



دانشکده مهندسی کامپیوتر

جزوه درس

برنامه‌سازی پیشرفته

استاد درس: سید صالح اعتمادی

نیم‌سال دوم

سال تحصیلی ۹۸-۹۹

فهرست مطالب

۶	۲ Functions and Parameters	پریا فصاحت - ۱۳۹۸/۱۱/۱۹
۶	۱.۲ معرفی زبان‌های Native و Managed	
۱۱	۳ ادامه بحث توابع و پارامترها	نیوشا یقینی - ۱۳۹۸/۱۱/۲۱
۱۲	۴ Method/Function Features	محمد مهدی جاوید - ۱۳۹۸/۱۱/۲۶
۱۲	۱.۴ متد اورلودینگ	
	۲.۴ فرستادن براساس مقدار یا آدرس	
۲۰	Pass by reference VS Pass by value	
۲۷	۳.۴ توابع با پارامترهای ورودی متغیر (variadic functions) :	
۳۲	۴.۴ توابع و کلاس‌های عمومی (Generic Functions) :	
۳۷	۵.۴ قراردادهای نام‌گذاری (Naming Conventions) :	
۴۲	۶.۴ Fold Expression :	
۴۶	۷.۴ نحوه صحیح تقسیم کردن دو عدد بر یکدیگر :	
۴۹	۸.۴ تمرین کلاسی :	
۵۳	۵ Class and Object	روزبه غزوی - ۱۳۹۸/۱۱/۲۸
۵۴	۱.۵ تعریف کلاس (class)	

۵۴	ایجاد یک شیء از کلاس	۲.۵
۵۵	بررسی یک نمونه مثال از کلاس	۳.۵
۵۶	سطح دسترسی (Access Modifiers)	۴.۵
۵۷	فیلد (Field)	۵.۵
۵۷	سازنده (Constructor)	۶.۵
۵۸	متد (Method)	۷.۵
۵۸	صفت (Property)	۸.۵
۵۹	صفات Auto-implemented	۹.۵
۶۰	فضای نام (Namespace)	۱۰.۵

۶ آرایه‌ها و کلاس

نیکی تراکتی - ۱۳۹۸/۱۲/۳

۶۱	آرایه‌ها در ++C	۱.۶
۶۳	آرایه‌ها در Java	۲.۶
۶۴	آرایه‌ها در Python	۳.۶
۶۵	آرایه‌ها در C#	۴.۶
۶۶	کلاس در ++C	۵.۶

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

امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

۷۰	Value Type و Reference Type	۱.۷
۷۰	Value Type	۲.۷
۷۱	Pass by Value ارسال با مقدار	۳.۷
۷۲	Reference Type	۴.۷
۷۳	Pass by Reference ارسال با ارجاع	۵.۷
۷۴	Heap و Stack	۶.۷
۷۸	Java in Class	۷.۷

۸ کار با فایل در سی شارپ

بابک بهکام کیا - ۱۳۹۸/۱۲/۱۰

۷۹	آشنایی با فایل	۱.۸
۸۰	برنامه ثبت نام دانش آموزان	۲.۸

۹ شی گزایی و استاتیک

محمدجواد مهدی‌تبار - ۱۳۹۸/۱۲/۱۲

۸۵

۱.۹ اهداف اصلی این جلسه ۸۵

۲.۹ کلمه های کلیدی مهم ۸۵

۳.۹ دستورات مهم فایل ۹۲

۴.۹ کد زده شده درون کلاس ۹۳

۱۰ Directory/File) دیرکتوری و فایل

علی رهنما علمداری - ۱۳۹۸/۱۲/۱۷

۹۸

۱.۱۰ Directoryclass ۹۸

۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص ۱۰۰

۳.۱۰ منابع ۱۰۳

۱۱ شبیه سازی چیلین وارز

شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹

۱۰۴

۱.۱۱ حل مسئله ۱۰۴

۲.۱۱ Queue ۱۱۶

۱۲ stacks- objects

بنفشه قلی نژاد - ۱۳۹۸/۱۲/۲۴

۱۲۰

۱.۱۲ استک ۱۲۲

۲.۱۲ objects ۱۲۷

۱۳ Destructor

یاسمین مدنی - ۱۳۹۹/۱/۱۶

۱۳۴

۱.۱۳ عناوین کلی جلسه ۱۳۴

۲.۱۳ دیستراکتور و فاینالایزر ۱۳۵

۳.۱۳ Struct ۱۳۸

۱۴ داده نوع های Value Type و Reference Type

مسعود گلستانه - ۱۳۹۹/۱/۱۸

۱۴۰

۱.۱۴ تفاوت میان داده های value type و reference type در سی شارپ ۱۴۱

۲.۱۴ کپی سطحی (shallow copy) ۱۴۴

۳.۱۴ باکسینگ و آنباکسینگ ۱۴۵

۴.۱۴ Nullable ها در سی شارپ ۱۴۶

۳ ادامه مبحث توابع و پارامترها ۱۴۸
 یاسمن توکلی - ۱۳۹۹/۱/۲۰

۱۶ Exceptions & Operators ۱۴۹
 بیان دیوانی آذر - ۱۳۹۹/۱/۲۵

۱.۱۶ Exceptions ۱۴۹

۲.۱۶ Indexers ۱۵۶

۱۷ Operator Overloading ۱۵۸
 نیکی مجیدی فرد - ۱۳۹۸/۱/۳۰

۱.۱۷ Operator Conversion ۱۶۰

۲.۱۷ Pairs Operator ۱۶۳

۱۸ واسط ها ۱۶۷
 باوان دیوانی آذر - ۱۳۹۹/۲/۱

۱.۱۸ چالش ۱ ۱۶۷

۲.۱۸ چالش ۲ ۱۷۲

۳.۱۸ نکات ۱۷۵

۴.۱۸ خلاصه بندی ۱۷۵

۵.۱۸ تمرینات اضافی ۱۷۶

۱۹ Interface IEnumerable, IDisposable ۱۷۷
 آزاده دارابی مقدم - ۱۳۹۹/۲/۶

۱.۱۹ Generic Interface ۱۷۷

۲.۱۹ Generic Constraints ۱۷۹

۳.۱۹ IDisposable ۱۸۰

۴.۱۹ StreamReader Class ۱۸۱

۵.۱۹ Stopwatch ۱۸۱

۶.۱۹ IEnumerable ۱۸۲

۲۰ چگونگی کارکرد مموری

سعید شهباز زاده - ۱۳۹۹/۲/۸

۱۸۵

۱۸۵ Memorymanager Example ۱.۲۰

۲۱ اشاره گر به تابع

مهدیه نادری - ۱۳۹۸/۲/۱۳

۱۹۴

۱۹۴ اشاره گر به تابع در زبان پایتون ۱.۲۱

۱۹۵ اشاره گر به تابع در زبان سی شارپ ۲.۲۱

جلسه ۲

Functions and Parameters

پریا فصاحت - ۱۳۹۸/۱۱/۱۹

جزوه جلسه ۲م 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 جزوه‌ی 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++ که native نامیده می‌شوند:

کد نوشته‌شده تنها کدی است 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 C#, Java که managed نامیده می‌شود:

این زبان‌ها تبدیل به یک زبان میانی می‌شوند. برای مثال در زبان C# تبدیل به `ILCode`^۱ می‌شود. و در زبان Java؛ `Java Byte Code` کد میانی است. لازم به ذکر است که کدهای میانی به تنهایی قابل اجرا نیستند؛ و نیاز به یک `Run Time Enviroment` نیاز دارند که برای جاوا `Java Run Time Enviroment` و برای سی‌شارپ `DotNet Run Time Enviroment` نام دارد. شایان ذکر است؛ در این زبان‌ها می‌توان کد `Cross Platform` نوشت.

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

- به طور کلی در زبان‌های managed نیازی به از دسترس خارج کردن حافظه مصرف شده نیست. چرا که به علت دارا بودن ویژگی مدیریت حافظه^۲؛ 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.

```

۱ int* nums1 = (int*) malloc(sizeof(int) * 5);
۲ //Malloc
۳
۴ int* nums2 = new int[5];
۵ //new

```

نمونه کد ۱: تفاوت نوشتاری `malloc` و `new`

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

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

```

۱ public class Program
۲ {
۳     int[] nums = ReadFromInput();
۴     static void Main(string[] args)
۵     {
۶         SortNumbers();
۷     }
۸ }

```

نمونه کد ۲: استفاده از متغیر `global`

^۱ intermediate Language

^۲ Garbage Collection

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

```

۱ public class Program
۲ {
۳     static void Main(string[] args)
۴     {
۵         int[] nums = ReadFromInput();
۶         SortNumbers(nums);
۷     }
۸ }
```

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:

```

۱ static void Sort(int[] nums)
۲ {
۳     for(int i=0; i<nums.Length; i++)
۴     {
۵         for(int j=i+1; j<nums.Length; j++)
۶         {
۷             if (nums[i] < nums[j])
۸             {
۹                 Swap(ref nums[i], ref nums[j]);
۱0            }
۱1        }
۱2    }
۱3 }
```

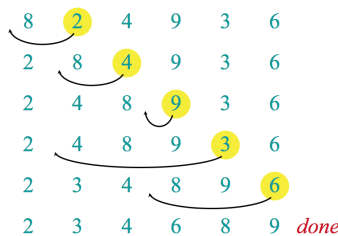
نمونه کد ۴: تابع Sort در زبان C#

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. [۹]

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

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

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



شکل ۱.۲: Sort Function Algorithm

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

```

۱ void sort(std::vector<int>& nums)
۲ {
۳     for (size_t i = 0; i < nums.size(); i++)
۴     {
۵         for (size_t j = i + 1; j < nums.size(); j++)
۶         {
۷             if (nums[i] < nums[j])
۸                 swap(nums[i], nums[j]);
۹         }
۱0    }
۱1 }
```

نمونه کد ۵: تابع Sort در زبان CPP

در نمونه کد فوقانی `size_t` در واقع برای استفاده از سایز یا شمار^۳ آورده شده است. که البته مصارف دیگری [۹] هم دارد.

- و اما شاهد استفاده از `vector` هستیم.

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

^۳count

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

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

ادامه مبحث توابع و پارامترها

نیوشا یقینی - ۱۳۹۸/۱۱/۲۱

جزوه جلسه ۳م مورخ ۱۳۹۸/۱۱/۲۱ درس برنامه‌سازی پیشرفته در زمان مقرر تحویل داده نشد.

جلسه ۴

Method/Function Features

محمد مهدی جاوید - ۱۳۹۸/۱۱/۲۶

۱.۴ متد اورلودینگ

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

۱.۱.۴ سیگنیچر تابع

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

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

شکل ۱.۴: سیگنیچر یک تابع در متد اورلودینگ

۲.۱.۴ متد اورلودینگ با تعداد پارامترهای ورودی متفاوت

تابع اول ۴ پارامتر ورودی دارد.

تابع دوم ۳ پارامتر ورودی دارد.

تابع سوم ۲ پارامتر ورودی دارد.

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

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

شکل ۲.۴: توابع با تعداد پارامترهای ورودی متفاوت

مثال زیر را حل کنید:

۱- در مثال زیر کدام تابع صدا خواهد شد؟

۲- خروجی برنامه چه خواهد بود؟

```

۱ using System;
۲
۳ namespace MethodOverLoading
۴ {
۵     class Program
۶     {
۷         static int Multiply (int x, int y, int z, int i, int j) => x * y * z * i * j; function-1 //
۸         static int Multiply (int x, int y, int z, int i) => x * y * z * i; function-2 //
۹         static int Multiply (int x, int y, int z) => x * y * z; function-3 //
۱0        static int Multiply (int x, int y) => x * y; function-4 //
۱1        static void Main(string[] args)
۱2        {
۱3            Console.WriteLine(Multiply(1, 2, 3));
۱4        }
۱5    }
۱6 }

```

نمونه کد ۶: سوال توابع با تعداد ورودی‌های متفاوت

جواب سوال :

خروجی برنامه : ۶

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

۳.۱.۴ متد اورلودینگ با نوع (تایپ) داده‌ای متفاوت :

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

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

مثال از چند نوع یا تایپ داده‌ای متفاوت :

int - double - string - bool

```
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 صدا زده خواهد شد؟

۲- خروجی برنامه چه خواهد بود؟

```

۱ using System;
۲
۳ namespace MethodOverLoading
۴ {
۵     class Program
۶     {
۷
۸         static string add (string name, string lastName) => name + lastName; 1 function //
۹         static int add (int num1, int num2) => num1 + num2; 2 function //
۱۰        static double add (double num1, double num2) => num1 + num2; 3 function //
۱۱        static void Main(string[] args)
۱۲        {
۱۳            Console.WriteLine(add("Ali", "Reza"));
۱۴        }
۱۵    }
۱۶ }

```

نمونه کد ۷: سوال توابع با تعداد پارامترهای یکسان و نوع داده‌ای متفاوت

جواب سوال :

۱- تابع شماره یک صدا زده خواهد شد.

۲- خروجی برنامه = AliReza خواهد بود.

سوال :

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

جواب سوال :

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

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

۴.۱.۴ متد اورلودینگ در بقیه زبان‌ها:

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

علت چیست؟

در پایتون توابع آبجکت (شیء) 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 خواهند ماند. برای فهم بیشتر این موضوع کد زیر را نگاه کنید.

```

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

```

نمونه کد ۸: متد اورلودینگ در پایتون

به نظر شما خروجی برنامه بالا چه خواهد بود؟

برنامه بالا اکسپشن خواهد داد

اما در چه خطی؟

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

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

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

۲.۴ فرستادن براساس مقدار یا آدرس

Pass by reference VS Pass by value

۱.۲.۴ کلمه کلیدی ref

ref keyword

هنگامی که توابع را صدا می‌زنیم. می‌توانیم 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 داده‌هایی که به طور خودکار به صورت آدرس (reference types) به تابع داده خواهد شد:

۱- آبجکت(شیء) هایی که از کلاس‌ها ساخته می‌شوند.

۲- رشته‌ها (string).

۳- لیست و آرایه.

نوع داده‌هایی که به صورت مقدار (value types) به تابع داده می‌شود:

۱- اعداد (دابل - فلوت - اینتیجر و...)

۲- بولین‌ها (مقادیر صحیح یا غلط)

۳- کارکتر (char)

۴- استراکت‌ها

۵- اینام‌ها (enums)

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

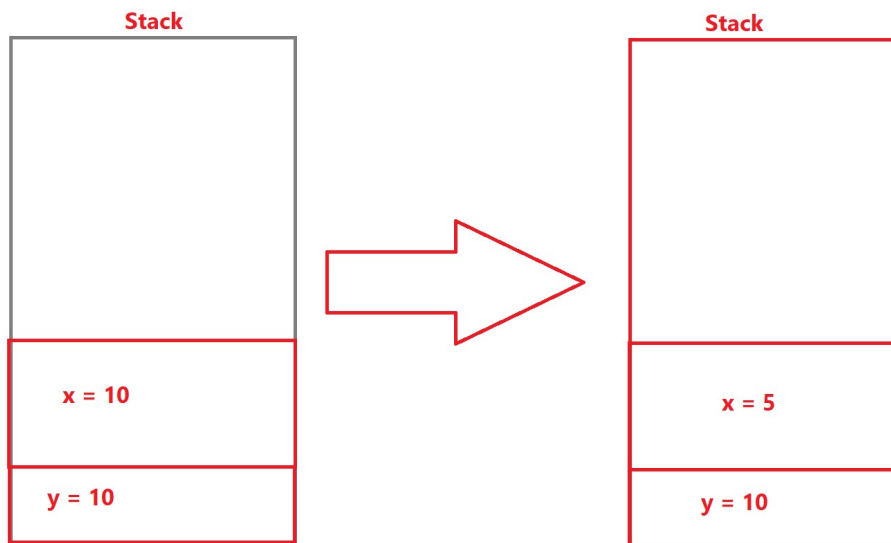
اگر ما هنگام صدا زدن تابع به آن نوع داده‌ای مقدار (value Type) بدهیم. همانند این است. که مقداری که در آدرس متغیر ذخیره شده. را در متغیر ورودی تابع ذخیره خواهد کرد. برای فهم بیشتر این موضوع به توضیحات زیر نگاه کنید.

