

```

را برای آن تعریف میکنیم.

```

1 static void RunSortExperiment<_Type>(
2     string label,
3     _Type[] list,
4     sort_delegate<_Type> sortfn,
5     print_delegate<_Type[]> printfn,
6     StudentComparer cmp)
7 {
8     sortfn(list, RndCmp);
9     sortfn(list, cmp);
10    Console.WriteLine(label);
11    printfn(list);
12 }

```

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 برای تعریف یک تابع لازم است را باید نوشت، البته می توان آن را خلاصه کرد و اگر متغیر هایی را از قبل تعریف کردیم میتوانیم دیگر نوع آنها را در زمان استفاده در تابع ننویسیم. لازم نیست نوع متغیر های اطراف را وارد کنید و هر وقت بخواهید قابل دسترس هستند. مثلاً زمان صدا زدن تابع Sort() بنویسیم:

```

1 sort(stdlist, (s1, s2) => { return s1.GPA > s2.GPA; });

```

یا بطور خلاصه تر

```

1 sort(stdlist, (s1, s2) => s1.GPA > s2.GPA );

```

که این روش همان استفاده از lambda expression است. مثلاً در متود Find() در لیست ها که در واقع یک delegate است دارد: این تابع به عنوان ورودی یک Student میگیرد و یک predicate برمیگرداند. به ترتیب اعضای لیست را با عدد داده شده مقایسه میکند و اگر برابر شد true برمیگرداند.

```

1 List<Student> std_list = new List<Student>(stdlist);
2 var s97532412 = std_list.Find( Student s ) => {
3     return s.id == 97532412;
4 };

```

در فریم ورک دات نت 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 Stu-Student, Student, bool> stdcmp dentComparer ، 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 نگاهی بیندازیم، از نظر تعداد ورودی و خروجی شباهت هایی دارند که سبب شده بتوانیم به صورت Generic تعریف شان کنیم. و برخی از آن هارا حذف کنیم.

```

1  delegate int binary_op_int(int a, int b);
2  delegate void sort_delegate<_Type>(_Type[] list, Func<_Type, _Type, bool> stdcmp);
3  delegate void print_delegate<_Type>(_Type[] stdlist);
```

حال میتوانیم تابع RunSortExperiment() را بازنویسی کنیم.

```

1  static void RunSortExperiment<_Type>(
2      string label,
3      _Type[] list,
4      sort_delegate<_Type> sortfn,
5      Action<_Type[]> printfn,
6      Func<_Type, _Type, bool> cmp)
7  {
8      sortfn(list, RndCmp);
9      sortfn(list, cmp);
10     Console.WriteLine(label);
11     printfn(list);
12 }
```

و در Main() تغییرات لازم را ایجاد نماییم.

```

1  RunSortExperiment("InterfaceMethod",
2      stdlist,
3      sort,
4      print,
5      new StudentComparer().IsGreater);
6  RunSortExperiment("StdCmpGPA", stdlist, sort, print, StdCmpGPA);
7  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 سی‌پلاس‌پلاس
- Multi-Threading •

c++ در Closure ۱.۲۳

ابدا وکتوری از اعداد صحیح را تشکیل می‌دهیم. سپس از فانکشن `for_each` موجود در کتابخانه `algorithm` استفاده می‌کنیم. (نمونه کد ۲۰۷)

```

1 #include <vector>
2 #include <iostream>
3 #include <algorithm>
4
5 using namespace std;
6
7 int main(int argc, char const *argv[])
8 {
9     vector<int> vNums {1, 2, 5, 3, 4, 1};
10
11    for_each(vNums.begin(), vNums.end(), [] (int n) {
12        cout << n << endl;
13
14    return 0;
15 }
```

نمونه کد ۲۰۷ : تعریف وکتور و استفاده از `for_each`

همان طور 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 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)

```

1 int main(int argc, char const *argv[])
2 {
3     vector<int> vNums {1, 2, 5, 3, 4, 1};
4     vector<int>::iterator it;
5
6     for(it = vNums.begin(); it != vNums.end(); it++)
7     {
8         cout << *it << endl;
9     }
10
11     return 0;
12 }
```

نمونه کد ۲۰۸ : تعریف iterator برای وکتور for_each

توضیح: در این روش، ابتدا یک iterator برای پیمایش در طول وکتورمان تعریف می‌کنیم. سپس با استفاده از آن یک حلقه‌ی فور نوشه و مقدار اولیه‌ی آن را 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 حلقه‌ی 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 MoveNext() در زبان سی‌شارپ است. و عملکرد i++ همانند() در زبان IEnumarable سی‌شارپ است.

*حال برای مطرح کردن نکته‌ی جدیدی به (نمونه کد ۲۰۷) باز می‌گردیم. با نگاه کردن به سومین ورودی در می‌بایس که یک expression lambda for_each 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 offset 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)

```

1 int main(int argc, char const *argv[])
2 {
3     vector<int> vNums {1, 2, 5, 3, 4, 1};
4     int offset = 10;
5
6     for_each(vNums.begin(), vNums.end(), [] (int n) {
7         cout << n + offset << endl;
8     });
9
10    return 0;
11 }
```

نمونه کد ۲۰۹ : استفاده از متغیر محلی در lambda expression

در این مرحله ما با ارور زیر مواجه می‌شویم:

```

int main(int argc, char const *argv[])
{
    vector<int> vNums;
    int offset = 10;
    for_each(vNums.begin(), vNums.end(), [] (int n) {
        cout << n + offset << endl;
    });
    return 0;
}
```

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 در واقع ورودی سوم تعريفی 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)

```

1 int main(int argc, char const *argv[])
2 {
3     vector<int> vNums {1, 2, 5, 3, 4, 1};
4     int offset = 10;
5
6     for_each(vNums.begin(), vNums.end(), [offset] (int n) {
7         cout << n + offset << endl;
8     });
9
10    return 0;
11 }
```

C++ Clause Capture نمونه کد ۲۱۰ : استفاده از

نکته: برای اینکه مشخص کنیم که متغیرهای داخل Capture Clause به صورت pass by value یا 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 می‌خواهیم تمام متغیرهای محلی‌مان pass by value شوند، کافیست داخل براکت فقط یکبار علامت `=` را بگذاریم و در این صورت نیازی به نوشتندن تک تک متغیرها نیست.
- اگر می‌خواهیم متغیرمان pass by reference شود، کافیست جلوی نام آن علامت `&` در Capture Clause گذاشته شود. و اگر می‌خواهیم تمام متغیرهای محلی‌مان pass by reference شوند، کافیست داخل براکت فقط یکبار علامت `&` را بگذاریم و در این صورت نیازی به نوشتندن تک تک متغیرها نیست.

برای مشاهدهٔ آنچه گفته شد، نمونه کد زیر را ببینید.

```

1 int main(int argc, char const *argv[])
2 {
3     vector<int> vNums {1, 2, 5, 3, 4, 1};
4     int offset = 10;
5     int sum = 0;
6     offset : pass by value, sum : pass by reference
7
8     for_each(vNums.begin(), vNums.end(), [offset, &sum] (int n) {
9         cout << n + offset << endl;
10        sum += n;
11    });
12
13     offset and sum : pass by value
14     for_each(vNums.begin(), vNums.end(), [=] (int n) {
15         cout << n + offset << endl;
16         sum += n;
17    });
18
19     offset : pass by reference, sum : pass by value
20     for_each(vNums.begin(), vNums.end(), [&offset, sum] (int n) {
21         cout << n + offset << endl;
22         sum += n;
23    });
24
25     offset and sum : pass by reference
26     for_each(vNums.begin(), vNums.end(), [&] (int n) {
27         cout << n + offset << endl;
28         sum += n;
29    });
30
31     return 0;
32 }
```

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 آبحکت ساخته می‌شود. که این آبحکت 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 سایت‌ها رجوع کنید. [\[moreabout\]](#) [\[cppclosure\]](#)

Local Variables*

c# در Closure ۲.۲۳

برای آشنا شدن با این مفهوم، ابتدا لیستی از اکشن را تعریف می‌کنیم؛ سپس یک حلقه فور می‌نویسیم و داخل آن lambda 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 closure lambda expression خود می‌توانیم از متغیر *i* استفاده کنیم، مفهوم closure است که در زبان سی‌شارپ، پنهان و نهفته است. در واقع چون *i* یک متغیر محلی نزدیک به فانکشن ما می‌باشد، استفاده از آن برای ما ممکن شده است.

نکته: نکته‌ی حائز اهمیت در این قسمت این است که در حلقه *i* فور می‌کدی اجرا نمی‌شود؛ بلکه در این حلقه صرفاً اکشن‌هایی تعریف و به لیست اضافه می‌گردند. قسمتی که این اکشن‌ها شروع به اجرا شدن می‌کنند در حلقه *i* فوراً می‌باشد.

نکته‌ی جالب: با اجرا شدن Main برنامه، فکر می‌کنیم که اعداد ۰ تا ۹ هر کدام در خطی جداگانه در خروجی چاپ می‌شوند. در حالی که کاملاً اشتباه کرده ایم و خروجی ۱۰ بار عدد ۱۰ را چاپ می‌کند.