```
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 ۹: value by pass

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

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



شکل ۵.۴: pass by value

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

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

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

کلمه کلیدی ref (ref keyword) :

هرگاه که ما تابعی را صدا بزنیم. و بخواهیم که آدرس متغیر داده شده به تابع با آدرس پارامتر ورودی تابع یکسان باشد. از این کلمه کلیدی استفاده خواهیم کرد.
اگر آدرس هر دو متغیر یکسان باشند. پس با تغییر هر یک از آن‌ها دیگری نیز تغییر خواهد کرد.

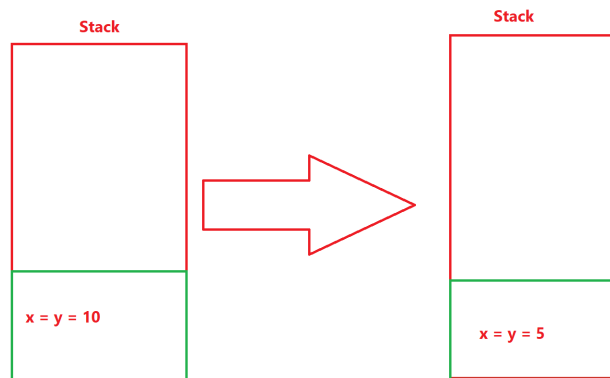
سوال: خروجی برنامه زیر چه خواهد بود؟

```
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 ۱۰: keyword ref

*** نکته‌ای که در این سوال وجود دارد. این است که چون دو متغیر x و y قبل از آن‌ها از کلمه کلیدی ref استفاده شده. پس آدرس استک آن‌ها یکسان خواهد بود. و با تغییر یکی دیگری نیز تغییر خواهد کرد.
پس خروجی برنامه در کنسول عدد ۵ و ۵ خواهد بود. که نشان دهنده این است که هر دو متغیر آدرس یکسانی دارند. و با تغییر یکی دیگری نیز تغییر می‌کند.

برای تفهیم بیشتر این موضوع به عکس زیر نگاه کنید.



شکل ۶.۴: ref keyword

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

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

*** در صورتی که از کلمه کلیدی out استفاده می‌کنیم. حتما باید که در تابع دوم مقداردهی شود.

کلمه کلیدی out (out keyword) :

هرگاه بخواهیم. که متغیری را به عنوان پارامتر ورودی به تابع دیگری بدهیم. و آن را در تابع دوم که صدا زده شده مقدار دهی اولیه کنیم. می‌توانیم از این کلمه کلیدی استفاده کنیم.

*** در تابع اول که این متغیر تعریف می‌شود. به هیچ وجه نباید مقدار دهی اولیه شود.

*** هنگام استفاده از این کلمه کلیدی آدرس استک هر دو متغیر یکسان خواهد شد.

*** یکی از راه‌هایی که بتوانیم چند خروجی از یک تابع داشته باشیم.

برای تفهیم بیشتر این مطلب به کد زیر نگاه کنید.

```

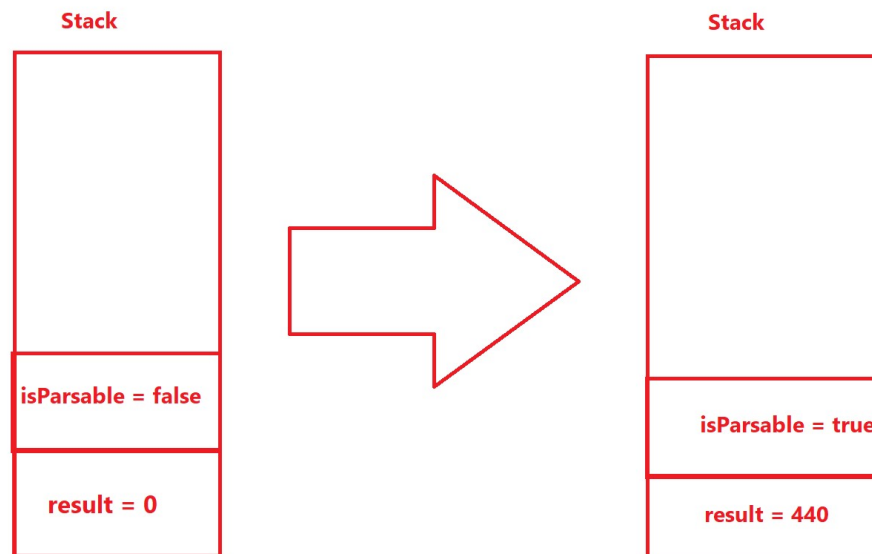
۱ static void Main(string[] args)
۲ {
۳     int result;
۴     bool isParsable = int.TryParse("440", out result);
۵     Console.WriteLine(result + "\t" + isParsable);
۶ }

```

نمونه کد \out keyword :

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

برای تفهیم بیشتر این مطلب به عکس زیر نگاه کنید :



شکل ۷.۴: out keyword

جرا مقدار اولیه متغیر result مساوی با صفر است؟
 داده‌هایی که روی استک تعریف می‌شوند. می‌توانند مقدار پیشفرض بگیرند. این اتفاق در فیلدهای کلاس‌ها یا ساخت آرایه‌ای از نوع اعداد صحیح و... می‌تواند رخ دهد.
 مقدار پیشفرض بعضی از داده‌ها را در پایین می‌آوریم :
 the first constant in the enum : enum
 value of zero : int
 false : bool
 '\0' : char

فرق کلمه کلیدی ref با out :

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

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

۳.۴ توابع با پارامترهای ورودی متغیر (variadic functions) :

C# : ۱.۳.۴

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

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

برای تفهیم بیشتر مطالب به کدهای زیر نگاه کنید.

```

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

```

نمونه کد ۱۲ : 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.

```

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

```

نمونه کد ۱۳: params Keyword

مشکل برنامه بالا در کجاست؟

همانطور که در بالاتر گفته شد. اگر قرار است. که توابع تعداد پارامترهای ورودی متفاوتی داشته باشند. الزما

باید به عنوان آخرین پارامتر به تابع داده شوند.

پس کد زیر نحوه صحیح کد بالا را نمایش می‌دهد.

```

۱ public static int HowManyMatches (int matchNumber, params int[] numbers)
۲ {
۳     int totalMatch = 0;
۴     foreach (int number in numbers)
۵         if (number == matchNumber)
۶             totalMatch++;
۷     return totalMatch;
۸ }
۹ static void Main(string[] args)
۱۰ {
۱۱     int[] numbers = {1, 2, 3, 4, 5, 2, 3, 1,};
۱۲     int match = 2;
۱۳
۱۴     Console.WriteLine(HowManyMatches(match, numbers));
۱۵     Console.WriteLine(HowManyMatches(2, 1, 2, 3, 4, 5, 2, 3, 1));
۱۶ }

```

نمونه کد ۱۴: params keyword

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

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

Python : ۲.۳.۴

در پایتون کافی است. که قبل از آن پارامتر ورودی که قرار است. تعداد متفاوتی از مقادیر را بگیرد. یک ستاره (*) (asterisk) قرار دهیم.

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

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

```

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

```

نمونه کد ۱۵ : 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.

```

۱ languages_and_their_grades = { "Python" : 100 ,
۲                               "C++" : 99 ,
۳                               "C#" : 80 ,
۴                               "Java" : 90 }
۵ grade1 = languages_and_their_grades["Python"]
۶ print(grade1) #100

```

نمونه کد ۱۶ : Python Dictionary

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

```

۱ def sum_of_numbers (*numbers):
۲     result = 0
۳     for number in numbers :
۴         result += number
۵     return result
۶
۷ print(sum_of_numbers(1, 2, 3, 4, 5, 6))

```

نمونه کد ۱۷: Python Variable-length Arguments

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

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

```

۱ def print_max_grade_with_course (**languages_with_grades):
۲     max_grade = 0
۳     course_with_max_grade = None
۴
۵     for course, grade in languages_with_grades.items() :
۶         if grade > max_grade :
۷             max_grade = grade
۸             course_with_max_grade = course
۹
۱۰    print(course_with_max_grade)
۱۱    print(max_grade)
۱۲
۱۳ print_max_grade_with_course( Python = 100 ,
۱۴                               Cpp = 99      ,
۱۵                               JAVA = 95      ,
۱۶                               Cs = 90       )

```

نمونه کد ۱۸: Python Variable-length KeywordArguments

خروجی برنامه بالا Python و 100 خواهد بود.

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

Java : ۳.۳.۴

در جاوا به جای استفاده از کلمه کلیدی params کافی است. که بعد از نوع داده‌ای آن سه نقطه (...) بگذاریم. که به آن Ellipses می‌گویند.

*** نباید بیش از یک پارامتر ورودی با طول داده‌ای متفاوت در سیگنیچر تابع باشد.

*** می‌توان از آرایه نیز استفاده کرد.

برای تفهیم بیشتر مطالب به کد زیر نگاه کنید:

```

۱ public static int Sum (int... numbers) {
۲     int result = 0;
۳     for (int number : numbers)
۴         result += number;
۵     return result;
۶ }
۷ public static void main(String[] args) {
۸     int[] numbers = {1, 2, 3, 4, 5};
۹     System.out.println(Sum(1, 2, 3, 4, 5));
۱۰    System.out.println(Sum(numbers));
۱۱ }

```

نمونه کد ۱۹: Java Variable-length Arguments

خروجی کد بالا عدد ۱۵ خواهد بود.

در تابع Sum آرگومان numbers آرایه‌ای از اعداد صحیح است.

۴.۴ توابع و کلاس‌های عمومی (Generic Functions) :

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

: Csharp

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

برای تفهیم بیشتر مطالب به کدهای زیر نگاه کنید.

توابع عمومی :

```

۱ public static void print <_Type> (params _Type[] collection)
۲ {
۳     foreach (_Type element in collection)
۴         Console.WriteLine(element);
۵ }
۶ static void Main(string[] args)
۷ {
۸     print<string>("Python", "C++", "C#", "Java");
۹     print("Python", "C++", "C#", "Java");
۱۰    print<object>(Reza "Ali, 22, true, 'M');
۱۱ }
```

نمونه کد ۲۰: 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.

```
۱ static void Swap<_Type> (ref _Type element1, ref _Type element2)
۲ {
۳     _Type hold = element1;
۴     element1 = element2;
۵     element2 = hold;
۶ }
۷ static void Main(string[] args)
۸ {
۹     int num1 = 1, num2 = 2;
۱۰ Swap(ref num1, ref num2);
۱۱ Console.WriteLine(num1);
۱۲ Console.WriteLine(num2);
۱۳
۱۴ double dnum1 = 5.1, dnum2 = 5.3;
۱۵ Swap(ref dnum1, ref dnum2);
۱۶ Console.WriteLine(dnum1);
۱۷ Console.WriteLine(dnum2);
۱۸ }
```

نمونه کد ۲۱: Generic Functions

خروجی برنامه بالا چه خواهد بود؟

جواب :

2

1

3.5

1.5

کلاس‌ها یا اینترفیس‌های عمومی (Generic classes and interfaces) :

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

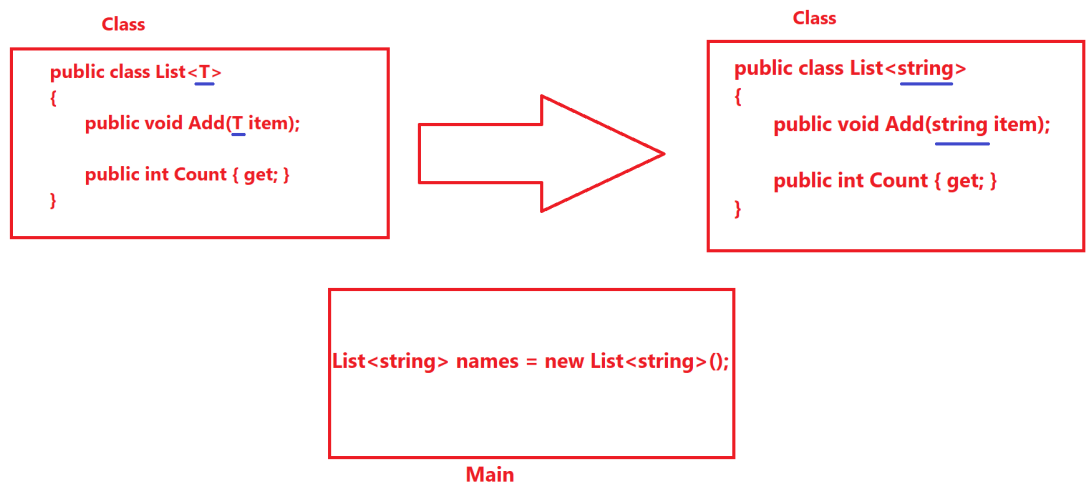
```

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

```

نمونه کد ۲۲: Generic Class

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



شکل ۸.۴: generic class

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

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

نمونه کد ۲۳: Generic Interface

: Java

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

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

نمونه کد ۲۴: Generic Method

خروجی برنامه بالا :

Reza

true

123

M

140

70

2

: Python

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

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

۵.۴ قراردادهای نام گذاری (Naming Conventions) :

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

: Famous Naming Conventions

Pascal Casing : هر گاه که نام یک معرف یا معین کننده هویت (Identifier) از چند کلمه تشکیل شده باشد. و تمامی کلمات حرف اول آن ها به صورت بزرگ Upper-case نوشته شود. به آن پاسکال کیس می گوییم.

به مثال های زیر برای تفهیم بیشتر مطالب نگاه کنید :

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

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

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

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

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

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

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

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

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

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

به مثال‌های زیر برای تفهیم بیشتر مطالب نگاه کنید :

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

: Csharp

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

به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید:

```

۱ public class ClientActivity
۲ {
۳     public void ClearStatistics()
۴     {
۵         .....
۶     }
۷ }
```

نمونه کد ۲۵: Method and Classes Naming convention

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

به مثال زیر برای تفهیم بیشتر مطالب نگاه کنید:

```

۱ public void Func (int studentId)
۲ {
۳     string name = "Ali";
۴ }
```

نمونه کد ۲۶: Method Arguments and Local Variables

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

```

۱ public static const Id = 98521000;
```

نمونه کد ۲۷: Constants or Readonly Variables

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

```

۱ public interface IComparable
۲ {
۳ }

```

نمونه کد ۲۸: Interface

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

```

۱ static void Main(string[] args)
۲ {
۳     .....
۴ }

```

نمونه کد ۲۹: Curly brackets

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

```

۱ public class Student
۲ {
۳     private string _name;
۴     public String Name
۵     {
۶         get => _name;
۷         set => _name = value;
۸     }
۹ }

```

نمونه کد ۳۰: Fields and Properties

: Python

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

```
۱ class Student:
```

نمونه کد ۳۱: Classes in Python

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

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

نمونه کد ۳۲: Functions and Variables

: Java

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

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

نمونه کد ۳۳: Classes and Interfaces

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

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

نمونه کد ۳۴: Methods

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

```
۱ int studentId;
```

نمونه کد ۳۵: Methods

۶.۴ Fold Expression :

از بی‌سوادای ام (محمد مهدی جاوید) در زبان سی‌پلاس‌پلاس عذر خواهی می‌کنم.

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

```

۱ template <typename T>
۲ auto add (T num1, T num2)
۳ {
۴     return num1 + num2;
۵ }
۶ int main(int argc, char const *argv[])
۷ {
۸     cout << add(1, 2) << endl;
۹     cout << add(1.1, 6.1) << endl;}
۱۰    cout << add(true, true) << endl;
۱۱ }
```

نمونه کد ۳۶: Fold Expression

خروجی برنامه بالا به صورت زیر است:

3

7.2

2

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