علت: علت این اتفاق، همان نکته‌ی است که در بالا گفته شد. زیرا اسکوپ مربوط به *i* مربوط به کل حلقه می‌باشد. پس ما در واقع فقط یک *i* داریم. که این *i* pass by reference می‌شود. در واقع با اجرای Main ابتدا حلقه می‌فور 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 اکشن‌ها شروع به اجرا می‌کنند ما فقط یک متغیر *i* داریم که مقدار آن ۱۰ می‌باشد. بنابراین عدد 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 نرا داخل آبجکت قرار می‌دهد.

حال می‌خواهیم برای جا افتادن مفهوم تغییری را در کد ایجاد کنیم: این بار، در حلقه‌ی فور تغییری ایجاد می‌کنیم که بعد از چاپ نیکی 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 نیکی زیاد می‌شود. و به اکشن بعدی درون لیست funcs، i با مقدار ۱۱ داده می‌شود. سپس این عدد نیز چاپ می‌شود و دوباره مقدار نیکی زیاد می‌شود و به اکشن بعدی داخل لیست، نیکی با مقدار ۱۲ داده می‌شود و به همین ترتیب تا عدد ۱۹ در خروجی چاپ می‌شود.

Multi-Threading ۳.۲۳

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 طور که می بینید، وقتی Main برنامه اجرا می شود، حلقه‌ی فور شروع به اجرا شدن می کند و ابتدا عدد ۰ چاپ می شود، سپس ۲۰۰ میلی ثانیه صبر می کند 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 می شود. 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 بقیه‌ی برنامه انجام می شود. و در این حین ما نمی توانیم کاری انجام بدھیم.

خروجی کد بالا طبق چیزی که گفته شد، به صورت زیر می باشد:

```

0
1
2
3
4
5
Doing-other-things
```

DownloadAsyncSimple ۱.۳.۲۳

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 سایت‌ها به صورت استرینگ است و می‌خواهیم زمان انجام آن را اندازه‌گرفته و همزمان کارهای دیگری را انجام دهیم. (نمونه کد ۲۱۴)

```

1  class Program
2  {
3      private static void DownloadAsyncSimple()
4      {
5          using (HttpClient client = new HttpClient())
6          {
7              var asyncResultIust = client.GetStringAsync("http://www.iust.ac.ir/");
8              var asyncResultGithub = client.GetStringAsync("https://github.com/");
9              var asyncResultGoogle = client.GetStringAsync("https://www.google.com");
10             int i = 0;
11             while (asyncResultIust.IsCompleted != true ||
12                 asyncResultGithub.IsCompleted != true ||
13                 asyncResultGoogle.IsCompleted != true)
14             {
15                 Console.WriteLine(${i++} "Waiting...");
16                 Console.Write($,"google:{asyncResultIust.IsCompleted}");
17                 Console.Write($,"stack:{asyncResultGithub.IsCompleted}");
18                 Console.WriteLine($,"varzesh:{asyncResultGoogle.IsCompleted}");
19                 Console.WriteLine("Work" "Doing");
20                 Thread.Sleep(50);
21             }
22             Console.WriteLine(asyncResultIust.Result.Length);
23         }
24     }
25     static void Main(string[] args)
26     {
27         DownloadAsyncSimple();
28     }
}

```

نمونه کد ۲۱۴ : دانلود محتوای سایت‌ها

توضیح مرحله به مرحله:

مرحله ۱ : ابتدا، آبجکتی از کلاس HttpClient به نام client می‌سازیم.* خط ۵

مرحله ۲ : client متodi 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 asyncResultVarzesh و asyncResultStackOverflow و asyncResultGoogle را تعریف

کردیم تا محتوای دانلود شده را در آن ها بریزیم. *خط ۷ و ۸ و ۹*

- نکته ای که در متدهایGetStringAsync وجود دارد، این است که خط ۷ و ۸ و ۹ لزوماً به این معنی نمی‌باشد که همان موقع نتیجه آماده شده است و در متغیر ریخته شده است، بلکه چون متدهای 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 تسك از استرینگ می‌باشد که یک ویژگی بولین به نام IsCompleted دارد که اگر true باشد به معنی این است که تسك 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 انتهای نیز طول یکی از محتواهای دانلود شده را چاپ می‌کنیم. *خط ۱۱ تا ۲۲*

قسمتی از ابتدا و انتهای خروجی این نمونه کد در صفحه‌ی 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
```

DownloadTaskDemo ۲.۳.۲۳

* در این قسمت می خواهیم با مت دیگری در رابطه با تسك ها آشنا شویم. (نمونه کد ۲۱۵)

```

1  class Program
2  {
3      private static void DownloadTaskDemo()
4      {
5          using (HttpClient client = new HttpClient())
6          {
7              var asyncResultGoogle = client.GetStringAsync("http://www.google.com/");
8              var asyncResultGithub = client.GetStringAsync("http://github.com");
9              var asyncResultVarzesh = client.GetStringAsync("http://varzesh3.com");
10
11             asyncResultGoogle.ContinueWith(str => Console.WriteLine($"Goog-Done-{str.Result.Length}"));
12             asyncResultGithub.ContinueWith(str => Console.WriteLine($"Github-Done-{str.Result.Length}"));
13             asyncResultVarzesh.ContinueWith(str => Console.WriteLine($"Varzeh-Done-{str.Result.Length}"));
14
15             Task.WaitAll(asyncResultGoogle, asyncResultGithub, asyncResultVarzesh);
16         }
17         Console.WriteLine("Done.");
18     }
19     static void Main(string[] args)
20     {
21         DownloadTaskDemo2();
22     }
}

```

نمونه کد ۲۱۵ ContinueWith :

توضیح: یک اکشن که ورودیش یک `Task<string>` می باشد را به عنوان ورودی خود می گیرد و طرز کار آن این است که هر موقع تسك اولیه تمام شد، در ادامه‌ی آن تسك دیگری را انجام دهد. و در انتها با نوشتن `Task.WaitAll` و دادن تسك های اولیه به عنوان ورودی برنامه منتظر می‌ماند تا همه‌ی تسك ها انجام شوند و سپس بقیه‌ی کد را اجرا می‌کند.

خروجی نمونه کد بالا به صورت زیر می‌باشد.

```

Goog-Done-47062
Varzeh-Done-50592
Github-Done-131868
Done.

```

GetUrlContentLength ۳.۳.۲۳

* در بخش بعدی می‌خواهیم یک تابع بنویسیم که آدرس سایت را به صورت استرینگ از ورودی بگیرد و خروجی آن تسكی از نوع int است.(نمونه کد ۲۱۶)

```

1 class program
2 {
3     static Task<int> GetUrlContentLength(string url)
4     {
5         HttpClient client = new HttpClient();
6         var asyncResultGoogle = client.GetStringAsync(url);
7         Task<int> result = asyncResultGoogle.ContinueWith((ts) => ts.Result.Length);
8         return result;
9     }
10    static void Main(string[] args)
11    {
12        var tasks = new Task<int>[]{
13            GetUrlContentLength("http://stackoverflow.com"),
14            GetUrlContentLength("http://www.google.com"),
15            GetUrlContentLength("http://varzesh3.com"),
16        };
17
18        Task.WaitAll(tasks);
19
20        foreach (var t in tasks)
21        {
22            Console.WriteLine(t.Result);
23        }
24    }
25 }
```

نمونه کد ۲۱۶

توضیح: ابتدا، مانند قسمت های قبلی بک آبجکت از کلاس HttpClient به نام client تعریف کردیم.* خط ۵ سپس خروجی متدهGetStringAsync را که یک تسك از نوع استرینگ می‌باشد را در خط ۶ سپس تسك جدیدی را با نوع خروجی int به نام result تعریف کردیم و گفتیم که هر موقع تسك asyncResultGoogle تمام شد، در ادامه طول محتوای دانلود شده را به ما برگرداند.* خط ۷ در Main برنامه نیز آرایه ای از تسك هایی تعریف کردیم که خروجی‌شان از نوع int می‌باشد و با دستور Task.WaitAll(tasks) گفتیم که صبر کند تا تمامی تسك ها تمام بشود و سپس با دستور foreach در طول آرایه پیمایش می‌کنیم و خروجی هر تسك 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
```

GetUrlContentLengthAsync ۴.۳.۲۳

* در این قسمت می خواهیم مفهوم جدیدی را بررسی کنیم. وقتی متدهای async باشد، می توانیم در آن از کلیدواژه await استفاده کنیم. (نمونه کد ۲۱۷)

```
1 class program
2 {
3     static async Task<int> GetUrlContentLengthAsync(string url)
4     {
5
6         HttpClient client = new HttpClient();
7         Console.WriteLine(${url}" Before-Await:");
8         string result = await client.GetStringAsync(url);
9         Console.WriteLine(${url}" After-Await:");
10        return result.Length;
11    }
12
13    static void Main(string[] args)
14    {
15        var tasks = new Task<int>[]{
16            GetUrlContentLengthAsync("http://stackoverflow.com"),
17            GetUrlContentLengthAsync("http://www.google.com"),
18            GetUrlContentLengthAsync("http://varzesh3.com"),
19        };
20
21        Task.WaitAll(tasks);
22
23        foreach (var t in tasks)
24        {
25            Console.WriteLine(t.Result);
26        }
27    }
28 }
```

نمونه کد ۲۱۷ : async method

توضیح: این متدهای مشابه قسمت قبلی می باشد، با این تفاوت که وقتی از کلمه await استفاده می کنیم و متدهای صدا می زنیم، خط های قبل از این کلمه اجرا می شوند و سپس برنامه متوقف می شود و دوباره به جایی که متدهای صدا زده شده می رود و دوباره خط های اجرا شوند، تا موقعی که متدهای صدا زده نشود و حال خطوط بعد از await شروع به اجرا شدن می کنند و به محض اینکه تمام شود طول استرینگ دانلود شده را برمی گرداند.