```

۱ template <typename ..._Types>
۲ auto sum (_Types ...numbers)
۳ {
۴     return (numbers + ...);
۵ }
۶ int main(int argc, char const *argv[])
۷ {
۸     cout << sum(1, 2, 67.5, true, 0001.0) << endl;
۹ }

```

نمونه کد ۳۷: Fold Expression

خروجی برنامه بالا 71.5001 است.

توضیح: پارامتر پک بالا به صورت زیر باز می‌شوند.

((pack1 + pack2) +) + packN

برای تفهیم بیشتر به مثال زیر نیز نگاه کنید:

```

۱ template <typename T>
۲ void print(T arg)
۳ {
۴     cout << arg << endl;
۵ }
۶ template <typename T, typename ..._Types>
۷ void print(T first, _Types ...args)
۸ {
۹     cout << first << endl;
۱۰    print(args...);
۱۱ }
۱۲ int main(int argc, char const *argv[])
۱۳ {
۱۴    print("Mahdi", 17, 023.0, true, 'J');
۱۵ }

```

نمونه کد ۳۸: Fold Expression

خروجی برنامه به صورت زیر خواهد بود:

Mahdi

17

0.023

1

J

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

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

```

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

```

نمونه کد ۳۹: Fold Expression

خروجی برنامه به صورت زیر خواهد بود :

14

3.401

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

استفاده کنیم.

به کد زیر نگاه کنید :

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

۷.۴ نحوه صحیح تقسیم کردن دو عدد بر یکدیگر :

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

نکته :

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

نکته بسیار مهم :

اگر دو عدد صحیح بر یکدیگر تقسیم شوند. مانند Floor Division عمل نخواهد کرد. و عدد به کوچکترین عدد نزدیکش گرد نخواهد شد. الزاما بخش اعشاری جدا خواهد شد. برای تفهیم بیشتر مطالب به مثال‌های زیر نگاه کنید :

```

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

```

نمونه کد ۴۰: Division

خروجی برنامه بالا به ترتیب :

```

1
1.4
1.4

```

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

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

نمونه کد ۴۱: Division

اگر فرض کنیم. که به عدد کوچکترش گرد می‌شود. خروجی اول باید عدد ۲- باشد. اما در صورتی که عدد ۱- است.

خروجی برنامه بالا: -1

-1.4

خواهد بود.

برتری پایتون در عمل تقسیم :

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

```
۱ num1 = int(2)
۲ num2 = int(3)
۳ print(type(num1))
۴ print(type(num2))
۵
۶ result = num1/num2
۷ print(type(result))
۸ print(result)
۹ print(num1/num2)^^I
```

نمونه کد ۴۲ : Division

خروجی برنامه بالا به صورت زیر است :

<class 'int'>

<class 'int'>

<class 'float'>

0.6666666666666666

0.6666666666666666

۸.۴ تمرین کلاسی :

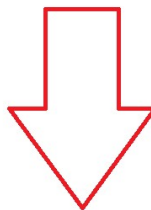
سوال : برنامه‌ای بنویسید که هر تعداد و نوع داده ورودی را بتواند. برعکس کند. و سپس تابع دیگری بنویسید. که آن داده‌ها را نمایش دهد.

Algorithm ۱.۸.۴

اگر ما از اولین پارامتر ورودی شروع کنیم. تا به وسط آن‌ها برسیم. و آن‌ها را با نظر خودشان از ته جابه‌جا کنیم. این کار انجام خواهد شد.
 برای تفهیم بیشتر مثالی کوچک را برای خود می‌زنیم.
 فرض کنید تعداد ورودی‌های ما ۵ تا است. اگر ما اولی را پنجمی و دومی را با چهارمی و سومی را دست نزنیم. این کار انجام خواهد شد.
 ما با ۵ ورودی این کار را دو بار انجام دادیم.

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

۱	۲	۳	۴	۵
---	---	---	---	---



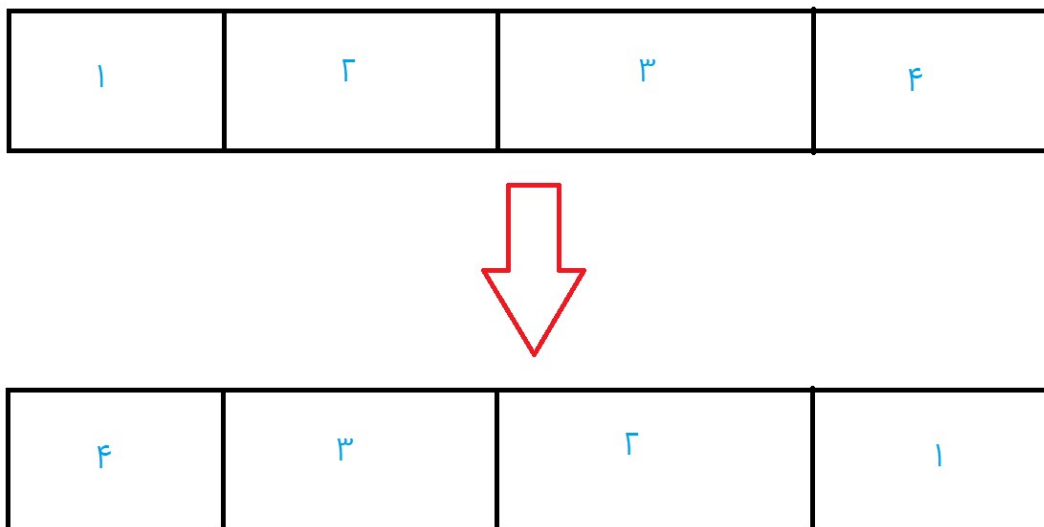
۵	۴	۳	۲	۱
---	---	---	---	---

نتیجه برعکس شده

شکل ۹.۴: Reverse with Odd Number of elements

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

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



تمامی اعضوها برعکس شده‌اند.

شکل ۱۰.۴: Reverse with Even Number of elements

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

Python ۲.۸.۴

```

۱ def reverse(l1):
۲     for idx in range(int(len(l1)/2)):
۳         l1[idx], l1[len(l1)-idx-1] = l1[len(l1)-idx-1], l1[idx]
۴
۵ def output_reverse(*args):
۶     args = list(args)
۷     Reverse(args)
۸     print(args)

```

نمونه کد ۴۳: Reverse and Print

۴ اگر $\text{int}(\text{len}(\text{l1})/2)$ باعث می‌شود. که اگر ۵ ورودی داشته باشیم. این کار دو بار انجام شده. و اگر ۴ ورودی داشته باشیم. نیز این کار دو بار انجام شود. چرا ورودی‌های تابع `output_reverse` را به لیست تبدیل کردیم؟ زیرا tuple ها غیرقابل تغییر هستند.

C# ۳.۸.۴

```

۱ static void Reverse<_Type>(_Type[] list)
۲ {
۳     for (int i=0; i<(list.Count()/2); i++)
۴         (list[i], list[list.Count()-1-i]) = (list[list.Count()-1-i], list[i]);
۵
۶ }
۷ static void OutputReverse<_Type>(params _Type[] args)
۸ {
۹     Reverse(args);
۱۰    System.Console.WriteLine(string.Join(", ", args));

```

نمونه کد ۴۴: Reverse and Print

توضیحات:

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

۴.۸.۴ : C++

```
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 }
```

نمونه کد ۴۵: Reverse and Print

جلسه ۵

Class and Object

روزبه غزوی - ۱۳۹۸/۱۱/۲۸

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

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

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

۱.۵ تعریف کلاس (class)

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

```
۱ public class Circle
۲ {
۳     // Fields & Properties & Methods & Events go here //
۴ }
```

نمونه کد ۴۶: نمونه ای از یک کلاس

در نمونه مثال فوق قبل از کلمه ی `class` از کلمات رزرو شده ی سطوح دسترسی استفاده شده است. و چون در این مورد از کلمه ی `public` استفاده شده ، هر کسی میتواند شی ایی از این کلاس را ایجاد کند. به دنبال کلمه ی `class` نام دلخواه کلاس (`Circle`) قرار گرفته است. باقی مانده ی تعریف یک کلاس بدنه ی آن است که داده ها و رفتار های کلاس در آن تعریف میشود. فیلدها ، 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 مواقعی 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 ایی از یک کلاس یاد میشود. اشیا میتوانند با استفاده از کلمه ی رزرو شده ی `new` که به دنبال آن نام کلاس می آید تعریف شوند :

```
۱ Circle object1 = new Circle();
۲ Circle object2 = new Circle();
۳ Circle object3 = new Circle();
```

نمونه کد ۴۷: 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 مقداری به یک شیء از نوع `Circle` ارجاع پیدا کرده است.

۳.۵ بررسی یک نمونه مثال از کلاس

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

```

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 ۴۸: مثالی از یک کلاس

```

public class MyClass
{
    public string myField = string.Empty;
    public MyClass()
    {
    }
    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}", parameter1, parameter2);
    }
    public int MyAutoImplementedProperty { get; set; }
    private int myPropertyVar;
    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}

```


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

۴.۵ سطح دسترسی (Access Modifiers)

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

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

• Public

• Private

• Protected

• Internal

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

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

۵.۵ فیلد (Field)

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

۶.۵ سازنده (Constructor)

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

```

۱ class MyClass
۲ {
۳     public MyClass()
۴     {
۵     }
۶ }
۷

```

نمونه کد ۴۹: مثالی از کانستراکتر بدون پارامتر

```

۱ class Product
۲ {
۳     public ID Id;
۴     public string Name;
۵     public int Price;
۶     public double Rate;
۷     public Product(ID id , string name, int price, double rate)
۸     {
۹
۱۰         this.Id=id;
۱۱         this.Name=name;
۱۲         this.Price=price;
۱۳         this.Rate=rate;
۱۴     }
۱۵ }

```

نمونه کد ۵۰: مثالی از کانستراکتر پارامتر دار

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

۷.۵ متد (Method)

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

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

```
۱ public void MyMethod(int parameter1, string parameter2)
۲ {
۳     // write your method code here .. //
۴ }
۵
```

نمونه کد ۵۱: مثالی از یک متد در کلاس

۸.۵ صفت (Property)

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

```
۱ private int myPropertyVar;
۲
۳ public int MyProperty
۴ {
۵     get { return myPropertyVar; }
۶     set { myPropertyVar = value; }
۷ }
```

نمونه کد ۵۲: 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 میگيرد.

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

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

```

۱ private int myPropertyVar;
۲
۳ public int MyProperty
۴ {
۵     get {
۶         return myPropertyVar / 2;
۷     }
۸
۹     set {
۱۰
۱۱         if (value > 100)
۱۲             myPropertyVar = 100;
۱۳         else
۱۴             myPropertyVar = value; ;
۱۵     }
۱۶ }

```

نمونه کد ۵۳: مثالی از یک صفت

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

۹.۵ صفات Auto-implemented

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

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

```

۱ 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 ی رزرو شده ی namespace تعریف شود :

```
۱ namespace CSharpTutorials
۲ {
۳     class MyClass
۴     {
۵
۶     }
۷ }
```

نمونه کد ۵۴: نمونه ای از فضای نام

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

جلسه ۶

آرایه‌ها و کلاس

نیکی نزاکتی - ۱۳۹۸/۱۲/۳

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

۱.۶ آرایه‌ها در C++

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

```
int grade=0;
```

با این کار به اندازه‌ی ۴ بایت از فضا برای متغیر grade از حافظه اشغال می‌شود.

حال اگر بخواهیم تعداد بیشتری متغیر از نوع دلخواه داشته باشیم و در عین حال لازم نباشد که بالا را چندین بار تکرار کنیم از آرایه استفاده می‌کنیم.

• Static Array

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

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

• Dynamic and Static Array

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

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

```
delete[] pGrades;
```

واضح است که با توجه به غیر قابل تغییر بودن اندازه‌ی آرایه pGrades اگر به عنوان مثال اندازه‌ی آن ۲۰ باشد، عضو ۲۱ به آرایه قابل اضافه کردن نخواهد بود.

• Vector

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

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

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

یا در هنگام تعریف اولیه به آن مقدار دهی کرد:

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

بعد از مقدار دهی می‌توان به خانه‌های آن دسترسی داشت.

```
vGrades[2]=5;
```

```
int a=vGrades[1];
```

```
int b=vGrades.at(3);
```

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

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

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

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

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

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

نمونه کد ۵۵: 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*` در زبان `c++` است.

• ArrayList

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

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

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

```
gradeList.add(5.1);
```

```
gradeList.add(6.0);
```

```
gradeList.add(7.2);
```

```
gradeList.add(4.8);
```

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

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

در نتیجه مقدار `a` برابر `۴/۸` خواهد بود. دستور زیر عضوی از `gradeList` را که `index` آن ۱ است را

از `gradeList` حذف کرده و مقدار آن را بر می‌گرداند:

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

مقدار `d` برابر `۶/۰` است.

۳.۶ آرایه‌ها در Python

آرایه‌ها در `python` از همه نظر `dynamic` هستند.

```
list=[]
```

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

انجام شود:

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

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

```
list.remove(5)
```

۴.۶ آرایه‌ها در C#

• Dynamic and Static Array

```
int count=5;
int[] list = new int[count];
```

• List

```
using System.collection.Generic;
List<int> myList = new List<int>();
```

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

```
myList.Add(5);
```

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

```
int a=myList[0];
```

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

• 2 Dimensional Array

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

و یا:

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

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

```
۱ for(int i=0; i<myjaggedArray.Length; i++)
۲ {
۳     myjaggedArray[i] = new int[2];
۴ }
```

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 private قرار دارند و بیرون از کلاس نمی‌توان به آن‌ها دسترسی داشت. برای قابل دسترسی 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 ۵۶: کلاس در C++

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

یک pointer است، تعریف کرد:

```

۱ Course(string title, string instName, string instDegree, int credits)
۲ {
۳     this->m_Title = title;
۴     this->m_InstructorName = instName;
۵     this->m_InstructorDegree = instDegree;
۶     this->m_Credits = credits;
۷ }

```

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

```

۱ #include <string>
۲ using namespace std;
۳
۴ class Instructor
۵ {
۶     string m_Name;
۷     string m_Degree;
۸     double m_Rating;
۹
۱۰ public:
۱۱     Instructor(string name, string degree, double rating)
۱۲         : m_Name(name)
۱۳         , m_Degree(degree)
۱۴         , m_Rating(rating)
۱۵     {}
۱۶ };

```

متدهای کلاس مانند دیگر متدها تعریف می‌شوند:

```

۱ string GetInfo()
۲ {
۳     return m_Name + " " + m_Degree;
۴ }

```

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

```

۱  #include<string>
۲  #include "Instructor.hpp"
۳
۴  using namespace std;
۵
۶
۷  class Course
۸  {
۹      string m_Title;
۱۰     Instructor m_Instructor;
۱۱     int m_Credits;
۱۲
۱۳ public:
۱۴     Course(string title, string instName, string instDegree, int credits)
۱۵         : m_Title(title)
۱۶         , m_Instructor(instName, instDegree, 5.3)
۱۷         , m_Credits(credits)
۱۸     {}
۱۹
۲۰     string GetCourseInfo()
۲۱     {
۲۲         string result;
۲۳         result += m_Title;
۲۴         result += "\n";
۲۵         result += m_Instructor.GetInfo();
۲۶         return result;
۲۷     }
۲۸ };

```

در اینجا object m_Instructor 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:

```

۱  string GetCourseInfo()
۲  {
۳      string result;
۴      result += m_Title;
۵      result += "\n";
۶      result += m_Instructor.GetInfo();
۷      return result;
۸  }

```

جلسه ۷

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

امیرحسین سماوات - ۱۳۹۸/۱۲/۰۵

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

۱.۷ Value Type و Reference Type

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

- Value Type
- Reference Type

۲.۷ Value Type

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

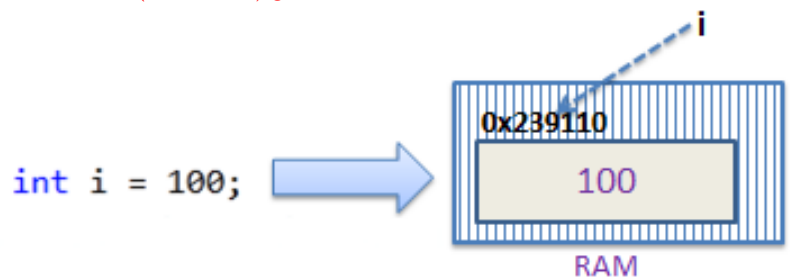
نکته : تمام Type Value ها از فضای نام System.ValueType استفاده میکنند که آن فضای نام هم در فضای نام System.Object قرار دارد.

برای مثال متغیری از نوع int را در نظر بگیرید :

```
int i=100;
```

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

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



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

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

۳.۷ Pass by Value ارسال با مقدار

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

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

```

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

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

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

```
۱ 100
۲ 200
۳ 100
```

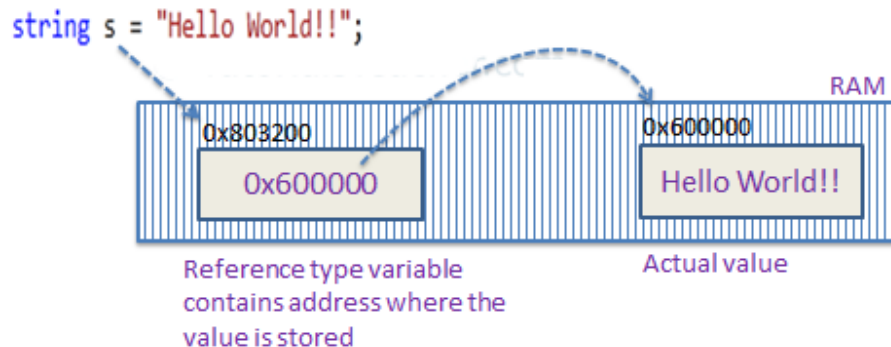
۴.۷ Reference Type

برخلاف Value Type ها ، Reference Type ها مقادیرشان را به صورت مستقیم در خود ذخیره نمی کنند. در عوض آنها آدرس مکانی از حافظه را که مقدار در آن 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 :

```
۱ 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 (`0x۸۰۳۲۰۰`) را برای متغیر "S" انتخاب کرده است. مقداری که در متغیر "S" قرار میگیرد `0x۶۰۰۰۰۰` 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

۵.۷ Pass by Reference ارسال با ارجاع

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

```

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

```

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

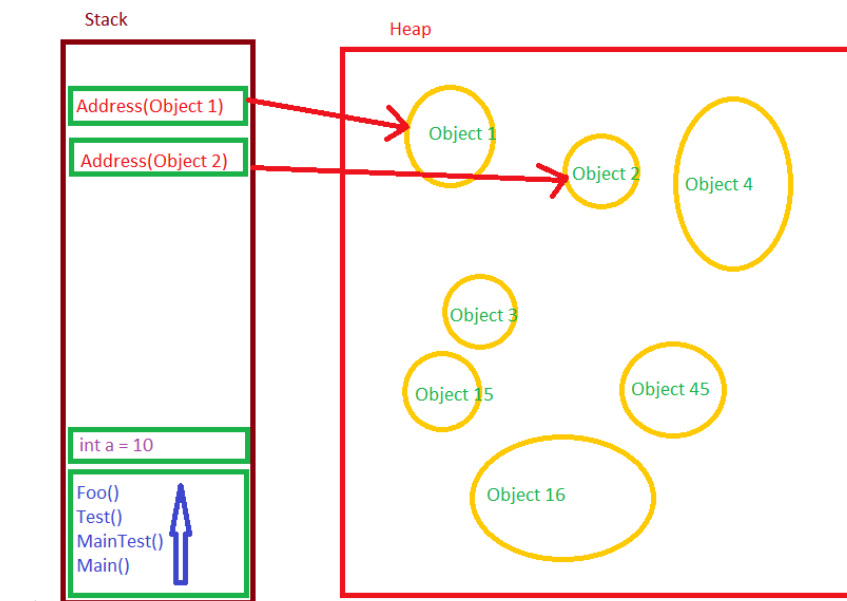
۶.۷ Heap و Stack

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

```

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

```



خب

حالا در عکس بالا میبینید که object ۱ و object ۲ وصل هستند و به 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.

۱.۶.۷ cpp

```

۱  #include <iostream>
۲  using namespace std;
۳  void func(int a)
۴  {
۵      a++;
۶  }
۷  int main()
۸  {
۹      int a=2;
۱۰     func(a);
۱۱     cout<<a;
۱۲ }

```

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

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

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

```

۱  #include <iostream>
۲  using namespace std;
۳  void func(int *a)
۴  {
۵      (*a)++;
۶  }
۷  int main()
۸  {
۹      int a=2;
۱۰     func(&a);
۱۱     cout<<a;
۱۲ }

```

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

رو ..

به عنوان مثال در

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

یعنی ادرس b رو بزار در a

```
int &b=a
```

یعنی با رفرنس اینجا کار داریم.

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

```

۱ #include <iostream>
۲ using namespace std;
۳ void func(int& a)
۴ {
۵     a++;
۶ }
۷ int main()
۸ {
۹     int a=2;
۱۰    func(a);
۱۱    cout<<a;
۱۲ }
```

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

۲.۶.۷ جمع بندی:

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

در صورتی که شما در حال نوشتن برنامه به شکل موازی باشین و قرار باشه که تابع به شکل موازی با جایی که صدا زده شده اجرا بشه در صورت فرستادن متغیر با `reference` 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 `reference` فرستاد هم با `value` ولی فرستادن با مقدار این جور جا ها سریع تره چون به کامپایلر اجازه `optimize` کردن کد رو میده .

۷.۷ Java in Class

مفهوم کلاس:

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

شکل کلی کلاس:

```

۱  Public class Classname{
۲
۳      protected int id;
۴      public String text;
۵      private double spt;
۶
۷
۸      public Classname(int id,String text) {
۹          this.id = id;
۱0         this.text=text;
۱۱     }
۱۲
۱۳     public String functionName(){
۱۴         String data = "AP" "Learning";
۱۵         return data;
۱۶     }
۱۷     public class Main {
۱۸         public static void main(String[] args) {
۱۹             Classname c = new Classname(4,"AP");
۲۰             c.functionName();
۲۱         }
۲۲     }
۲۳ }
```

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

مرسی که برای خواندن این جزوه وقت گذاشتید موفق باشید.

جلسه ۸

کار با فایل در سی شارپ

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

ایتما با چند ویژگی فایل آشنا می شویم (نمونه کد ۵۷).

```
۱ class Program
۲ {
۳     static void Main(string[] args)
۴     {
۵         string stdid = Console.ReadLine();
۶         File.WriteAllText("stdid.txt", stdid + "\n")
۷     }
۸ }
```

نمونه کد ۵۷: مثالی برای `File.WriteAllText`

(نمونه کد ۵۸).


```

۱ class Program
۲ {
۳     static void Main(string[] args)
۴     {
۵         string fileName = "stdid.txt";
۶         string content = File.ReadAllText(fileName);
۷         System.Console.WriteLine(content);
۸     }
۹ }

```

نمونه کد ۵۸: مثالی برای `File.ReadAllText`

۲.۸ برنامه ثبت نام دانش آموزان

۱.۲.۸ انتخاب اسم فایل

قبل از همه چیز اسم فایلی که قرار است با آن کار کنیم را انتخاب می کنیم. (نمونه کد ۵۹).

```

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

```

نمونه کد ۵۹: انتخاب اسم فایل

۲.۲.۸ پیاده سازی اولیه

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

و اگر کاربر چیز دیگری به عنوان ورودی بدهد در خروجی پیغامی برایش چاپ شود. (نمونه کد ۶۰).

```

۱ class Program
۲ {
۳     static void Main(string[] args)
۴     {
۵         if (args.Length != 1 )
۶         {
۷             Usage();
۸             return;
۹         }
۱0
۱1         if (args[0] == "add")
۱2             AddStudent();
۱3         else if (args[0] == "list" )
۱4             PrintStudents();
۱5         else if (args[0] == "find")
۱6             FindStudent(args);
۱7         else
۱8             Usage();
۱9     }
۲۰ }

```

نمونه کد ۶۰: پیاده سازی add,list,find

(نمونه کد ۶۱).

```

۱ private static void Usage()
۲ {
۳     System.Console.WriteLine(" students. register to used be can program This ");
۴     System.Console.WriteLine(" follows: as is syntax usage The ");
۵     System.Console.WriteLine(" [searchstring] add|list|find cs.exe ");
۶ }

```

نمونه کد ۶۱: پیاده سازی متد usage()

۳.۲.۸ AddStudent()

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

(نمونه کد ۶۲).

```

۱ private static void AddStudent()
۲ {
۳     Console.Write(" Id?");
۴     string id = Console.ReadLine();
۵     Console.Write(" Name?");
۶     string name = Console.ReadLine();
۷     File.AppendAllText(StorageFileName, " {id} , {name} \n ");
۸ }

```

نمونه کد ۶۲: پیاده سازی متد AddStudent()

۴.۲.۸ PrintStudents()

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

(نمونه کد ۶۳).

```

۱ private static void PrintStudents()
۲ {
۳     var lines = File.ReadAllLines(StorageFileName);
۴     foreach(var line in lines)
۵         System.Console.WriteLine(line);
۶ }

```

نمونه کد ۶۳: 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 دانش آموزی را نیز وارد کند. و اگر چنین دانش آموزی وجود داشت اسم و شماره دانشجوییش را در خروجی تایپ کند. در صورتی که این دانش آموز 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 ۶۴).

```

۱ class Program
۲ {
۳     static void Main(string[] args)
۴     {
۵         if (args.Length != 1 && args.Length != 2)
۶         {
۷             Usage();
۸             return;
۹         }
۱0
۱1         if (args[0] == "add")
۱2             AddStudent();
۱3         else if (args[0] == "list")
۱4             PrintStudents();
۱5         else if (args[0] == "find")
۱6             FindStudent(args);
۱7         else
۱8             Usage();
۱9     }
۲۰ }

```

نمونه کد ۶۴:

FindStudent() ۵.۲.۸

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

(نمونه کد ۶۵).

```

۱ private static bool FindStudent(string[] args)
۲ {
۳     if (args.Length != 2)
۴         return false;
۵
۶     string searchKey = args[1].ToLower();
۷
۸     string[] lines = File.ReadAllLines(StorageFileName);
۹     bool found = false;
۱0    foreach(string line in lines)
۱1    {
۱2        string[] tokens = line.ToLower().Split(',');
۱3        string id = tokens[0];
۱4        string name = tokens[1];
۱5        if (name == searchKey)
۱6        {
۱7            Console.WriteLine(" {id} : {name} ");
۱8            found = true;
۱9        }
۲0    }
۲1
۲2    if (! found)
۲3        Console.WriteLine(" Found! Not ");
۲۴
۲۵    return true;
۲۶ }

```

نمونه کد ۶۵: 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 ۲ string با کمک `ToLower()` حروف بزرگ هر دو آن‌ها را به حروف کوچک تبدیل می‌کنیم

جلسه ۹

شی گرای و استاتیک^۱

محمدجواد مهدی تبار - ۱۳۹۸/۱۲/۱۲

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

۱.۹ اهداف اصلی این جلسه

- تبدیل کردن یک شی از کلاس به رشته و بالعکس^۲
- تابع های استاتیک و غیر استاتیک
- عمومی یا خصوصی بودن اعضای کلاس نظیر متد ها و ویژگی های کلاس^۳

۲.۹ کلمه های کلیدی مهم

args •

statics •

object oriented and statics^۱

deserialize and serialize^۲

public or private methods or member variable^۳

- `private and public`

- `serialize and deserialize`

۱.۲.۹ args

`args` مخفف کلمه `arguments` می باشد که آرگمان های خط فرمان ^۴ را نشان می دهد. که در واقع پارامتر متد اصلی ^۵ است. تنها و روی که متد اصلی می گیرد `args` می باشد و یا می توان به آن ورودی نداد.

```

۱ static void Main(string[] args)
۲ {
۳ }

```

نمونه کد ۶۶: متد اصلی

همانطور که در نمونه کد ۶۶ می بینید `args` از نوع آرایه ای از رشته ^۶ است. بعد از `build` کردن برنامه و قبل از اجرا شدن برنامه می توان به عنوان پارامتر به متد اصلی داد. در واقع تنها متدی که در طول برنامه اجرا می شود متد اصلی است.

نحوه ی استفاده از args

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

```

dotnet build
dotnet run first second

```

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

برای واضح تر شدن مطلب می توانید از مستند ها استفاده کنید .

• [microsoft doc 1](#)

command line^۴
Main^۵
string array^۶

- [microsoft doc 2](#)

- [dotnet perl](#)

۲.۲.۹ statics

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

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

وقتی از فیلد `static` برای کلاس استفاده می‌کنیم یعنی دیگر نمی‌توانیم از آن کلاس شی بسازیم. در واقع برای دسترسی به متد ها و متغیر های آن کلاس باید از دستور،

`<class-name.variable > or <class-name.method >` استفاده کنیم. در واقع با عملگر `dot` (.) این کار ممکن است.

اگر از فیلد `static` برای کلاس استفاده کنیم تمام اعضای آن کلاس هم باید `static` باشند.

```

۱ static class Example
۲ {
۳     static int Id;
۴     static void Main(string[] args)
۵     {
۶         Example ex = new Example();
۷         above line is incorrect , static class cannot be instantiated //
۸         int X = Example.Id;
۹     }
۱۰ }
```

نمونه کد ۶۷: کلاس استاتیک


```

۱ public class Example
۲ {
۳     public static int X ;
۴     public int Y ;
۵     method static //
۶     public static void print()
۷     {
۸         Console.WriteLine( Example.X);
۹     }
۱۰
۱۱ }
۱۲ public class Program
۱۳ {
۱۴     static void Main(string[] args)
۱۵     {
۱۶         Example.X = 5 ;
۱۷         Example.print();
۱۸     }
۱۹ }

```

نمونه کد ۶۸: متد استاتیک

با توجه به کد نمونه کد ۶۸ 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 شی نیست و برای class می‌باشد.

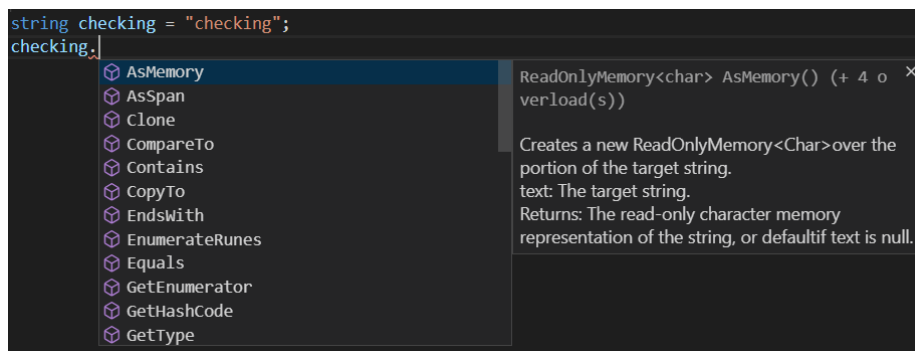
static and non-static method

static method

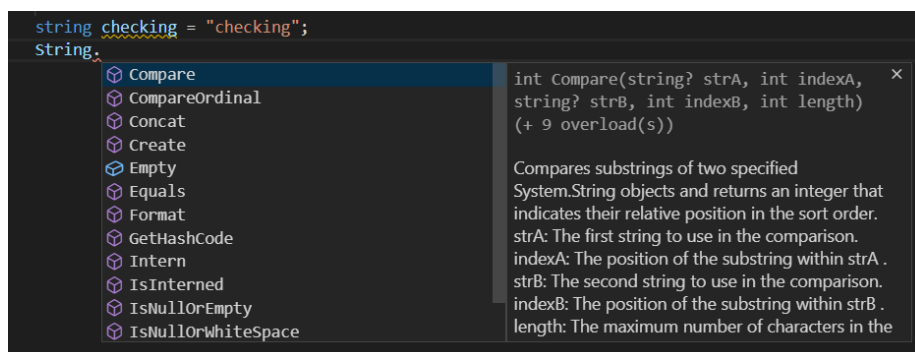
در کل هر وقت از دستور <class-name.> استفاده کنیم پیشنهاد هایی که از auto complete می‌آید همه ی آن‌ها متد های static هستند.

non-static method

هر وقت یک شی از یک کلاس درست می‌کنیم و برای آن شی از dot استفاده می‌کنیم پیشنهاد هایی که از auto complete 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](#)

۳.۲.۹ public or private

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

موقعی که باید از `public` استفاده کرد

از `public` موقعی استفاده می‌کنیم که از همه جای برنامه بخواهیم به آن `method` یا `variable` 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 تغییر دهد.

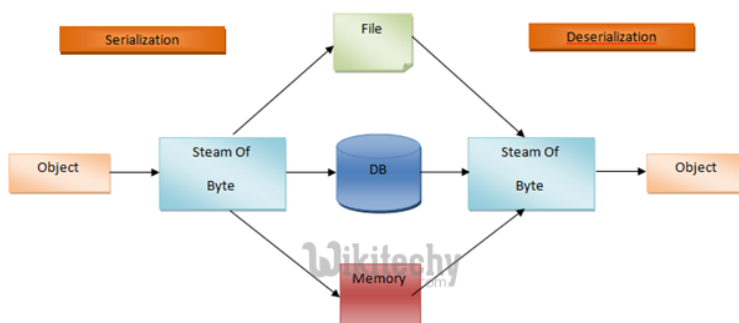
موقعی که باید از `private` استفاده کرد

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 `method` یا `variable` فقط بخواهیم درون کلاس استفاده کنیم و جاهای دیگر کارایی ندارد بهتر است آن را `private` کنیم.
 پس فرق بین `public or private` در امنیت، دسترسی و نیاز به آنها است. برای درک بیشتر `public or private` و کارایی آن وبسایت‌های زیر مفید هستند

- [microsoft doc 1](#)
- [microsoft doc 2](#)
- [dotnetperl](#)

۴.۲.۹ serialize and deserialize

به عمل تبدیل کردن یک شی به بایت برای ذخیره یا انتقال آن شی در فایل یا حافظه `serialize` است. هدف اصلی `serialize` ذخیره کردن آن شی است که هر موقعی که نیاز شد دوباره بتوان آن شی را تولید کرد. به عمل تبدیل کردن بایت های ذخیره شده به شی `deserialize` می گویند. برای `serialize` معمولا شی های یک کلاس رو به صورت رشته ^۷ در یک فایل می ریزند. برای `deserialize` معمولا هر خط فایل تشکیل دهنده ی یک شی است و با زدن حلقه ی `for` روی هر خط آن می توان شی ها را بدست آورد. و معمولا آنها را در آرایه یا لیستی از نوع آن شی می ریزند.



شکل 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 ۶۹: serialize

^۷string

```

۱ private List<Student> Students;
۲ private void desrialize(string file)
۳ {
۴     string[] lines = File.ReadAllLines(file);
۵     foreach(string line in lines)
۶     {
۷         Student s = Student.Parse(line);
۸         Students.Add(s);
۹     }
۱۰ }
۱۱ public static Student Parse(string line)
۱۲ {
۱۳     string[] tokens = line.Split(',');
۱۴     int id = int.Parse(tokens[0]);
۱۵     string name = tokens[1];
۱۶     return new Student(id, name);
۱۷ }
۱۸ }

```

نمونه کد ۷۰: deserialize

همانطور که در نمونه کد ۷۰ مشاهده می‌کنید متد Parse از نوع static است و برای صدا زدن آن باید از دستور <classname.method> استفاده کرد. برای درک بیشتر مفهوم، serialize and deserialize و کارایی آن وبسایت های زیر مفید هستند.

• [microsoft doc](#)

• [csharpcorner](#)

۳.۹ دستورات مهم فایل

File.ReadAllText(string path): محتویات درون فایل را می‌خواند و خروجی آن از نوع رشته است. رشته است که اندیس i ام آن خط i ام فایل است.

File.ReadAllLines(string path): محتویات درون فایل را می‌خواند و خروجی آن از نوع آرایه ای از

File.WriteAllText(string path, string content): یک فایل جدید درست کرده و محتوای داده

شده (ورودی دوم) را در فایل می‌ریزد.

File.WriteAllLines(string path, string[] content): یک فایل جدید درست کرده و محتوای داده

شده (ورودی دوم) را در فایل می‌ریزد به طوری که اندیس *i* ام آن خط *i* ام فایل را تشکیل دهد.

`File.AppendAllText(string path, string content)`: به فایل موجود در `path` محتوای داده شده (ورودی دوم) را اضافه می‌کند. و اگر فایل مورد نظر موجود نباشد آن را ایجاد می‌کند.

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

۴.۹ کد زده شده درون کلاس

```

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

```

نمونه کد ۷۱: نحوه ی استفاده از برنامه

همانطور که در نمونه کد ۷۱ مشاهده می‌کنید، در این جلسه مثال دانش‌جو و واحد درسی انتخاب شده است. در این برنامه کاربر با استفاده از `args` می‌تواند با دستورات

`add` ، `list` ، `find [wanted]` ، دانشجویی را یا ثبت نام بکند یا لیست دانشجو ها را بگیرد و یا دانشجوی خاصی را پیدا کند. پس طبیعتاً دو کلاس داریم یکی از نوع دانش‌جو^۸ و دیگری از نوع واحد درسی^۹ و یک کلاس از نوع برنامه^{۱۰} که متد اصلی در آن است.

Student^۸

Course^۹

Program^{۱۰}

```

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 ۷۲: 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 ۷۳: Course member variables and constructor

در نمونه کد ۷۳ هر بار که یک شی جدید ساخته می‌شود در سازنده^{۱۱} آن متد LoadFile صدا زده می‌شود تا شی‌های ساخته شده در فایل در لیست دانش‌جو‌ها ریخته شود. که همان مفهوم deserialize که در نمونه کد ۷۰ به آن اشاره شده است.

^{۱۱} Constructor

دستور add

```

۱ Program class //
۲ Course math = new Course(name: "Math", code: 101, "Students.csv");
۳ if (args[0] == "add")
۴ {
۵     Student s = Student.GetStudentFromConsole();
۶     math.RegisterStudent(s);
۷     math.StoreInFile();
۸ }
۹ Student class //
۱۰ public static Student GetStudentFromConsole()
۱۱ {
۱۲     System.Console.Write(" Id?");
۱۳     int id = int.Parse(Console.ReadLine());
۱۴     System.Console.Write(" Name?");
۱۵     string name = Console.ReadLine();
۱۶     return new Student(id, name);
۱۷ }
۱۸ Course class //
۱۹ internal void StoreInFile()
۲۰ {
۲۱     List<string> lines = new List<string>();
۲۲     foreach(Student s in this.Students)
۲۳     {
۲۴         lines.Add(s.Serialize());
۲۵     }
۲۶     File.WriteAllLines(StorageFileName, lines);
۲۷ }

```

نمونه کد ۷۴: add student

در نمونه کد ۷۴ دستور add دانش‌جو را از ورودی می‌گیرد که متد آن از نوع static و با دستور Student.GetStudentFromConsole(); می‌توان این کار را انجام داد و او را ثبت نام و در فایل ذخیره می‌کند که همان مفهوم serialize که در نمونه کد ۶۹ اشاره شده می‌باشد.

دستور find

```

۱ Program class //
۲ else if (args[0] == "find" && args.Length == 2)
۳ {
۴     string searchKey = args[1];
۵     string result = Found "Not";
۶     Student s = math.FindStudent(searchKey);
۷     if (null != s)
۸         result = s.Serialize();
۹     Console.WriteLine(result);
۱۰ }
۱۱ Course class //
۱۲ public Student FindStudent(string searchKey)
۱۳ {
۱۴     Student foundStudent = null;
۱۵     searchKey = searchKey.ToLower();
۱۶     foreach(Student s in Students)
۱۷     {
۱۸         if (s.Name.ToLower() == searchKey)
۱۹         {
۲۰             foundStudent = s;
۲۱         }
۲۲     }
۲۳     return foundStudent;
۲۴ }

```

نمونه کد ۷۵: find student

دستور find در لیست دانشجو ها می گردد و اگر دانش جوی مورد نظر را پیدا کند آن دانش جو را برمی گرداند و در غیر این صورت null برمی گرداند.

دستور list

```

۱ Program class //
۲ else if (args[0] == "list" && args.Length == 1)
۳     math.PrintStudents();
۴ Course class //
۵ internal void PrintStudents()
۶ {
۷     foreach(Student s in this.Students)
۸     {
۹         Console.WriteLine(s.serialize());
۱۰     }
۱۱ }

```

نمونه کد ۷۶: list student

دستور list تمام دانش جو های آن فایل را چاپ می کند.

وبسایت‌های `microsoft doc` [؟] `dotnet perl` [؟] `GeeksforGeeks` [؟] کمک
شایانی در یادگیری `csharp` می‌کنند.

جلسه ۱۰

Directory/File) دیرکتوری و فایل

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 پوشه‌ها کارکرد همانند کلاس `File` کلاس، `Directory` یکسری متدهای `static` دارد که به ما اجازه انجام عملیات‌های مختلف بر روی پوشه‌ها را می‌دهند

۱.۱۰ Directoryclass

۱.۱.۱۰ ایجاد پوشه

به وسیله متد `CreateDirectory` نمونه کد ۷۷ می‌توان پوشه جدید در مسیر مشخص شده ایجاد کرد

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

نمونه کد ۷۷: ساخت پوشه در سی‌شارپ

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

۲.۱.۱۰ بررسی وجود یک پوشه

بوسیله متد `Exists` نمونه کد ۷۸ می توان بررسی کرد که یک پوشه وجود دارد یا خیر:

```

۱ if (!Directory.Exists("D:\\Test"))
۲ {
۳     Directory.CreateDirectory("D:\\Test");
۴ }

```

نمونه کد ۷۸: بررسی وجود یک پوشه در سی شارپ

۳.۱.۱۰ دریافت لیست فایل های موجود در یک پوشه

بوسیله دستور `GetFiles` نمونه کد ۷۹ می توان لیست فایل های داخل یک پوشه را بدست آورد. این دستور آرایه ای از رشته ها را بر میگردداند که شامل مسیر و نام فایل های داخل پوشه است:

```

۱ var files = Directory.GetFiles("D:\\MyFolder");
۲
۳ foreach (var file in files)
۴ {
۵     Console.WriteLine(files);
۶ }

```

نمونه کد ۷۹: فایل های یک پوشه در سی شارپ

۴.۱.۱۰ بدست آوردن زیر پوشه های داخل یک پوشه

بوسیله دستور `GetDirectories` نمونه کد ۸۲ می توان لیست پوشه های داخل یک پوشه را بدست آورد:

```

۱ var subDirectories = Directory.GetDirectories("D:\\MyFolder");
۲
۳ foreach (var directory in subDirectories)
۴ {
۵     Console.WriteLine(directory);
۶ }

```

نمونه کد ۸۰: زیر پوشه های یک پوشه در سی شارپ

حال با دانستن کلاس و متدهای بالا قصد نوشتن کدی را داریم نمونه کد ۸۱ که با گرفتن آدرس یک پوشه، فایل های موجود در آن و همچنین فایل های موجود در تمام زیر دایرکتوری های آن را بنویسد

- نکته: برای این کار لازم است با مفهوم تابع بازگشتی^۱ آشنا باشیم

^۱RecursiveFunction

```

۱ static void Main(string[] args)
۲ {
۳     printfileInDir(@"c:\test\...");
۴
۵ }
۶
۷ private static void printfileInDir(string v)
۸ {
۹     var dir=Directory.GetDirectories(v);
۱۰    printfile(v);
۱۱    foreach(var d in dir )
۱۲        printfileInDir(d);
۱۳
۱۴ }
۱۵
۱۶ private static void printfile(string v)
۱۷ {
۱۸     var files =Directory.GetFiles(v);
۱۹     foreach(var f in files)System.Console.WriteLine(f);
۲۰ }

```

نمونه کد ۸۱: تمام فایل های پوشه در سی شارپ

۲.۱۰ یافتن تمام فایل های شامل یک رشته خاص

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

۱.۲.۱۰ منطق اصلی برنامه

```

۱ static void Main(string[] args)
۲ {
۳     DirectoryIndex dirIdx = new DirectoryIndex(dir: @"C:\test\...");
۴     dirIdx.CreateIndex("*.cs");
۵     while(true)
۶     {
۷         System.Console.Write(" Query?");
۸         string q = Console.ReadLine();
۹         if (q == "exit")
۱۰            break;
۱۱
۱۲         List<string> result = dirIdx.Query(q);
۱۳         System.Console.WriteLine(" for found "Files + q);
۱۴         foreach(var f in result)
۱۵             System.Console.WriteLine(f);
۱۶     }

```

نمونه کد ۸۲: منطق برنامه

* کلاس DirectoryIndex نمونه کد ۸۳ محلی برای ایندکس (ذخیره کردن) تمام اطلاعات درون

فایل های یک پوشه میباشد

- * متد `CreateIndex` نمونه کد ۸۴ در کلاس `DirectoryIndex` وظیفه به وجود آوردن ایندکس با یک فیلتر به عنوان پارامتر را دارد
- * متد `Query` نمونه کد ۸۷ در کلاس `DirectoryIndex` وظیفه تطابق رشته ورودی با اطلاعات داخل فایل ها را بر عهده دارد

DirectoryIndex ۲.۲.۱۰

```

۱ class DirectoryIndex
۲ {
۳     private string dir;
۴     private Dictionary<string, List<string>> Index;
۵
۶     public DirectoryIndex(string dir)
۷     {
۸         this.dir = dir;
۹         this.Index = new Dictionary<string, List<string>>();
۱۰    }
۱۱ }

```

نمونه کد ۸۳: `DirectoryIndex`

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

CreateIndex ۳.۲.۱۰

```

۱ internal void CreateIndex(string filter)
۲ {
۳     List<string> allFiles = new List<string>();
۴     GetAllFiles(dir, filter, ref allFiles);
۵
۶     foreach(var file in allFiles)
۷     {
۸         AddToIndex(file);
۹     }
۱۰ }

```

نمونه کد ۸۴: `CreateIndex`

- * متد `GetAllFiles` نمونه کد ۸۵ 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 ۸۱ دارد

* متد `AddToIndex` نمونه کد ۸۶ 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.