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 بپر مفهوم `async` و `await` می‌توانید به لینک های [\[asyncprog\]](#) و [\[asyncemic\]](#) رجوع کنید.

جلسه ۲۴

Async Pattern و Tasks و Thread

زهرا مومنی نژاد - ۱۳۹۹/۷/۱۸

جزوه جلسه ۱۲۴ مورخ ۱۳۹۹/۷/۱۸ درس برنامه‌سازی پیشرفته تهیه شده توسط زهرا مومنی نژاد. در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته

در این جلسه به مفهوم زیر به صورت گستره پرداخته شد: Thread و همچنین نکاتی در جهت تکمیل مطالب زیر بیان شد: Async pattern،

Async و Await ۱.۲۴

وقتی ما با UI سروکار داریم و یک متده که زمان اجرای آن طولانیست (مثل خواندن یک فایل بزرگ و ذخیره آن در پایگاه داده) را در رویداد کلیک یک دکمه میگذاریم زمانی که روی آن دکمه کلیک شود رابط کاربری اپلیکیشن قفل شده و به اصلاح هنگ میکند ، زیرا رابط کاربری و بقیه متدها در برنامه نویسی همگام (Synchronous) در یک نخ (Thread) از سی پی یو اجرا میشوند پس رابط کاربری تا زمانی که فعالیت متده خاتمه نیابد پاسخی به کاربر نمیدهد.

برنامه نویسی ناهمگام در این شرایط بسیار کارآمد است ، زیرا در این روش رابط کاربری و متدها به هم متکی نبوده و متدها به صورت جداگانه اجرا میشوند. این دو برچسب هستند که مشخص میکنند در کدام بخش کد

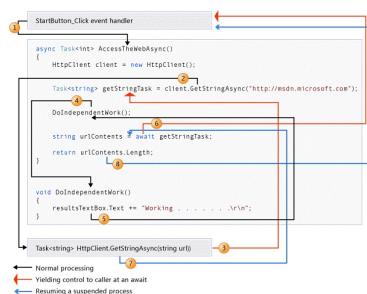
پاسخ دهی باید بعد از اتمام کار از سرگرفته شود.

زمانی که شما در بخشی از کد خود از کلمه کلیدی `async` و بر روی متدها، عبارات لامبда 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 باشد متوقف شود.

نکات مهم در الگوی متدهای `async`

- کلمه کلیدی `async` قبل از نوع بازگشته متد.
- نوع بازگشته از نوع `Task` که در اینجا به صورت ز می باشد زیرا خروجی متدهای یک عدد صحیح است.
- نام متدهای که با کلمه `Async` خاتمه یافته است.

نحوه کار متدهای `async`



شکل ۱.۲۴ : نحوه کار متدهای `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 `GetStringAsync` دانلود می شود.

- فعالیتی که در داخل متدهای `GetStringAsync` انجام می‌شود، باعث معلق شدن فرآیند اجرای متدهای دیگر می‌شود. ممکن است منتظر ماندن برای اتمام این فرآیند باعث قفل شده منابع شود. برای جلوگیری از بروز این مشکل متدهای `GetStringAsync` کنترل اجرای برنامه را به متدهای `DoIndependentWork` و `AccessTheWebAsync` می‌سپارند. متدهای `GetStringTask<TResult>` یک `Task<string>` باز می‌گردانند. مقدار بازگشتی از این متدهای متغیر `getStringTask` ذخیره می‌شود.
- از آنجا که `getStringTask` هنوز با عملگر `await` اجرا نشده است، می‌توان کار دیگری که به نتیجه بازگشتی متدهای `GetStringAsync` وابسته نیست را انجام داد. در مثال بالا متدهای `DoIndependentWork` و `AccessTheWebAsync` این کار را انجام می‌دهند.
- متدهای `DoIndependentWork` یک متدهای `synchronous` است و بعد از انجام کار خود به متدهای دیگر می‌گردد.
- کنترل اجرای برنامه به متدهای `AccessTheWebAsync` را فراخوانی کرده است باز می‌گردد و فعالیت‌هایی که به نتیجه `getStringTask` وابسته نیستند را انجام می‌دهد. زمانی که اجرای `Task<string>` پایان یابد کنترل اجرای برنامه به `AccessTheWebAsync` باز می‌گردد و سایر کدها را اجرا می‌کند.
- رشته موجود در `getStringTask` (تولید شده توسط عملگر `await`) `GetStringAsync` توسط عملگر `GetStringTask` گرفته شده و در متغیر `urlContents` ذخیره می‌گردد.
- حال متدهای `AccessTheWebAsync` محتوای سایت را دارد و می‌تواند طول آن را محاسبه کرده و به عنوان خروجی بازگرداند. سپس کار متدهای `AccessTheWebAsync` کامل می‌شود و برنامه می‌تواند کار خود را ادامه دهد.

یک مثال برای فهم بهتر از مطالب فوق: در این مثال دو متده هم متکی نیستند.

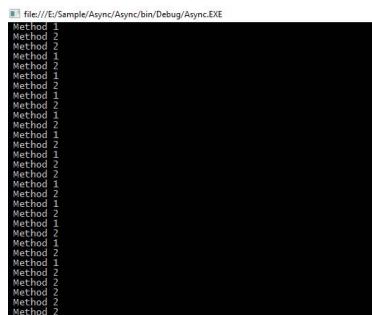
```

1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Method1();
6          Method2();
7          Console.ReadKey();
8      }
9
10     public static async Task Method1()
11     {
12         await Task.Run(() =>
13         {
14             for (int i = 0; i < 100; i++)
15             {
16                 Console.WriteLine(1" Method ");
17             }
18         });
19     }
20
21
22     public static void Method2()
23     {
24         for (int i = 0; i < 25; i++)
25         {
26             Console.WriteLine(2" Method ");
27         }
28     }
29 }
```

نمونه کد ۲۱۸: کد مثالی در سی شارپ

در کد بالا متده یک و دو به هم متکی نیستند و ما از متده Main آن هارا صدا میزنیم و میبینیم که متده 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 متده Main کلمه Async را استفاده کنیم به این اور میخوریم:



شکل ۳.۲۴: ارور دریافتی

Tasks ۲.۲۴

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 بیترن 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 متده WhenAny هر کدام از های Task در حال پردازش که خاتمه یابند، کل عملیات خاتمه خواهد یافت. فرض کنید نیاز دارید تا دمای کنونی هوای منطقه خاصی را از چند وب سرویس مختلف دریافت کنید. می توان در این حالت تمام این ها 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 می خواهید تعداد زیادی Url را پردازش کنید، اما نمی خواهید برای نمایش اطلاعات، تا پایان عملیات تمامی آن ها مانند WhenAll صبر کنید. می خواهید به محض پایان کار یکی از ها Task عملیات نمایش نتیجه آن را انجام دهید.

Task.Run() •

برای فراخوانی متده استفاده می شود.

Task.Start() •

زمانی که Task جدیدی را ایجاد می کنید به وسیله مت Task 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. 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);
```

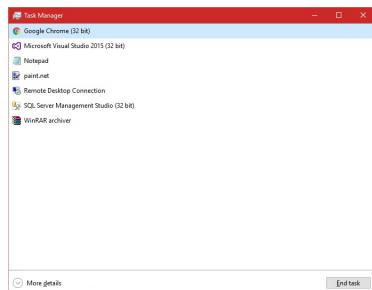
شكل ۴.۲۴ : دستور قابل استفاده جهت ایجاد وقفه در سی شارپ

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 می دهد. یکی از کاربردهای Delay منهای صبر کردن تا مدت زمانی مشخص، ایجاد مکانیزم timeout است. برای مثال حالت Task.WhenAny را در نظر بگیرید. اگر در اینجا timeout مدنظر ما ۳ ثانیه باشد، می توان یکی از ها Task.Delay را با آرگومان مساوی ۳۰۰۰ معرفی کرد. اگر هر کدام از های task تعریف شده زودتر از ۳ ثانیه پایان یافته باشد که بسیار خوب؛ در غیر اینصورت Task.Delay معرفی شده کار را تمام می کند.

Thread ۳.۲۴

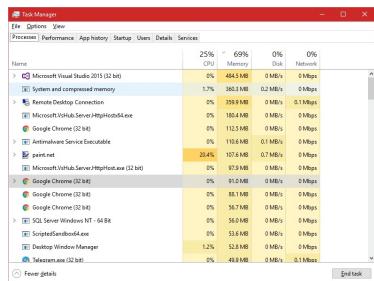
فرض کنید زمانی که در حال تماشای یک فیلم هستید دیگر امکان انجام کارهای دیگر، مثلًاً تایپ در برنامه Word یا برنامه نویسی نباشد. اما هیچ گاه این مشکلات برای شما بوجود نمی آید، زیرا سیستم عامل ها به بهترین شکل عملیات هم زمانی را پیاده سازی کرده و به شما این اجازه را می دهند تا در آن واحد نسبت به

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 مربنده 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 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، Process مربوط به برنامه paint.net مقدار ۴.۲۰ درصد از CPU و همچنین ۱۰۷MB از حافظه را اشغال کرده است. در اینجا بیشتر به بحث CPU 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 می‌گویند. هر Process می‌تواند شامل یک یا چندین Thread باشد که هر Thread وظیفه انجام یک عملیات خاص را برعهده دارد. اما زمان اجرای یک Thread اولیه اجرا می‌شود که به آن Thread Main گفته می‌شود. با دو مفهوم Thread و Process آشنا شدیم، اما همانطور که گفتیم در زبان سی شارپ می‌توانیم Thread هایی ایجاد کنیم که هر Thread یک کار خاص را انجام می‌دهد. برای کار با Thread ها و برای شروع، با کلاسی به نام Thread که در فضای نام System.Threading قرار دارد کار می‌کنیم. به صورت زیر می‌توانیم یک thread جدید با استفاده از کلاس Thread ایجاد کرده و آنرا اجرا کنیم:

```

1 static void Main(string[] args)
2 {
3     var thread1 = new Thread(Thread1Job);
4     var thread2 = new Thread(Thread2Job);
5     var thread3 = new Thread(Thread3Job);
6     thread1.Start();
7     thread2.Start();
8     thread3.Start();
9 }
10
11 public static void Thread1Job()
12 {
13     for (int counter = 0; counter < 50; counter++)
14     {
15         Console.WriteLine(" thread1: " + counter);
16     }
17 }
18
19 public static void Thread2Job()
20 {
21     for (int counter = 0; counter < 50; counter++)
22     {
23         Console.WriteLine(" thread2: " + counter);
24     }
25 }
26
27 public static void Thread3Job()
28 {
29     for (int counter = 0; counter < 50; counter++)
30     {
31         Console.WriteLine(" thread3: " + counter);
32     }
33 }
```

نمونه کد ۲۱۹: کد مثالی در سی شارپ

همانطور که مشاهده می کنید در کد بالا ۳ شی از نوع Thread ایجاد کردیم و برای پارامتر Constructor متد مورد نظر را ارسال کردیم. کلاس Thread Constructor 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 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 ThreadJob پس از اجرای ThreadJob اجرا شده و الى 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 صرکنیم 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 نیاز با 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 کاربردهای 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 مثلاً چند تخم مرغ و ساندیچ را درست کنیم می توانیم از کد زیر استفاده کنیم:



شکل ۸.۲۴: متد قابل استفاده جهت درست کردن تخم مرغ و ساندویچ به تعداد دلخواه

```

15  [System]
16  static void Main(string[] args)
17  {
18      Console.WriteLine($"Starting breakfast {Thread.CurrentThread.ManagedThreadId}");
19
20      var eggResult = MakeEgg();
21      var sandResult = MakeSandwich();
22
23      var breakfastReady = Task.WhenAll(eggResult, sandResult).ContinuedWith(t =>
24          Console.WriteLine($"Making Toast {Thread.CurrentThread.ManagedThreadId}"));
25
26      breakfastReady.Wait();
27
28      Console.WriteLine($"Breakfast is ready: {Thread.CurrentThread.ManagedThreadId}");
29
30      Parallel.Invoke(
31          () => Console.WriteLine("Hacking egg"),
32          () => MakeSandwich
33      );
34
35      int i = 10;
36      Task<string> t = Task.Run(() => $"test ({i})");
37      Task<string> t1 = new Task<string>(() => $"test ({i})");
38
39      t1.Start();
40  }

```

شکل ۹.۲۴ متد فوق Main

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 نوشتمن 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 تمام 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:

```

1  static void Main2(string[] args)
2  {
3      EggParam p = new EggParam();
4      p.EggCount = 4;
5      Thread tEgg = new Thread(
6          new ParameterizedThreadStart(MakeEgg)
7      );
8      tEgg.Name = "Thread" "Egg";
9      Thread tSandwich = new Thread(MakeSandwitch);
10     tSandwich.Name = "Thread" "Sandwitch";
11
12     tEgg.Start(p);
13     tSandwich.Start();
14
15     tEgg.Join();
16     tSandwich.Join();
17     Console.WriteLine(p.TimeSpent);
18     Console.WriteLine("Ready!" is "Breakfast");
19 }

```

نمونه کد ۲۴۰ برنامه Main

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 thread 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:

```

1  public static void PrintNumbers()
2  {
3      Console.WriteLine(" > numbers printing is {0}, Thread.CurrentThread.Name");
4      for (int counter = 0; counter < 10; counter++)
5      {
6          Thread.Sleep(200 * new Random().Next(5));
7          Console.WriteLine(counter);
8      }
9      Console.WriteLine();
10 }
```

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:

```

1  static void Main(string[] args)
2  {
3      Thread[] threads = new Thread[10];
4
5      for (int index = 0; index < 10; index++)
6      {
7          threads[index] = new Thread(PrintNumbers);
8          threads[index].Name = string.Format("#{0}." thread "Worker", index);
9      }
10
11     foreach (var thread in threads)
12     {
13         thread.Start();
14     }
15
16     Console.ReadLine();
17 }
```

نمونه کد ۲۲۲: برنامه Main

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:

```
Worker thread #0: Is printing numbers > Worker thread #1: Is printing numbers > Worker thread #2:  
Is printing numbers > Worker thread #3: Is printing numbers > Worker thread #4:  
Is printing numbers > Worker thread #5: Is printing numbers > Worker thread #6:  
Is printing numbers > Worker thread #7: Is printing numbers > { 1, 2, 3, 4, 5, 6, 7, 8, 9,  
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99 }  
Is printing numbers > Worker thread #8: Is printing numbers > { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,  
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99 }
```

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 گفته شد مشکل وجود همزمانی یا Concurrency در زمان اجرای thread هاست. زمانبندی 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 شی 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 می‌گویند کلیه کدهایی که در بدنه lock قرار دارند thread-safe هستند. برای اینکه کد داخل متدهای 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 متدهای PrintNumbers را به صورت زیر تغییر می‌دهیم:

```
1  public static void PrintNumbers()
2  {
3      lock (threadLock)
4      {
5          Console.WriteLine(" > numbers printing is " + Thread.CurrentThread.Name);
6          for (int counter = 0; counter < 10; counter++)
7          {
8              Thread.Sleep(200 * new Random().Next(5));
9              Console.WriteLine(counter);
10         }
11     }
12 }
13 }
```

نمونه کد ۲۲۳: تغییر متدهای PrintNumbers

با اعمال تغییر بالا، زمانی که thread جدیدی قصد وارد شدن به scope مشخص شده را داشته باشد، باید منتظر بماند تا کار 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 و درک بهتر مفاهیم فوق می توانید به لینک های زیر مراجعه کنید:

[Asynchronous] [task] [parallel] [thread] [asyncprog] [asyncprog] [asyncmic]

جلسه ۲۵

LINQ : Language Integrated Query

فاطمه میرجليلي - ۱۳۹۹/۲/۲۷

جزوه جلسه ۱۲۵ام مورخ ۱۳۹۹/۲/۲۷ درس برنامه‌سازی پیشرفته تهیه شده توسط فاطمه میرجليلي.

در ادامه پرداختن به موضوع Delegate و استفاده عمده از Events ، و Multi-Threading ، Strategy Pattern * که در جلسات پیشین گفته شد به معرفی یکی از feature های زبان سی شارپ به نام Linq می پردازیم.

کوتاه شده عبارت Linq Query Integrated Language به معنای زبان جستجوی 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 سی شارپ به شمار می رود و معادلی برای آن در زبان هایی مثل java موجود نمی باشد.

* 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 هایی که در آن استفاده می‌شود باید استاتیک باشد.

در نمونه کد زیر یک مثال از یک متده استاتیک را داخل کلاس استاتیک Ext مشاهده می‌کنید که به این متده گفته می‌شود .

```

1 static class Ext
2 {
3     public static int Next(this int n, int offset)
4     {
5         return n+offset;
6     }
7 }
```

نمونه کد ۲۲۴ : در اینجا متده استاتیک Next به عنوان یک گزینه برای متغیر n تعریف می‌شود .

طریقه استفاده از آن به صورت زیر است .