• GetAllFiles

```

۱ private void GetAllFiles(string subdir, string filter, ref List<string> allFiles)
۲ {
۳     var files = Directory.GetFiles(subdir, filter);
۴     foreach(var file in files)
۵         allFiles.Add(file);
۶
۷     foreach(var d in Directory.GetDirectories(subdir))
۸         GetAllFiles(d, filter, ref allFiles);
۹ }

```

نمونه کد ۸۵: GetAllFiles

• AddToIndex

```

۱ private void AddToIndex(string file)
۲ {
۳     string [] tokens =
۴     File.ReadAllText(file).Split(' ', ',', '.', '(', ')', '\n', '\r', ';');
۵
۶     foreach(var tok in uniqueTokens)
۷     {
۸         if (!string.IsNullOrEmpty(tok))
۹         {
۱0            if (!this.Index.ContainsKey(tok))
۱1                this.Index.Add(tok, new List<string>());
۱2
۱3            this.Index[tok].Add(file);
۱4        }
۱5    }
۱6 }

```

نمونه کد ۸۶: AddToIndex

۴.۲.۱۰ Query

```

۱ public List<string> Query(string q)
۲ {
۳     if (this.Index.ContainsKey(q))
۴         return this.Index[q];
۵
۶     return new List<string>();
۷ }

```

نمونه کد ۸۷: Query

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

۳.۱۰ منابع

[?] `microsoft docs` [?] `C# Notes for Professionals`

جلسه ۱۱

شبیه سازی چیلین وارز

شهرزاد آذری آزاد - ۱۳۹۸/۱۲/۱۹

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

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

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

- enum
- queue
- switch

۱.۱۱ حل مسئله

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

```

۱ static void Main(string[] args)
۲ {
۳     Table t = new Table(rows: 10, cols: 8);
۴     t.Player1 = new Player(row: 2, col: 3, table: t);
۵     t.Player2 = new Player(row: 8, col: 5, table: t);
۶     t.Print();
۷ }

```

نمونه کد ۸۸: کد اولیه در program.cs

به این منظور، برای شبیه سازی صفحه ی چیلین وارز مشابه نمونه کد ۸۸ ابتدا از کلاس های table و player و متود های داخل هر یک، مثل print استفاده می کنیم و سپس هر یک از این کلاس ها و متود ها را پیاده سازی می کنیم.

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

را `Generate type 'table' -> Genarate class 'table' in new file`

انتخاب می کنیم.

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

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

برای نمایش صفحه ی بازی یک آرایه ی دو بعدی از کاراکتر ها به عنوان ممبر ورایبل می سازیم و آن را در کانستراکتور، همانطور که در نمونه کد ۸۹ می بینید، نیو می کنیم :

```

۱ internal class Table
۲ {
۳     public int Rows;
۴     public int Cols;
۵     private char[,] Board;
۶
۷     public Table(int rows, int cols)
۸     {
۹         this.Rows = rows;
۱۰        this.Cols = cols;
۱۱        this.Board = new char[rows, cols];
۱۲    }
۱۳ }

```

نمونه کد ۸۹: 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 کاراکترها برای مشخص کردن دیوار، سلول خالی و بازیکن استفاده کنیم. به این منظور از Enum استفاده می کنیم.

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

```

۱ enum CellType
۲ {
۳     Empty, Wall, Player
۴ }

```

نمونه کد ۹۰: 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:

```

۱ private void FillBoard(CellType empty)
۲ {
۳     for(int i=0; i<Rows; i++)
۴         for(int j=0; j<Cols; j++)
۵         Board[i,j] = CellType.Empty;
۶ }

```

نمونه کد ۹۱: متود FillBoard در کلاس Table

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

```

۱ private void MakeWalls()
۲ {
۳     for(int i=0; i<Rows; i++)
۴     {
۵         Board[i, 0] = CellType.Wall;
۶         Board[i, Cols-1] = CellType.Wall;
۷     }
۸     for(int i=0; i<Cols; i++)
۹     {
۱۰        Board[0, i] = CellType.Wall;
۱۱        Board[Rows-1, i] = CellType.Wall;
۱۲    }
۱۳ }

```

نمونه کد ۹۲: 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 محدوده ی بازی خود را بداند.
- دوم آن که بعد از تعریف هر بازیکن در خانه ای از صفحه ، آن خانه، خانه ی بازیکن باشد و دیگر خالی محسوب نشود.

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

```

۱ internal class Player
۲ {
۳     public int Row;
۴     public int Col;
۵
۶     private Table Table;
۷     public Player(int row, int col, Table table)
۸     {
۹         this.Table = table;
۱۰        if (row < table.Rows && col < table.Cols)
۱۱        {
۱۲            this.Row = row;
۱۳            this.Col = col;
۱۴        }
۱۵        else
۱۶            Console.WriteLine("Error");
۱۷    }
۱۸ }

```

نمونه کد ۹۳: کلاس player

برای این که بتوانیم از table.Rows و table.Cols استفاده کنیم، باید متغیرهای 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 ۹۴ تعریف می کنیم:

```

۱ public int Rows {get; private set;}
۲ public int Cols {get; private set;}

```

نمونه کد ۹۴: 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:

```

۱ private Player _Player1;
۲
۳ public Player Player1
۴ {
۵     get { return _Player1; }
۶     set {
۷         this._Player1 = value;
۸         Board[value.Row, value.Col] = CellType.Player;
۹     }
۱۰ }

```

نمونه کد ۹۵: تعریف گت و ست برای Player در کلاس Table

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

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

۹۶ می نویسیم:

```

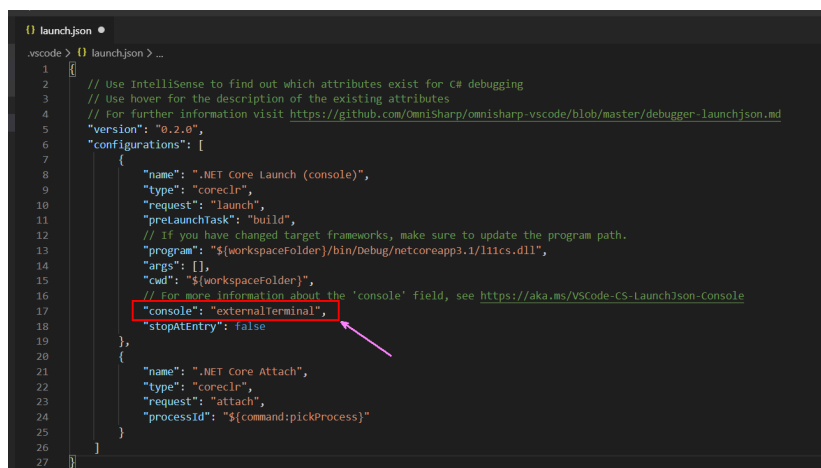
۱ public void Print()
۲ {
۳     Console.Clear();
۴     PrintFirstLine();
۵     for (int i =0; i<Rows; i++)
۶     {
۷         Console.Write(i + " ");
۸         for (int j=0; j<Cols; j++)
۹             PrintCell(i,j);
۱0        Console.WriteLine();
۱۱    }
۱۲ }

```

نمونه کد ۹۶: متود Print در کلاس Table

برای نمایش خروجی کد خود در اکسترنال ترمینال ، میتوان در فایل launch.json مقدار کنسول را

مشابه ۱.۱۱ از اینترنال کنسول به اکسترنال ترمینال تغییر داد.



شکل ۱.۱۱: externalTerminal

متود PrintFirstLine را به مانند نمونه کد ۹۷ پیاده سازی می کنیم:

```

۱ private void PrintFirstLine()
۲ {
۳     Console.Write(" ");
۴     for (int i = 0; i < Cols; i++)
۵         Console.Write(i + " ");
۶     Console.WriteLine();
۷ }

```

نمونه کد ۹۷: متود PrintFirstLine در کلاس Table

متود PrintCell را نیز به شکل نمونه کد ۹۸ می نویسیم:

```

۱ private void PrintCell(int i, int j)
۲ {
۳     CellType ct = Board[i,j];
۴     switch(ct)
۵     {
۶         case CellType.Empty:
۷             Console.Write('-');
۸             break;
۹
۱۰        case CellType.Player:
۱۱            int playerNumber = GetPlayer(i, j);
۱۲            Console.Write(playerNumber);
۱۳            break;
۱۴
۱۵        case CellType.Wall:
۱۶            Console.Write('w');
۱۷            break;
۱۸    }
۱۹    Console.Write(' ');
۲۰ }

```

نمونه کد ۹۸: 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:

```

۱ private int GetPlayer(int r, int c)
۲ {
۳     int playerNumber = -1;
۴
۵     if (Player1.Col == c && Player1.Row == r)
۶         playerNumber = 1;
۷     else if (Player2.Col == c && Player2.Row == r)
۸         playerNumber = 2;
۹     else
۱0        Console.WriteLine("ERROR");
۱۱
۱۲    return playerNumber;
۱۳ }

```

نمونه کد ۹۹: متود GetPlayer در کلاس Table

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

```

C:\Program Files\dotnet\dotnet.exe
 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 ۱۰۰ دستوراتی اضافه میکنیم تا صفحه ی بازی را رنگی چاپ کند.


```

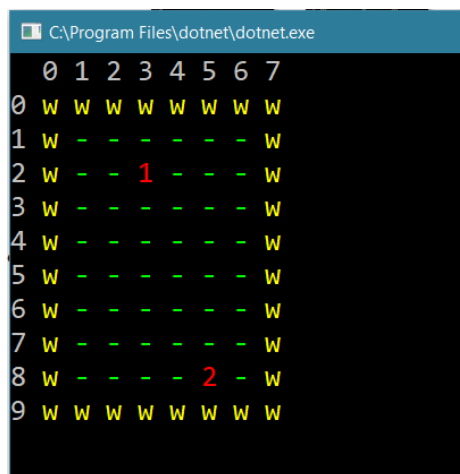
۱ private void PrintCell(int i, int j)
۲ {
۳     var color = Console.ForegroundColor;
۴     CellType ct = Board[i,j];
۵     switch(ct)
۶     {
۷         case CellType.Empty:
۸             Console.ForegroundColor = ConsoleColor.Green;
۹             Console.Write('-');
۱0            break;
۱۱
۱۲         case CellType.Player:
۱۳             Console.ForegroundColor = ConsoleColor.Red;
۱۴             int playerNumber = GetPlayer(i, j);
۱۵             Console.Write(playerNumber);
۱۶             break;
۱۷
۱۸         case CellType.Wall:
۱۹             Console.ForegroundColor = ConsoleColor.Yellow;
۲۰             Console.Write('w');
۲۱             break;
۲۲
۲۳         case CellType.Visiting:
۲۴             Console.ForegroundColor = ConsoleColor.Cyan;
۲۵             Console.Write('o');
۲۶             break;
۲۷     }
۲۸     Console.Write(' ');
۲۹     Console.ForegroundColor = color;
۳۰ }

```

نمونه کد ۱۰۰: 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 داکيومنت ماکروسافت استفاده کنید [۹].

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



شکل ۳.۱۱: اجرای Main بعد از نوشتن نمونه کد ۱۰۰

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

```

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

```

نمونه کد ۱۰۱: استفاده از کلاس Vector

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

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

```

۱ internal void Move(Vector v)
۲ {
۳     this.Row += v.row;
۴     this.Col += v.col;
۵     Table.Update();
۶ }

```

نمونه کد ۱۰۲: متود Move در کلاس Player

توجه کنید که Row ، Col در کلاس وکتور، باید به صورت پابلیک تعریف شده باشند تا بتوان در نمونه کد ۱۰۲ از آن ها استفاده کرد.

حالا متود Table.Update را نیز در کلاس Table مطابق نمونه کد ۱۰۳ پیاده سازی می کنیم:

```

۱ public void Update()
۲ {
۳     for(int i=1; i<Rows-1; i++)
۴     for(int j=1; j<Cols-1; j++)
۵     {
۶         if ((Player1.Row == i && Player1.Col == j) ||
۷             (Player2.Row == i && Player2.Col == j) )
۸             Board[i,j] = CellType.Player;
۹         else
۱0            Board[i,j] = CellType.Empty;
۱۱     }
۱۲ }

```

نمونه کد ۱۰۳: متود Update در کلاس Table

به حلقه ی خود در Main مطابق نمونه کد ۱۰۴ دستوراتی اضافه می کنیم:

```

۱ while ('q' != Console.ReadKey().KeyChar)
۲ {
۳     t.Player1.Move(v);
۴     v.Negate();
۵     t.Player2.Move(v);
۶     t.Print();
۷ }

```

نمونه کد ۱۰۴: تغییر Main در Program.cs

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

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

متود `Negate` را در کلاس وکتور می نویسیم:

```

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

```

نمونه کد ۱۰۵: متود `Negate` در کلاس `Vector`

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

```

۱ public void BFSVisit(int i, int j)
۲ {
۳     Queue<Vector> toVisit = new Queue<Vector>();
۴     toVisit.Enqueue(new Vector(i, j));
۵
۶     while (toVisit.Count != 0)
۷     {
۸         var v = toVisit.Dequeue();
۹         Board[v.row, v.col] = CellType.Visiting;
۱0        foreach(Vector n in GetNeighbors(v))
۱1        {
۱2            if (Board[n.row, n.col] != CellType.Visiting)
۱3                toVisit.Enqueue(n);
۱4        }
۱5        Print();
۱6        Console.ReadKey();
۱7    }
۱8 }

```

نمونه کد ۱۰۶: متود `BFSVisit` در کلاس `Table`

همانطور که متوجه شدید، برای به کار بردن این قطعه از کد، باید `Visiting` را به `Enum cellType`

اضافه کنیم. متود `GetNeighbors` را نیز به صورت زیر پیاده سازی می کنیم:

```

۱ private List<Vector> GetNeighbors(Vector v)
۲ {
۳     List<Vector> neighbors= new List<Vector>();
۴     if (v.row-1 > 0)
۵         neighbors.Add(new Vector(v.row-1, v.col));
۶     if (v.col-1 > 0)
۷         neighbors.Add(new Vector(v.row, v.col-1));
۸     if (v.row+1 < Rows-1)
۹         neighbors.Add(new Vector(v.row+1, v.col));
۱۰    if (v.col+1 < Cols-1)
۱۱        neighbors.Add(new Vector(v.row, v.col+1));
۱۲
۱۳    return neighbors;
۱۴ }

```

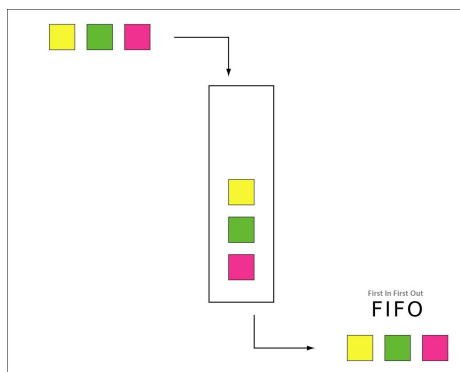
نمونه کد ۱۰۷: متود GetNeighbors در کلاس Table

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

۲.۱۱ Queue

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

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



شکل ۴.۱۱: first in first out

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

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

حالا در متود PrintCell کیس ویزیتینگ را اضافه می کنیم:

```

۱ case CellType.Visiting:
۲     Console.ForegroundColor = ConsoleColor.Cyan;
۳     Console.Write('o');
۴     break;

```

نمونه کد ۱۰۸: تغییر متود PrintCell در کلاس Table

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



شکل ۵.۱۱: اجرای متود BFSVisit

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

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

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

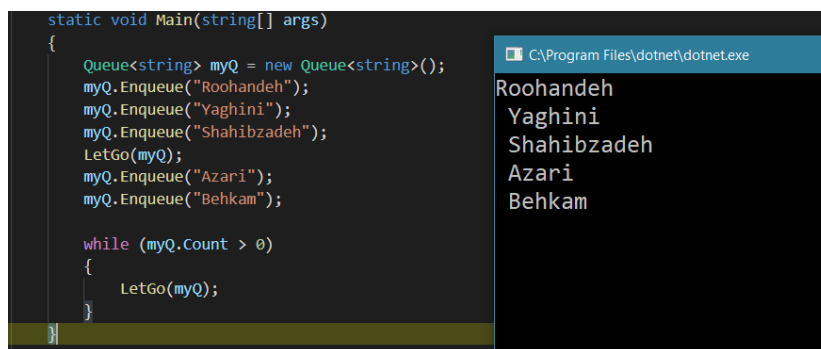
```

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 ۱۰۹: مثالی برای درک بهتر کیو در سی شارپ

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



```

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);
    }
}

```

C:\Program Files\dotnet\dotnet.exe

Roohandeh
Yaghini
Shahibzadeh
Azari
Behkam

شکل ۶.۱۱: خروجی ۱۰۹

برای یادگیری بهتر، می توانید داکيومنت ماکروسافت را در زمینه ی کیو بخوانید [۹].

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

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

برای اطلاعات بیشتر درباره ی کیو در پایتون، می توانید داکيومنت پایتون را مطالعه کنید [۹].

جلسه ۱۲

stacks- objects

بنفشه قلی نژاد - ۱۳۹۸/۱۲/۲۴

جزوه جلسه ۱۲ ام مورخ ۱۳۹۸/۱۲/۲۴

برنامه‌سازی پیشرفته

stacks-objects

۱.۱۲ استک

۱.۱.۱۲ استک چیست؟

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

استک در `csharp` اینگونه تعریف میشود:

```

۱ using System.Collections.Generic;
۲
۳ public class Program
۴ {
۵     static void Main(string[] args)
۶     {
۷         Stack<string> stack = new Stack<string>();
۸     }
۹ }
```

نمونه کد ۱۱۰: تعریف کلاس استک

۲.۱.۱۲ call stack

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

به این نمونه کد توجه کنید:

```

۱ using System;
۲
۳ namespace c16
۴ {
۵     class Program
۶     {
۷         static void Main(string[] args)
۸         {
۹             MethodA();
۱۰        }
۱۱        static void MethodA()
۱۲        {
۱۳            Console.WriteLine(Before" A: "In);
۱۴            MethodB();
۱۵            Console.WriteLine(After" A: "In);
۱۶        }
۱۷
۱۸        static void MethodB()
۱۹        {
۲۰            Console.WriteLine(Before" B: "In);
۲۱            MethodC();
۲۲            Console.WriteLine(After" B: "In);
۲۳        }
۲۴
۲۵        static void MethodC()
۲۶        {
۲۷            Console.WriteLine(C" "In);
۲۸        }
۲۹
۳۰    }
۳۱ }
۳۲

```

نمونه کد ۱۱۱ : call stack

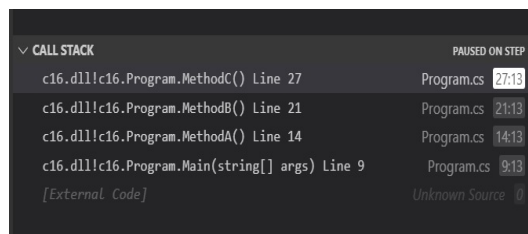
در این مثال، زمانی که چند متد تودرتو در یک دیگر صدا زده می شوند، تا زمانی که متد باز شده بسته نشود، نمی توان تا انتهای برنامه پیش رفت.

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

```

In A:Before
In B before
In C:
In B after
In C after

```



شکل ۱۰.۱۲: call stack

در صورت دیباگ کردن خط به خط این برنامه، در قسمت Call stack می بینیم که آخرین متد فراخوانی شده، بالا تر از همه قرار دارد.

همان طور که در تصویر ۱۰.۱۲ می بینید، دقیقاً مانند کتاب هایی که به ترتیب روی یک دیگر اند، این متد ها نیز به ترتیب روی هم فراخوانی شده اند.

۳.۱.۱۲ stack's API

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

در API استک، بیشترین و کاربردی ترین متد ها شامل:

- push()
- pop()

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

```

۱ using System;
۲ using System.Collections.Generic;
۳ class program
۴ {
۵     static void Main(string[] args)
۶     {
۷         Stack<string> stack = new Stack<string>();
۸         stack.Push("Main");
۹         stack.Push("MethodA");
۱۰        stack.Push("MethodB");
۱۱        stack.Push("MethodC");
۱۲
۱۳        while (stack.Count > 0)
۱۴            Console.WriteLine(stack.Pop());
۱۵    }
۱۶ }

```

نمونه کد ۱۱۲:

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 برمی گرداند و چاپ می کند! و در این حلقه به اندازه ی استک، به ترتیب این عمل را اجرا می کند تا همه ی اعضا چاپ بشوند.

۴.۱.۱۲ کاربرد

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

```

۱ program class Program
۲     private static bool IsBalanced(string s)
۳     {
۴         Stack<char> stack = new Stack<char>();
۵         foreach(char c in s)
۶         {
۷             if (c == '(' || c == '[')
۸                 stack.Push(c);
۹
۱0            if (c == ')' || c == ']')
۱1                if ( !stack.TryPop(out char top) || !IsCompatible(c, top))
۱2                    return false;
۱3        }
۱4        return stack.Count == 0;
۱5    }
۱6    private static bool IsCompatible(char c, char t)
۱7    {
۱8        (c == ')') && t == '('
۱9        || (c == ']' && t == '[') ;
۲0    }
۲1
۲2    static void Main(string[] args)
۲3    {
۲4        string a = )" (a+c) / (a+b) * (a-b) [( * "(a+b);
۲5        string b = )" (a+c) / (a+b) * (a-b) ( * "(a+b;
۲6        Console.WriteLine(IsBalanced(a));
۲7        Console.WriteLine(IsBalanced(b));
۲8    }
۲9 }

```

نمونه کد ۱۱۳: پرانتز‌گذاری صحیح

خروجی:

```

True
False

```

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

و اگر بسته میشود، با کمک متد `TryPop()`^۱ و متد `isCompatible()` که هردو بولین هستند، پرانتز ها را مقایسه می کند و اگر تا زمانی که اندازه ی استک صفر شود شرط برقرار باشد، نحوه ی پرانتز گذاری ها صحیح است.

متد `isCompatible()` نیز آخرین داده ی استک را با پرانتز بسته شده مقایسه می کند و اگر ترتیب آن صحیح بود، ارزش این متد `true` می باشد.

۲.۱۲ objects

۱.۲.۱۲ تعریف

همان طور که میدانیم، می توان یک کلاس را پیاده سازی کرده و شیئی را جداگانه برایش تعریف کرد. اما زمانی که بخواهیم متد ها و یا ویژگی هایی که برای شی بخصوص تعریف کردیم را پیاده سازی کنیم ، برای هر نوع شیئی علاوه بر ویژگی های به خصوص آن، یک سری متد های عام دیگر مانند: `lEquals()` , `Tostring()`, `GetType()` , `GetHashCode` معرفی می کند. سوال این است که چرا برای هر شیئی از انواع مختلف که می نویسیم، این توابع مشترک هستند؟

در سی شارپ علاوه بر شیئی که به طور عام تعریف میشود، یک کلاس `object` به طور خاص تعریف شده که می توان آن را به صورت یک شی جدا نوشت، که شامل ویژگی ها و متد های بالا است. در واقع هر شیئی که ما تعریف می کنیم، به نوعی یک نوع `object` است.

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

تعریف آبجکت در `csharp`

```

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

```

نمونه کد ۱۱۴: تعریف آبجکت در سی شارپ

^۱ این متد در صورتی که داده در استک موجود باشد، داده با 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 متد هایی که در کلاس آبجکت به صورت عمومی تعریف شده اند را جور دیگری تغییر داد، به صورتی که بتوان چیزی را که می خواهیم از آن ها بدست بیاروریم و کارکرد آن را تغییر دهیم، به اصطلاح آن متد را `override` کردیم.

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

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

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:

```

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

نمونه کد ۱۱۵: `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 منحصراً 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

```

۱ internal class Student
۲ {
۳     private string name;
۴     private int id;
۵
۶     public Student(string name, int id)
۷     {
۸         this.name = name;
۹         this.id = id;
۱۰    }
۱۱
۱۲    public override bool Equals(object obj)
۱۳    {
۱۴        if (! (obj is Student))
۱۵            return false;
۱۶        Student other = (Student) obj ;
۱۷        Student other = obj as Student;
۱۸
۱۹        return this.Id == other.Id ;
۲۰    }

```

نمونه کد ۱۱۶: Equals

در این قسمت ما تعریف کردیم که اگر شی ما دانش آموز نبود، آن را **false** برگرداند و در صورت دانش آموز بودن، شی را به قالب کلاس دانش آموز دریاورد، و اگر تمام ویژگی ها برابر باشند، مقدار برگردانده ی این متد **true** باشد. در صورت تعریف این متد، پاسخ کد ۱۱۵

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

اما راه بهتری نیز برای این کار وجود دارد!

در صورت به قالب درآوردن شی، به کلاسی که می خواهیم، در صورت **null** بودن، استثنایی از نوع **nullrefer-ence exception** پرتاب می کند.

نمونه کد ۱۱۵ 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:

```

۱ public override bool Equals(object obj)
۲ {
۳     if ((obj is Student))
۴         return this.Id == (obj as Student).Id && this.Name == (obj as Student).Name;
۵
۶     if ((obj is string))
۷         return this.Name == (obj as string);
۸
۹     if ((obj is int))
۱۰        return this.Id == (int) obj;
۱۱
۱۲        return false;
۱۳    }

```

نمونه کد ۱۱۷: Equals

همان طور که در نمونه کد ۱۱۷ می بینید، در صورت نوشتن `obj as (...)` آن شی را از نوع خاص تعیین شده در نظر میگیرد. به طوری که اگر null باشد، برنامه متوقف نمی شود، همان طور که در خط سوم و چهارم، و پنجم و ششم مشاهده می کنید، می توان ویژگی یک شی از کلاس بخصوص را با رشته ی حرفی یا عدد نیز مقایسه کند. (البته برای int نمی توان از `obj as int` استفاده کرد. چرا که از نوع ساختار یا `struct` است و از جنس کلاس نیست. به همین علت باید آن را قالب بندی یا cast کرد)

با تغییر کارکرد تابع Equals به صورت نمونه کد ۱۱۷ حاصل کار به صورت زیر است:

```

۱ public static void main
۲ {
۳     Student s = new Student(name: "Ali", id:98521231);
۴     Student s1 = new Student(name: "Zahra", id:77521231);
۵     //
۶     Console.WriteLine(s.Equals(s1));
۷     Console.WriteLine(s.Equals(s2));
۸     Console.WriteLine(s.Equals(98521231));
۹     Console.WriteLine(s.Equals("Ali"));
۱۰    Console.WriteLine(s.Equals("Zahra"));
۱۱    Console.WriteLine(s.Equals(98989898));
۱۲ }

```

نمونه کد ۱۱۸: Equals

جواب به ترتیب:

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

۴.۲.۱۲ tostring

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

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

نمونه کد ۱۱۹: ToString

این برنامه در صورت اجرا شدن **System.object** را چاپ می کند. می توان با تغییر کارکرد این متد، شی از نوع کلاس خاص را با فراخوانی متد ToString تبدیل به رشته ی حرفی کرد. مثلاً این متد را می توان در کلاس دانش آموز [۱۱۶] override کرد.

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

نمونه کد ۱۲۰: ToString

در متد main:

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

نمونه کد ۱۲۱: Equals

و به ازای نمونه کد ۱۲۱ رشته ی حرفی

Ali :98532445

چاپ میشود

۵.۲.۱۲ GetHashCode

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

در دیکشنری، برای بیشتر شدن سرعت لازم این است که در صورت برابر بودن دو شی، یک کد مساوی برگرداند و در صورت نبود، حداقل امکان مساوی نباشند.

به همین علت، معمولا متد **GetHashCode** و **Equals** با هم پیاده سازی می شوند. در این صورت، می توان از خود شی از نوع کلاس خاص، به عنوان کلید در دیکشنری استفاده کنیم.

برای مثال این متد را برای کلاس دانش آموز [۱۱۶] override می کنیم.

```

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

```

نمونه کد ۱۲۲: **getHashCode**

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

برای مثال:

```
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 ۱۳۳: GetHashCode()

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

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

جلسه ۱۳

Destructor

یاسمین مدنی - ۱۳۹۹/۱/۱۶

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

۱.۱۳ عناوین کلی جلسه

موضوعات مطرح شده در این جلسه به شرح زیر است:

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

۲.۱۳ دیستراکتور و فاینالایزر

۱.۲.۱۳ دیستراکتور

هنگام نوشتن یک برنامه در راستای کنترل کردن منابع استفاده شده مانند فایل یا غیره همچنین هنگام استفاده مستقیم از حافظه نیازمند استفاده از 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 ۱۲۴: کلاس تست C++

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

```

1 Test(){}

```

نمونه کد ۱۲۵: کانستراکتور C++

از آنجا که در تعریف این کلاس مستقیماً 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

```
۱ ~Test(){}
۲
```

نمونه کد ۱۲۶: دیستراکتور C++

پیاده سازی دیستراکتور کلاس است که برای انجام این کار کافیسیت از یک علامت مد در ابتدای کانستراکتور 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 b تنها در محدوده شرط قابل دسترسی است

```
۱ if (true)
۲ {
۳     Test b(15);
۴     cout << "statement" If "in" << endl;
۵ }
```

نمونه کد ۱۲۷: C++

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

```
۱ int main()
۲ {
۳     Test *pTest = MethodForHeapAllocDemo();
۴     delete pTest;
۵ }
```

نمونه کد ۱۲۸: C++

در زبان برنامه نویسی ++C بسته به دلخواه برنامه نویس می توان یک کلاس را روی Heap یا Stack تعریف کرد این یکی از تفاوت های این زبان با سی شارپ است. در سی شارپ هر کلاس لزوماً روی Heap تعریف 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 اتومات حافظه را ممکن می سازد. این به آن معناست که بار مدیریت حافظه های اختصاص یافته از دوش برنامه نویس برداشته می شود. برای اطلاعات بیشتر میتوانید به اینجا [۴]. مراجعه کنید Garbage Collector تنها متغیر ها و اشیایی که روی Heap قرار دارند را مورد بررسی قرار میدهد و متغیرهای ذخیره شده روی Stack همانند ++C با پایان یافتن محدوده تعریف از دسترس خارج میشوند.

کلاس زیر در زبان سی شارپ را در نظر بگیرید

```

۱  class Test
۲  {
۳      public int N {get; set;}
۴      public Test(int n)
۵      {
۶          N = n;
۷          Console.WriteLine($"{N}");
۸      }
۹
۱۰     ~Test()
۱۱     {
۱۲         Console.WriteLine($"{N}" finalizer "In");
۱۳     }
۱۴ }
```

نمونه کد ۱۲۹: کلاس تست

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

```

۱  ~Test()
۲  {
۳      Console.WriteLine($"{N}" finalizer "In");
۴  }
```

نمونه کد ۱۳۰: 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

۳.۱۳ Struct

۱.۳.۱۳ struct in cpp

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

```

۱ struct Test
۲ {
۳ };

```

نمونه کد ۱۳۱: تعریف یک استراکت

۲.۳.۱۳ struct in csharp

استراکت ها با کلاس ها در این زبان متفاوت اند.

struct یک Value Type و کلاس ها Reference Type اند

```

۱ struct Test
۲ {
۳     int a;
۴     int b;
۵     int c;
۶ }

```

نمونه کد ۱۳۲: تعریف یک استراکت

این نکته شایان ذکر است که سائز کلی استراکت ها به اندازه سائز 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 Value Type و Reference Type

مسعود گلستانه - ۱۳۹۹/۱/۱۸

جزوه جلسه ۱۴ ام 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 (و نه رشته، به دلیلی که در ادامه گفته خواهد شد) Value type گفته می شوند. این تایپ ها مقداری ثابت دارند و هنگامی که شما متغیری از این نوع ایجاد می کنید، کامپایلر کدی را تولید می کند که یک بلوک از حافظه را، که به اندازه کافی برای نگه داشتن مقدار متناظر با آن بزرگ است، به آن اختصاص می دهد. 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 می کند باعث می شود مقدار آن (مانند ۴۲) در این بلوک حافظه کپی شود.

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

داده نوع های زیر همگی Reference Type هستند :

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

مهم:

توجه داشته باشید رشته در سی شارپ در واقع یک کلاس است. این امر به این دلیل است که هیچ اندازه‌ی استاندارد‌ی برای یک رشته وجود ندارد (رشته های مختلف می توانند شامل تعداد مختلفی از کاراکترها باشند) و تخصیص حافظه برای یک رشته به صورت پویا هنگام اجرای برنامه بسیار کارآمدتر از انجام این کار بصورت استاتیک در زمان کامپایل است [۴].