```

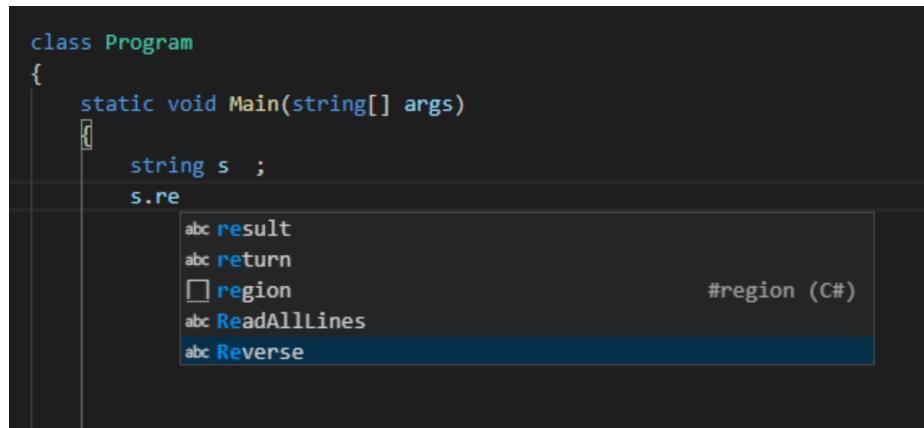
1 static void Main(string[] args)
2 {
3     int w = 5;
4     Console.WriteLine(w.Next(4));
5 }
```

برای مثال دیگر تابع استاتیک Reverse را به صورت زیر پیاده سازی می‌کنیم

```

1 public static string Reverse(this string str)
2 {
3     char[] chs = str.ToCharArray();
4     for(int i =0 ; i< chs.Length/2 ; i++)
5         (chs[i] , chs[chs.Length-i-1])=(chs[chs.Length-i-1] , chs[i]);
6     return new string(chs);
7 }
```

حال می‌بینیم برای هر متغیر از نوع string یک گزینه به نام Reverse موجود می‌باشد



```

1 string s = "AliHossein";
2 Console.WriteLine(s.Reverse());

```

نمونه کد ۲۲۵: طریقه استفاده از `method extention` در کد

در اینجا اگر بخواهیم متدهای Next را به گونه ای بنویسیم که دو پارامتر از ورودی بگیرد می توانیم آن را به صورت زیر بنویسیم و Next را از داخل کلاس Ext صدا بزنیم.

```

1 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

```

1 public static _Type[] Reverse<_Type>(this _Type[] chs)
2 {
3     for(int i=0; i<chs.Length/2; i++)
4         (chs[i], chs[chs.Length-i-1]) = (chs[chs.Length-i-1], chs[i]);
5     return chs;
6 }
7 public static string Reverse(this string str) =>
8     new String(str.ToCharArray().Reverse());

```

- یکی دیگر از مباحث مهمی که در مقدمه LINQ باید گفته شود چگونگی تعریف و استفاده از انواع

reference type و class در واقع یک Tuple می باشد. Tuple مختلف متغیر هایی از نوع Tuple است.

۱.۲۵ Tuple انواع تعريف

- در این روش در واقع یک کلاس جداگانه ایجاد نمیشود و اطلاعات داخل آن ذخیره می شود .

```
1 var t = Tuple.Create("Zahra", "Hosseini", 8.19, 98521343);
```

از مشکلات استفاده از این تعريف 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 تعريف نمیشود استفاده کرد .

```
1 var c = new {
2     name = "Zahra",
3     lastName = "Hosseini",
4     gpa = 8.18,
5     stdId = 98532412
6 };
7 Console.WriteLine($"{c.gpa} of {c.lastName} has {c.name} Student);
```

حال میتوان برای هر کدام از اطلاعات نام دلخواه انتخاب کرد که با آن شناخته شود. این تعريف نیز محدودیت هایی دارد مثلا اگر بخواهیم این Tuple را به عنوان ورودی یک تابع به کاربریم هیچ تایپی (string,int,Tuple,var,...) قابل تعريف نیست .

```

var c = new {
    name = "Zahra",
    lastName = "Hosseini",
    gpa = 18.8,
    stdId = 98532412
};

PrintStd( c )
{
    Console.WriteLine($"Student {c.name} has gpa of {c.gpa}");
}

```

هیچ تایپی نمی توان به
پارامتر ورودی داد

Value Tuple •

نوع دیگری از Tuple میتوان تعریف کرد که برخلاف حالت قبل یک Struct از نوع Value type 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.

```

1 (string name, string lastName, double gpa, int stdId) student =
2 ("Zahra", "Hosseini", 8.19, 98521343);
3
4 static void PrintStd((string name, string lastName, double gpa, int stdId) std)
5 {
6     var (n, l, g, s) = std;
7     (string na, string la, double gp, int st) = std;
8     Console.WriteLine(n, l, g, s);
9 }

```

از اطلاعات داخل این نوع Tuple میتوان به صورت item1, item2, ... و یا ایجاد نام جداگانه برای هر property استفاده کرد. همچنین میتوان از عملگر های = = و != برای مقایسه مقادیر دو استفاده کرد.

```

1  (string, string, double, int) std2 = student;
2  string name = "ali";
3  string lastName = "Zahraei";
4  double gpa = 9.19;
5  int stdId = 98521234;
6
7  var std3 = (name, lastName, gpa, stdId);
8
9  if ( std2 != std3 )
10 {
11     Console.WriteLine("equal");
12 }
```

Tuple کردن یک Deconstruct ۲.۲۵

Tuple کردن یک Tuple به این معنی است که هر کدام از item های مختلف یک Tuple deconstruct در متغیر های محلی تعریف می کنیم این کار این قابلیت را به ما می دهد تا از مقادیر item ها به صورت جداگانه استفاده کنیم.

```

1 static void PrintStd((string name, string lastName, double gpa, int stdId) std)
2 {
3     var (n, l, g, s) = std;
4     (string na, string la, double gp, int st) = std;
5     Console.WriteLine(n, l, g, s);
6 }
```

LINQ ۳.۲۵

کوتاه شده عبارت linq به معنای زبان پرس و جوی یکپارچه Language Integrated Query است. که برای استخراج و استفاده از داده های یک Database میتوان از آن کمک گرفت. عمدۀ لینک از extention method iEnumerable هایی روی تایپ تشكيل شده است.

در ابتدا برای استفاده از linq در کد خود باید از using System.Linq ; در ابتدای صفحه استفاده شود.

```
1 using System.Linq;
```

در نمونه کد زیر یک مثال از یک آورده شده است.

```

1  public static double[] ToDouble(this string[] list)
2  {
3      double[] result = new double[list.Length];
4      for(int i=0; i<list.Length; i++)
5          result[i] = double.Parse(list[i]);
6      return result;
7  }
8
9  string[] nums = {"6.15", "9.19", "2.17", "4.13", "6.15"};
10 double[] numsParsed = nums.ToDouble();
11
12 var numParsedWithLinq = nums.Select(s => double.Parse(s));

```

Select():

متدهای `Select()` یکی از متدهای پرکاربرد `linq` است که کاربرد آن دقیقاً معادل کلمه `select` میباشد. این متد یک `Lambda expression` به عنوان پارامتر ورودی میپذیرد. که عموماً نوع برگشتی آن یک `iEnumerable` شامل اطلاعات درخواستی است.

در این مثال یک `Select` و یک `iEnumerable` به عنوان ورودی گرفته و پس از اجرای `روی تک تک اعضای` یک `iEnumerable` به عنوان خروجی تحویل می‌دهد.

```

1  public static IEnumerable<_OutType> MySelect<_OutType, _InType>(
2      this IEnumerable<_InType> list, Func<_InType, _OutType> fn)
3  {
4      foreach(var input in list)
5          yield return fn(input);
6  }

```

نمونه کد ۲۲۶: شرح متدهای `Select` به صورت یک `function`

OrderBy():

متدهای `OrderBy()` اطلاعات موجود را بر اساس `key selector` که در بطن به آن داده میشود به صورت صعودی مرتب می‌کند.

```
1  .OrderBy(n => (n))
```

نمونه کد ۲۲۷: در اینجا `key selector` همان `object n` است.

OrderByDescending():

این متدهمانند متدهمانند OrderBy() است با این تفاوت که اطلاعات به صورت نزولی مرتب میشوند.

```
1   .OrderByDescending(n => n)
```

ToList():

این متدداده های موجود را به صورت یک لیست به خروجی تحويل می دهد.

ForEach():

زمانی که داده ها به صورت یک لیست موجود باشد متدهمانند ForEach براساس Action داده شده به آن عمل موردنظر را روی تک تک داده های لیست انجام می دهد. تفاوت این متده با متدهای گفته شده در این است که ورودی آن به جای Action یک Func می باشد.

```
1   .ToList()
2   .ForEach(n => WriteLine(n));
```

Take(int x);

این متده اندازه x تا از داده های موجود را برمیگرداند.

Where()

این متده برای فیلتر کردن داده ها براساس دارا بودن یک ویژگی یا شرط ورودی اش به کار می رود.

```
1   .Where(t => t.country != "INVALID")
```

Skip(int x);

این متده اول را از Enumerable شده به آن حذف میکند.

Trim()

این متده می تواند روی یک پارامتر از نوع `string` صدا زده شود و میتواند چند کاراکتر را به عنوان ورودی بپذیرد . در صورت وجود کاراکتر ها در ابتدا یا انتهای `string` موجود حذف میشوند.

```
١ Trim(" ", " ");
```

SelectMany():

این متده 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 متده می توانید به مثال زیر توجه کنید.

```
١ .SelectMany(t => {
 ٢   return new (string country, double percent) []{
 ٣     (t.country, t.male),
 ٤     (t.country, t.female)};
 ٥ })
```

نمونه کد ۲۲۸ : در این جا ورودی یک `Tuple` و خروجی دو `item` های دلخواه از `Tuple` اولیه می باشد.

Average():

این متده میانگین داده هارا بر اساس پارامتر ورودی اش برمی گرداند.

```
١ .Average(t => t.percent);
```

نمونه کد ۲۲۹ : در این مثال میانگین داده ها بر اساس `percent` بدست می اید

```

1 static void Main(string[] args)
2 {
3
4     File.ReadAllLines("Database")
5         .Skip(2)
6         .Select(l => {
7             var toks = l.Split(',').Select(t => t.Trim(" ", " ")).ToArray();
8             try
9             {
10                 return (
11                     country:toks[0],
12                     year:int.Parse(toks[1]),
13                     both:double.Parse(toks[2]),
14                     male:double.Parse(toks[3]),
15                     female:double.Parse(toks[4])
16                     );
17             }
18             catch
19             {
20                 return ("INVALID",0, 0, 0, 0);
21             }
22         })
23         .Where(t => t.country != "INVALID")
24         .SelectMany(t => {
25             return new (string country, double percent)[]{
26                 (t.country, t.male),
27                 (t.country, t.female)};
28         })
29         .OrderBy(t => t.percent)
30         .OrderByDescending(t => t.female - t.male)
31         .Take(20)
32         .ToList()
33         .ForEach(t => WriteLine(t));
34     }

```

نمونه کد ۲۳۰ : نمونه استفاده از تمام متد های ذکر شده

جلسه ۲۶

لينک

محمد حسین رجبی - ۱۳۹۹/۳/۱۶

۱.۲۶ ديزاين پtern (الگوي طراحی) چيست؟

- ديزاين پtern ها راه حل های رايچ و قابل استفاده برای مشكلات رايچ در طراحی نرم افزار هستند
- ديزاين پtern ها راه کارهایي بهينه و با قابلیت استفاده مجدد برای مشكلات برنامه نویسي هستند.

ديزاين پtern ها سرعت کد نویسي رو بالا ميرند چونکه الگوهایي هستند از پيش آمده و تست شده که در اختيار برنامه نويس ها برای انواع موقعیت ها استفاده میشوند.

ديزاين پtern ها به خودی خود مشكلات رو حل نمیکنند بلکه ابزار مناسبی هستن که به ما در حل مشكلات در برنامه نویسي کمک میکنند.

در اين جلسه حول فايل `gdp.csv` که مخفف `gross Domestic product` يا همان فعالیت اقتصادي مطالب جديدي در مورد لينک ياد بگيريم.

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

۵

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 `iran` هستند را جدا میکنیم سپس با متدهای `properSplit` هر خط را به ۴ قسمت تبدیل کرده و به عنوان خروجی یک `tuple` که به ترتیب مشتمل 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. اصطلاحاً به این حالت `Lazy Evaluation` میگویند که مربوط به `IEnumerable` است.

```

1 var iranLines = File.ReadAllLines(@"path")
2     .skip(1)
3     .where(l => l.ToLower().Contains("iran"))
4     .select(l => {
5         var toks = properSplit(l).ToArray();
6         return (
7             Country : toks[0],
8             year: int.Parse(toks[2]),
9             gdp : double.Parse(toks[3])
10            );
11        });

```

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 بر میگرداند.

```

1  private static IEnumerable<string> properSplit(string line)
2  {
3      var toks = line.Split("", StringSplitOptions.RemoveEmptyEntries);
4      yield return toks[0];
5      foreach (var t in toks[1].Split(',', StringSplitOptions.RemoveEmptyEntries))
6      {
7          yield return t;
8      }
9  }

```

نمونه کد ۲۳۲: متد properSplit

یکی دیگر از توابع قابل استفاده `OrderBy()` است که به صورت دیفالت از کم به زیاد بر اساس متغیر `sort` میکند. در این مثال میتوان با `OrderBy(t => t.gdp)` تمامی `tuple` ها را بر اساس فعالیت اقتصادی مرتب کرد.

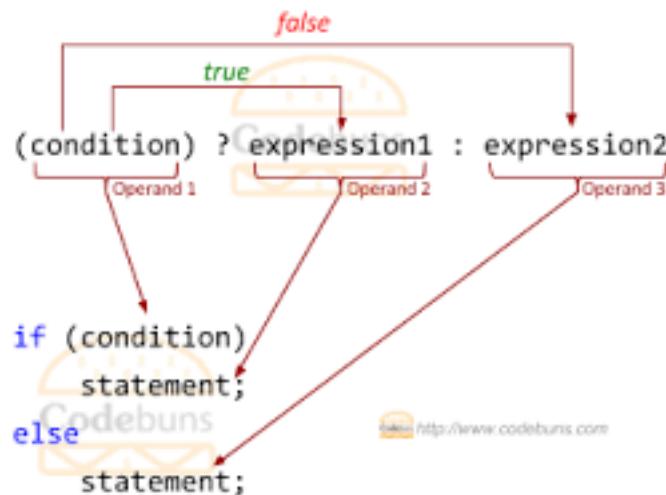
در ادامه شما میتوانید با `First()` و `Last()` به اولین و آخرین عضو دسترسی داشته باشید.

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()` و `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 `Min(t => t.gdp)` یا `Max(t => t.gdp)` می توان این دو مقدار را پیدا کرد.

Ternary operator ۳.۲۶



Aggregate ۴.۲۶

روشی برای جمع بندی است که دو ورودی گرفته و با الگوریتم داده شده فقط یه خروجی میدهد .

```
1 result.Aggregate( (t1, t2) => t1.gdp > t2.gdp ? t1 : t2).ToString()
2 result.Aggregate( (t1, t2) => t1.gdp < t2.gdp ? t1 : t2).ToString()
```

نمونه کد : ۲۳۳

در مثال بالا Aggregate دو tuple t1 , t2 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 دارد است را پیدا کند. توجه شود که result مجموعه ای تمام tuple هاست .

GroupBy ۵.۲۶

یکی دیگر از موارد قابل استفاده در این بخش GroupBy است که شما میتوانید یک **key selector** به آن دهید این متند اعضا یی که 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 :

```

1   result.where(t => t.year == 2012)
2       .GroupBy(t => (int) (t.gdp / 100))
3       .OrderByDescending(g => g.Key)
4       .ToList()
5       .ForEach(g => {
6           g.Key.ToString().Dump($" {g.Count()} \"Key\"");
7           g.ToList().ForEach(t => t.Dump());
8       });

```

نمونه کد ۲۳۴ :

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 :

```

1 File.ReadAllLines(@"path")
2     .skip(1)
3     .select(l => {
4         var toks = properSplit(l).ToArray();
5         return (
6             Country : toks[0],
7             Code : toks[1],
8             year: int.Parse(toks[2]),
9             pop : int.Parse(toks[3])
10            );
11        });

```

نمونه کد ۲۳۵

که روند مشابهی با کد قبل داد .

Join ۷.۲۶

برای ترکیب دو لیست به کار میروند . بطوریکه به ازای عضوی که در هر دو لیست وجود دارد یک **Func** خاصی را روی آنها انجام دهد .

برای درک بهتر به مثال زیر توجه کنید:

```

1   gdpResult.Join(popResult,
2     t => (code: t.code, year: t.year),
3     t => (code: t.code, year: t.year),
4     (t1, t2) => (t1.country, t1.year, normgdp:t1.gdp / t2.pop)
5   )
6   .Where(t => t.year == 2012)
7   .OrderBy(t => t.normgdp)
8   .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 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 :

- برای حذف عناصر تکراری به کار میروند . **Distinct**
- دو لیست را پشت هم میاورد یا میچسباند . **concat**
- چک کردن وجود یک عضو در لیست **Contains**
- آیا عضوی هست که یک شرط را برقرار کند . **Any**
- آیا همه ی اعضا شرطی را برقرار میکنند . **All**

۲۷ جلسه

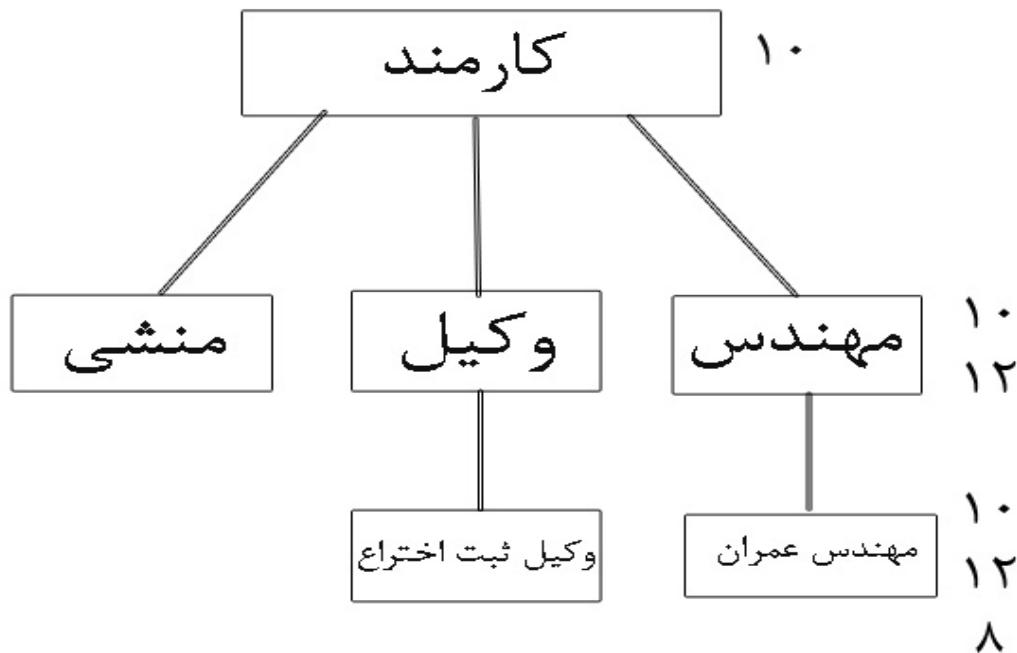
وراثت

امیرحسین درخشنان - ۱۳۹۹/۳/۳

جزوه جلسه ۱۲۷م مورخ ۱۳۹۹/۳/۳ درس برنامه‌سازی پیشرفته تهیه شده توسط امیرحسین درخشنان. در جهت
مستند کردن مطالب درس برنامه‌سازی پیشرفته

۱.۲۷ کاربرد وراثت

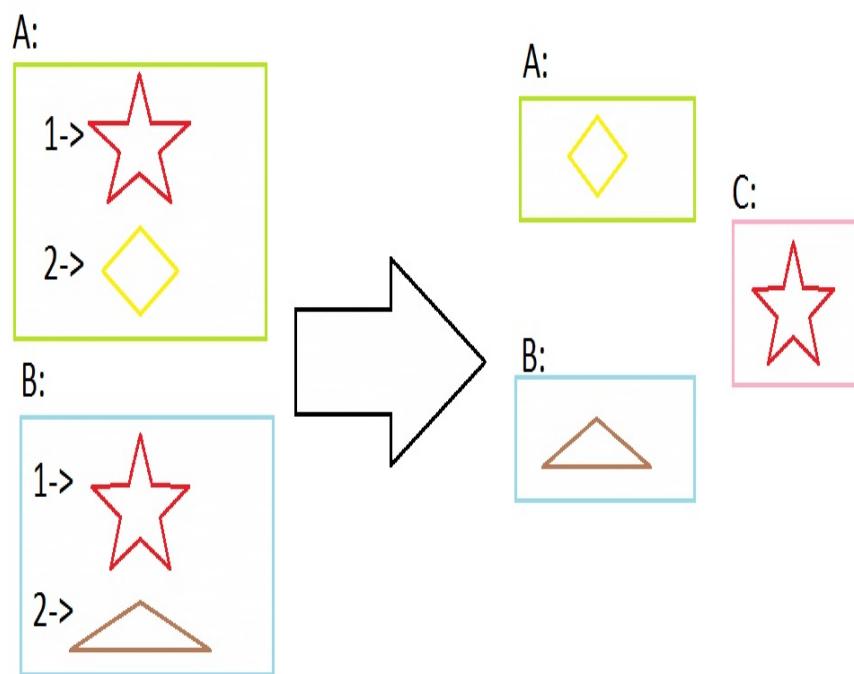
وراثت در برنامه نویسی برای اضافه کردن ویژگی های یک کلاس به کلاسی دیگر هست یعنی وقتی می گوییم کلاس B از کلاس A ارث بری میکند یعنی تمام ویژگی های A را دارد برای روشن تر شدن موضوع به عکس و مثال زیر توجه کنید.



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 پس 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 آن را جدا کرد و در موارد لزوم از آن 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

```

1  namespace note
2  {
3      class Security
4      {
5          string Name;
6          long Salary;
7          public string DoType()
8          {
9              return $"Type" "{this.Name}";
10         }
11     }
12 }
```

نمونه کد : ۲۳۷

```

1  namespace note
2  {
3      class Engineer
4      {
5          string Name;
6          long Salary;
7          public string DoBuild()
8          {
9              return $"Build" "{this.Name}";
10         }
11     }
12 }
```

نمونه کد : ۲۳۸

این دو کلاس در دو خط اولشان با هم یکسان هستند پس میتوان این دو خط را در کلاس دیگری به نام قرار داد Employee

```

1  namespace note
2  {
3      class Employee
4      {
5          public string Name;
6          public long Salry;
7      }
8 }
```

نمونه کد : ۲۳۹

حال با توجه به توضیحات گفته شده کلاس های Security و Engineer میتوانند از کلاس Employee بردن نحوه این کار با استفاده از : میباشد

```

1  namespace note
2  {
3      class Security:Employee
4      {
5          public string DoType()
6          {
7              return $"Type" "{this.Name}";
8          }
9      }
10 }

```

نمونه کد ۲۴۰ class Security :

```

1  namespace note
2  {
3      class Engineer:Employee
4      {
5          public string DoBuild()
6          {
7              return $"Build" "{this.Name}";
8          }
9      }
10 }

```

نمونه کد ۲۴۱ class Engineer :

در این حالت کلاس Employee را کلاس پدر یا base class و به کلاس های Security و Engineer و 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

```

1 public class Employee
2 {
3     public string Name;
4     public long Salary;
5     public Employee(string name, long salary)
6     {
7         this.Name=name;
8         this.Salary=salary;
9     }
10    public void AddSalary(long adding)
11    {
12        this.Salary+=adding;
13    }
14 }
```

نمونه کد ۲۴۲ :

حال کلاس Engineer را هم در نظر بگیرید که از کلاس Employee ارث میبرد و علاوه بر ان دارای یک شی به نام string می باشد حال در constructor این کلاس باید از `base` استفاده کرد

```

1 using System;
2 namespace note
3 {
4     public class Engineer : Employee
5     {
6         public string Field;
7         public Engineer(string name, long salary, string field)
8             : base(name, salary)
9         {
10            this.Field=field;
11        }
12    }
13 }
```

نمونه کد ۲۴۳ :

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.

```

1 public virtual AddSalary(long adding)
2 {
3     this.Salary+=adding;
4 }
```

addsalary virtual :

و در کلاس Engineer باید برای این متد از override استفاده کنیم.

```

1 public override void AddSalary(long multing)
2 {
3     this.Salary*=multing;
4 }
```

نمونه کد ۲۴۵ : addsalary override

یا اگر بخواهیم در کلاس Engineer به همان صورت باشد اما به شرط خاصی مثلاً اگر حقوق فعلی اش زیر ۱۰۰۰۰۰۰ بود به ان اضافه شود باید این گونه کار کرد.

```

1 public override void AddSalary(long adding)
2 {
3     if(this.Salary<1_000_000)
4         base.AddSalary(adding);
5 }
```

نمونه کد ۲۴۶ : addsalary if

یعنی به طور کلی با `virtual base.method` به متد `base.method` 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 مثلاً در مثال private string Name public string Name از Employee و Engineer فرض کنیم به جای this.Name در کلاس Engineer استفاده میکردیم در این صورت در کلاس Employee 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 این که 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 مثلاً کلاس Employee از abstract استفاده میکنیم به این صورت که class Employee به این معناست که تنها در کلاس هایی که از آن ارث برده اند میتوان از آن استفاده کرد یعنی چنین کلاسی تنها برای ارث بری کلاس های دیگر ساخته شده است و در کلاس هایی که از آن ارث نبرده اند نمیتوان از new Employee() استفاده کرد.

جلسه ۲۷. وراثت

۲۷۵

Polymorphism ۷.۲۷

به مثال زیر توجه کنید

```

1  using System.Collections.Generic;
2  using System;
3
4  namespace polymorphism
5  {
6      public class Shape
7      {
8          public int X;
9          public int Y;
10         public int Height;
11         public int Width;
12         public Shape(int x,int y,int height,int width)
13         {
14             (X,Y,Height,Width)=(x,y,height,width);
15         }
16         public virtual void Draw()
17         {
18             Console.WriteLine(tasks" drawing class base "Performing");
19         }
20     }
21     public class Triangle : Shape
22     {
23         public Triangle(int x, int y, int height, int width)
24             : base(x, y, height, width)
25         { }
26         public override Draw()
27         {
28             Console.WriteLine(Triangle" "Drawing);
29             base.Draw();
30         }
31         public int Area()
32         {
33             return Height*Width/2;
34         }
35     }
36     public class Circle : Shape
37     {
38         public Circle(int x, int y, int height, int width)
39             : base(x, y, height, width)
40         { }
41         public override Draw()
42         {
43             Console.WriteLine(Circle" "Drawing);
44             base.Draw();
45         }
46     }
47     class Program
48     {
49         static void Main(string[] args)
50         {
51             Triangle triangle=new Triangle(1,2,3,4);
52             Circle circle=new Circle(5,6,7,8);
53             Shape shape=triangle;
54             List<Shape> shapes=new List<Shape>{circle,triangle};
55         }
56     }
57 }
```

Polymorphism : ۲۴۷ نمونه کد

همانطور که مشاهده کردید میتوان triangle و ya 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 متدهایی که در کلاس پدر هست میتوان استفاده کرد مثلاً نمیتوان در shape.Area() استفاده کرد

۸.۲۷ استفاده از Seald

اگر برای یک کلاس مثلاً کلاس Engineer از `seald` استفاده کنیم به این صورت که

```
seald class Engineer
```

یعنی هیچ کلاس دیگری نمیتواند از این کلاس ارث ببرد.

جلسه ۳۰

Design Patterns – State Pattern

پارسا عیسی زاده - ۱۳۹۹/۳/۱۸

جزوه جلسه ۳۰ ام مورخ ۱۳۹۹/۳/۱۸ درس برنامه‌سازی پیشرفته تهیه شده توسط پارسا عیسی زاده . در جهت مستند کردن مطالب درس برنامه‌سازی پیشرفته جزو 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 لزوماً 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 Facade pattern ، Factor pattern . GitHub جست و جو بکنیم .

ها با سی-شارپ را هم ببینیم میتوانیم در design pattern را یاد بگیریم . 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 .

```

1 internal class Account
2 {
3     public int Balance {get; private set;} = 0;
4     public bool IsVerified {get; private set;} = false ;
5     public bool IsClosed {get; private set;} = false ;
6     public bool IsFrozen {get; private set;} = false ;
7     public Account(Action onUnFrozen) {}
8     public void Deposit(int value) => this.Balance += value ;
9     public void Withdraw(int value) => this.Balance -= value ;
10    public void HolderVerified() => isVerified = true ;
11    public void Close() => this.IsClosed = true ;
12    public void Freeze() => this.IsFrozen = true ;
13
14 }

```

نمونه کد ۲۴۸ : 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 Withdraw و Deposit 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 .

Guard Clause ۲.۳۰

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

مطلوبیمان نبود از متد خارج میشویم . برای مثال در متد Withdraw :

```

1 public void Withdraw(int value)
2 {
3     if(!this.IsVerified)
4         return ;
5     if(!this.IsClosed)
6         return ;
7
8     this.Balance -= value ;
9     ManageUnFeezing();
10 }
```

نمونه کد ۲۴۹ : متد Withdraw نوشته شده با guard clause

یادآوری : تکرار کردن یک کد از گتاهان کبیره (!) ایست که یک برنامه نویس میتواند مرتب شود . ما گفتیم که در متد های Deposit و Withdraw 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 استفاده میکنیم .

```

1 private void ManageUnFreezing()
2 {
3     if(!this.Frozen)
4         return ;
5
6     this.IsFrozen = false ;
7     this.OnUnFrozen() ;
8 }
```

نمونه کد ۲۵۰ : متدی برای حالتی که حساب بسته شده

State Pattern ۳.۳۰

برای استفاده از if تا الان دو تا راه یادگرفتیم 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 ، 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 .

```

1 public action ManageUnFreezing = DoNothing ;
2 private void UnFreeze()
3 {
4     this.IsFrozen = false ;
5     this.OnUnFrozen();
6     this.ManageUnFreezing = DoNothing ;
7 }
8
9 private void Freeze()
10 {
11     this.IsFrozen = true ;
12     this.ManageUnFreezing = UnFreeze ;
13 }
```

نمونه کد ۲۵۱ : حذف if از بدنه متدها

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 بے یک delegate نیاز داریم . از آنجاییکه یک همانند delegate یک متد است و ما نیاز به چند delegate داریم ، یک interface جدید تعریف میکنیم .

```

1 internal interface IAccountState
2 {
3     IAccountState Withdraw(Action doWithdraw);
4     IAccountState Deposit(Action doDeposit);
5     IAccountState Freeze();
6     IAccountState HolderVerified();
7     IAccountState Close();
8 }
```

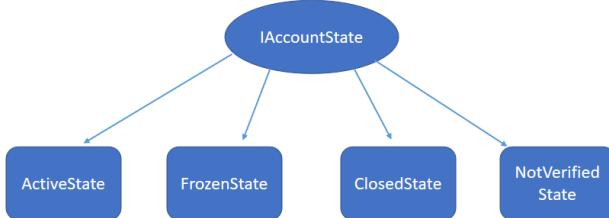
IAccountState : ۲۵۲

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 مان چه کاری انجام دهد و سپس وارد چه State ای بشود. به نوعی میتوان state ها را به دومینوی 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.

```

1 internal class ActiveState : IAccountState
2 {
3     private Action OnUnFrozen;
4
5     public ActiveState(Action onUnFrozen)
6     {
7         this.OnUnFrozen = onUnFrozen;
8     }
9
10    public IAccountState Close() => new ClosedState();
11
12    public IAccountState Deposit(Action doDeposit)
13    {
14        doDeposit();
15        return this;
16    }
17
18    public IAccountState Freeze() => new FrozenState(OnUnFrozen);
19
20    public IAccountState HolderVerified() => this;
21
22    public IAccountState Withdraw(Action doWithdraw)
23    {
24        doWithdraw();
25        return this;
26    }
27}
```

نمونه کد ۲۵۳

Frozen State ۲.۳.۳.**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 در آن 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 و گرنه deposit syntax مشابه 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 .

```

1 internal class FrozenState : IAccountState
2 {
3     private Action OnUnFrozen;
4     public FrozenState(Action onUnFrozen)
5     {
6         this.OnUnFrozen = onUnFrozen;
7     }
8     public IAccountState Close() => new ClosedState();
9     public IAccountState Deposit(Action doDeposit)
10    {
11        doDeposit();
12        this.OnUnFrozen();
13        return new ActiveState(OnUnFrozen);
14    }
15    public IAccountState Freeze() => this;
16    public IAccountState HolderVerified() => this;
17    public IAccountState Withdraw(Action doWithdraw)
18    {
19        doWithdraw();
20        this.OnUnFrozen();
21        return new ActiveState(OnUnFrozen);
22    }
23 }
```

FrozenState : ۲۵۴

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 متده است this را برمیگرداند.

Freeze

هویتی که تایید نشده نمیتواند حسابی را freeze کند (!) در نتیجه این متده است 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 متده است 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 متده انجام شود این است که حساب به حالت ClosedState برود.

```

1 internal class NotVerifiedState : IAccountState
2 {
3     private Action OnUnFrozen;
4     public NotVerifiedState(Action onUnFrozen)
5     {
6         this.OnUnFrozen = onUnFrozen;
7     }
8     public IAccountState Close() => new ClosedState();
9     public IAccountState Deposit(Action doDeposit)
10    {
11        doDeposit();
12        return this;
13    }
14    public IAccountState Freeze() => this;
15    public IAccountState HolderVerified() => new ActiveState(OnUnFrozen);
16    public IAccountState Withdraw(Action doWithdraw) => this;
17 }
```

نمونه کد ۲۵۵ : NotVerifiedState

Closed State ۴.۳.۳۰

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 بر میگرداند و کار دیگری انجام نمی دهد .

```

1 internal class ClosedState : IAccountState
2 {
3     public IAccountState Close() => this;
4     public IAccountState Deposit(Action doDeposit) => this;
5     public IAccountState Freeze() => this;
6     public IAccountState HolderVerified() => this;
7     public IAccountState Withdraw(Action doWithdraw) => this;
8 }
```

نمونه کد ۲۵۶ : 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 isClosed isVerified را برای تشخیص اینکه حساب در حالت state را میگذاریم برعهده کلاس state . پس تنها متد مورد نظر را برای کلاس state صدا می زنیم .

```

1 internal class Account
2 {
3     public int Balance {get; private set;} = 0;
4     IAccountState State;
5
6     public Account(Action onUnFrozen)
7     {
8         State = new NotVerifiedState(onUnFrozen);
9     }
10    public void Deposit(int value)
11    {
12        this.State = State.Deposit(() => {this.Balance += value;});
13    }
14    public void Withdraw(int value)
15    {
16        this.State = this.State.Withdraw(() => {this.Balance -= value;});
17    }
18    public void HolderVerified()
19    {
20        this.State = this.State.HolderVerified();
21    }
22    public void Close()
23    {
24        this.State = this.State.Close();
25    }
26    public void Freeze()
27    {
28        this.State = this.State.Freeze();
29    }
30}

```

Account : ۲۵۷

حال کلاس account تنها مقدار Balance و state را نگه میدارد و بقیه کارها را بقیه کلاس‌ها بسته به state حساب انجام میدهد. حال میبینیم که single responsibility principle که خوبی حس میشود. هم‌اکنون کد تمیز تری داریم که به راحتی مدیریت میشود. این pattern‌های مختلف برای طراحی برنامه علاوه بر اینکه کاربردی هستند نشان میدهد که ما چقدر برنامه نویسی بدلیم و مهم است که pattern‌های مختلف را یاد بگیریم.

۴.۳۰ ماشین حساب

یک مثال عملی برای state pattern ماشین حساب است. بعضی از state‌های 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.

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.

اگر دکمه ممیز فشرده شود برنامه وارد حالت اعشاری می‌شود که باز هر بار که دکمه صفر فشرده شود عدد نمایشگر تغییر می‌کند و در این حالت جدید اگر دوباره دکمه ممیز را زدیم عدد نمایشگر تغییری نمی‌کند.

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://refactoring.guru/design-patterns/behavioral-patterns>
http://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>