۱.۱.۱۴ فهم heap و stack

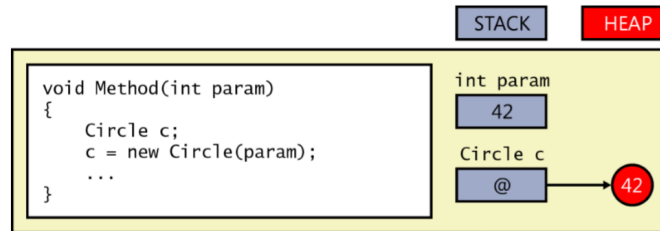
به نمونه کد سی شارپ زیر توجه کنید:

```

۱ public class Circle
۲ {
۳     //some property
۴ }
۵ void Test(int param)
۶ {
۷     Circle c;
۸     c = new Circle(param);
۹     ...
۱۰ }
```

Csharp ۱۳۳: کد نمونه

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



شکل ۱۰۱۴: تخصیص حافظه

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

۱. اگرچه شیء در هیپ ذخیره می شود ، اما رفرنس به شیء در پشته ذخیره می شود.
۲. حافظه ی هیپ بی نهایت نیست. اگر حافظه هیپ پر شود ، اپراتور `new` استثناء `OutOfMemoryException` را پرتاب می کند و شیء ایجاد نمی شود.

مهم:

برای بسیاری از توسعه دهندگان (مانند توسعه دهندگان زبان های مدیریت نشده ++C / C)، اصطلاحات `reference type` و `value type` در ابتدا عجیب به نظر می رسد. در ++C / C ، شما یک تایپ را ایجاد 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 پشته 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. [۹]

۲.۱۴ کپی سطحی (shallow copy)

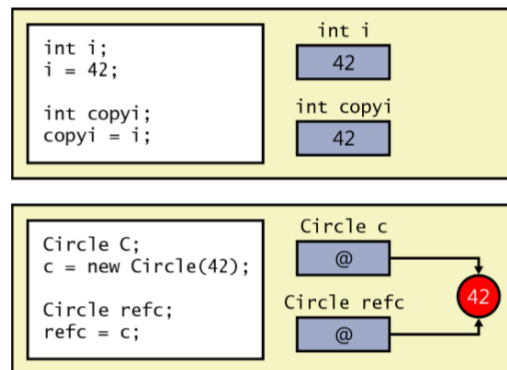
وضعیتی را در نظر بگیرید [نمونه کد ۲.۱۸] که در آن یک متغیر به نام *i* به عنوان *int* تعریف کنید و مقدار آن را ۴۲ اختصاص دهید. اگر متغیر دیگری به نام *Copyi* را به عنوان *int* اعلام کنید و سپس *i* را به *Copyi* اختصاص دهید، *Copyi* همان مقدار *i* را نگه می دارد (۴۲). با وجود اینکه *Copyi* و *i* مقدار یکسانی دارند، دو بلوک حافظه جداگانه این مقادیر را نگه می دارند: یکی بلوک برای *i* و بلوک دیگر برای *Copyi*. اگر مقدار *i* را تغییر دهید، مقدار *Copyi* تغییر نمی کند.

```

۱ int i = 42;           // declare and initialize i
۲ int copyi = i;       /* copyi contains a copy of the data in i and,
۳                       both contain the value 42 */
۴ i++;                 /* incrementing i has no effect on copyi;
۵                       i now contains 43, but copyi still contains 42 */
۶
۷ Circle c = new Circle(42);
۸ Circle refc = c;
```

Csharp ۱۳۴: کد نمونه

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



شکل ۲.۱۴: کپی سطحی

۳.۱۴ باکسینگ و آنباکسینگ

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

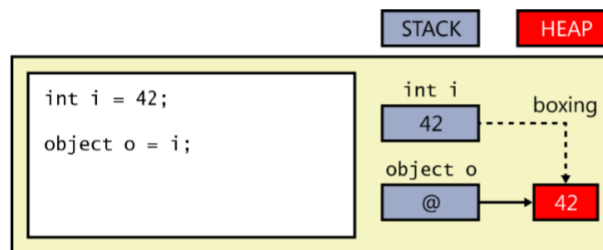
۱.۳.۱۴ باکسینگ

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

```
۱ int i = 42;
۲ object o = i;
```

Csharp ۱۳۵: کد نمونه

همانطور که میدانیم i یک value type است و روی پشته قرار دارد. اگر رفرنس داخل o مستقیماً به i اشاره کند ، رفرنس به پشته ارجاع می شود. اما ، همه رفرنس ها باید به اشیاء موجود در هیپ اشاره کنند. ایجاد رفرنس به موارد موجود در پشته می تواند استحکام زمان اجرا را به خطر بیاندازد و یک نقص امنیتی بالقوه ایجاد کند ، بنابراین این کارمجاز نیست. در نتیجه، ران تایم بخشی از حافظه را از هیپ برای این کار اختصاص می دهد ، مقدار عدد صحیح i را به این قطعه حافظه کپی می کند و سپس شیء o را به این نسخه ارجاع می دهد. به این کپی کردن خودکار یک چیز از پشته تا هیپ ، boxing گفته می شود. نمودار زیر نتیجه را نشان می دهد:



شکل ۳.۱۴: باکسینگ

مهم :

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

۲.۳.۱۴ آنباکسینگ

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

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

تأثیر این کست بسیار ظریف است. کامپایلر متوجه می شود که شما نوع *int* را در کست مشخص کرده اید. در مرحله بعد ، کامپایلر کدی را تولید می کند تا بررسی کند که در واقع در زمان اجرا به چه چیزی اشاره دارد. میتونه کاملاً هر چیزی باشه فقط به این دلیل که کست شما می گوید *o* به یک *int* اشاره دارد ، این بدان معنا نیست که در واقع این کار را می کند. اگر واقعاً به یک *int* باکس شده اشاره کرده و همه چیز مطابقت داشته باشد ، کست موفق می شوند و کد تولید شده توسط کامپایلر مقدار را از جعبه داخلی خارج می کند و آن را در *i* کپی می کنند و در غیر این صورت اکسپشن *InvalidCastException* رخ می دهد.

۴.۱۴ Nullable ها در سی شارپ

مقدار *null* برای مقداردهی رفرنس تایپ ها به کار می رود . و یک *value type* نمی تواند مقدار *null* را در خود نگه دارد. برای مثال *int null = i* باعث می شود که در زمان اجرا با خطا روبرو 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 :

```
۱ Nullable<int> i = null;
```

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

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

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

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

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

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

```
j = i; //illegal
```

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

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

جلسه ۱۵

ادامه مبحث توابع و پارامترها

یاسمن توکلی - ۱۳۹۹/۱/۲۰

جزوه جلسه ۱۵ ام مورخ ۱۳۹۹/۱/۲۰ درس برنامه‌سازی پیشرفته در زمان مقرر تحویل داده نشد.

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.

```
۱ static void Main(string[] args)
۲ {
۳     Console.WriteLine(Before" - Main "In);
۴     MethodA();
۵     Console.WriteLine(After" - Main "In);
۶ }
```

نمونه کد ۱۳۶: تابع Main که در آن تابع MethodA صدا زده می‌شود.

```

۱ private static void MethodA()
۲ {
۳     Console.WriteLine(Before" - A "In);
۴     MethodB();
۵     Console.WriteLine(After" - A "In);
۶ }

```

نمونه کد ۱۳۷: تابع MethodA که در آن تابع MethodB صدا زده می‌شود.

```

۱ private static void MethodB()
۲ {
۳     Console.WriteLine(Before" - B "In);
۴     MethodC();
۵     Console.WriteLine(After" - B "In);
۶ }

```

نمونه کد ۱۳۸: تابع MethodB که در آن تابع MethodC صدا زده می‌شود.

```

۱ private static void MethodC()
۲ {
۳     Console.WriteLine(Before" - C "In);
۴     MethodD();
۵     Console.WriteLine(After" - C "In);
۶ }

```

نمونه کد ۱۳۹: تابع MethodC که در آن تابع MethodD صدا زده می‌شود.

```

۱ private static void MethodD()
۲ {
۳     Console.WriteLine(Before" - D "In);
۴     Console.WriteLine(After" - D "In);
۵ }

```

نمونه کد ۱۴۰: تابع MethodD بدون اکسپشن (حالت عادی)

```

۱ private static void MethodD()
۲ {
۳     Console.WriteLine(Before" - D "In);
۴     throw new InvalidOperationException();
۵     Console.WriteLine(After" - D "In);
۶ }

```

نمونه کد ۱۴۱: 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 ۴م 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 ۱۴۲: تابع Main که به آن (try - catch) اضافه شده‌است.

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


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

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

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

نمونه کد ۱۴۳: تابع MethodB که به آن (try - catch) اضافه شده‌است.

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

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

همینطور که می‌بینید در اینجا بعد از پرتاب شدن اکسپشن به خط دهم تابع MethodB ۱۰ می‌رویم و بعد دوباره اکسپشن رو پرتاب می‌کنیم و به اینجا ۱۰۱۶ می‌رویم.

```

۱ private static void MethodB()
۲ {
۳     Console.WriteLine("Before" - B "In");
۴     try
۵     {
۶         MethodC();
۷         Console.WriteLine("Calling After - B "In");
۸     }
۹     catch
۱۰    {
۱۱        Console.WriteLine("Catch" - B "In");
۱۲    }
۱۳    finally
۱۴    {
۱۵        Console.WriteLine("Finally" - B "In");
۱۶    }
۱۷    Console.WriteLine("After" - B "In");
۱۸ }

```

نمونه کد ۱۴۴: 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 اکسپشن پرتاب نشده است. (حالت عادی)

خب در اینجا بلاک جدیدی به اسم finally می بینید، کد موجود در این بلاک به هر روی اجرا می شود، چه استثناء رخ دهد چه ندهد. (البته واضح است که اگر اکسپشن catch نشود، این بلاک اجرا نخواهد شد)

نکته ی 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 بلاک (finally) اضافه شده‌است و در MethodD اکسپشن پرتاب شده‌است

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

```

۱ private static void MethodB()
۲ {
۳     Console.WriteLine(Before" - B "In);
۴     try
۵     {
۶         MethodC();
۷         Console.WriteLine(C" Calling After - B "In);
۸     }
۹     catch (OverflowException)
۱0    {
۱1        Console.WriteLine(Catch" - B "In);
۱2    }
۱3    finally
۱4    {
۱5        Console.WriteLine(Finally" - B "In);
۱6    }
۱7    Console.WriteLine(After" - B "In);
۱8 }

```

نمونه کد ۱۴۵: تابع MethodB که در آن بلاک (catch) فقط اکسپشن از نوع (OverflowException) catch می‌کند

کمکی! اگر با انواع اکسپشن‌ها آشنا نیستید، می‌توانید با مراجعه به اینجا [۹] در قسمت Common .NET Exceptions با انواع اکسپشن‌ها در سی شارپ آشنا شوید.

```

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

```

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

همینطور که می‌بینید بعد از پرتاب شدن اکسپشن از بلاک try خارج می‌شویم و خط ۷ اجرا نمی‌شود و بعد وارد بلاک finally می‌شویم و بعد ۱۰۱۶ می‌رویم.

```

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

```

نمونه کد ۱۴۶: مثالی از کاربرد بلاک finally

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

۲.۱۶ Indexers

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

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

نمونه کد ۱۴۷: سینتکس Indexer

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

```
1 class PEntry
2 {
3     public PEntry 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 ۱۴۸: مثالی از Indexer

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

```
۱ struct IntBits
۲ {
۳     public bool this[int index]
۴     {
۵         get
۶         {
۷             return (bits & (1 << index)) != 0;
۸         }
۹         set
۱۰        {
۱۱            if (value)
۱۲                bits |= (1 << index);
۱۳            else
۱۴                bits &= ~(1 << index);
۱۵        }
۱۶    }
۱۷ }
```

نمونه کد ۱۴۹: قطعه کد نمونه برگرفته از کتاب [۹]

جلسه ۱۷

Operator Overloading

نیکی مجیدی فرد - ۱۳۹۸/۱/۳۰

جزوه جلسه ۱۷ ام مورخ ۱۳۹۸/۱/۳۰
برنامه سازی پیشرفته

overloading operator

۱.۱۷ Operator Conversion

• implicit

```
public static operator implicit target-type(source-type v) { return value; }
```

• explicit

```
public static operator explicit target-type(source-type v) { return value; }
```

در اینجا target-Type نشان دهنده نوعی است که می خواهیم source-Type را به آن تبدیل کنیم و value مقدار کلاس پس از تبدیل است .
conversion operator برای تبدیل کردن یک نوع داده ای به نوع داده ای دیگر استفاده می شود و یک شی از کلاس شما را به نوع دیگری که می خواهید تبدیل می کند .

• تفاوت implicit و explicit

تفاوت این دو در به ترتیب غیر مستقیم و مستقیم cast کردن است .
در مثال پایین ما کلاسی داریم که دو property ساعت و دقیقه از نوع int , یک constructor و یک copy constructor دارد که در آن 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 می‌خواهیم 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 داده‌هایمان 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 :

```

۱      public static Time operator+(Time lhs, Time rhs)
۲      {
۳          Time t = new Time(lhs);
۴          t.AddTo(rhs);
۵          return t;
۶      }
۷
۸      public static implicit operator Time(int fromHour)
۹      {
۱0         return new Time(fromHour, 0);
۱۱     }

```

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 :

```

۱      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 .

```

۱      public static Time Parse(string time)
۲      {
۳          int colPos = time.IndexOf(':');
۴          string hour = time.Substring(0, colPos);
۵          string minute = time.Substring(colPos+1);
۶          return new Time(int.Parse(hour), int.Parse(minute));
۷      }
۸
۹      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 "12:30" به Time تبدیل شده سپس این داده که حالا از نوع Time است با t7 جمع می شود سپس ۴ که از نوع int است به Time تبدیل شده و با نوع داده ای Time دیگر جمع می شود.

۲.۱.۱۷ explicit

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 }
```

۲.۱۷ Pairs Operator

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

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

۱.۲.۱۷ ==، !=

به کد زیر توجه کنید :

```

۱ public override bool Equals(object obj)
۲ {
۳     Time t = obj as Time;
۴     if(t != null)
۵         return t.h == this.h && t.m == this.m;
۶     return false;
۷ }
۸
۹ public static bool operator == (Time lhs, Time rhs)
۱۰ {
۱۱     return !lhs.Equals(rhs);
۱۲ }
۱۳
۱۴ public static bool operator !=(Time lhs, Time rhs)
۱۵ {
۱۶     return !lhs.Equals(rhs);
۱۷ }

```

operator == != پیاده‌سازی کرده ایم ولی کد بالا StackOverflowError می‌دهد . نحوه اجرا شدن کد بالا به این صورت است که ابتدا Equals در خط ۱۱ صدا می‌شود . در تابع Equals 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 :

```

۱ 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 می توانیم به صورت زیر بنویسیم :

```

۱ public static bool operator ==(Time lhs, Time rhs)
۲ {
۳     if (! ReferenceEquals(lhs, null))
۴         return lhs.Equals(rhs);
۵     return false;
۶ }
۷
۸ public static bool operator !=(Time lhs, Time rhs) => ! (lhs == rhs);

```

۲.۲.۱۷ <, > and <=, >=

pair operator < بعدی < است که برای مثال می توانید به کد زیر توجه کنید :

```

۱ public static bool operator <(Time lhs, Time rhs)
۲ {
۳     if (lhs.h == rhs.h)
۴         return lhs.m < rhs.m;
۵
۶     return lhs.h < rhs.h;
۷ }

```

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

همانطور که گفتیم باید pair operator ها را جفت پیاده سازی کنیم ، پس اینجا > نیز پیاده سازی کنیم که می توانیم خروجی آن را به صورت `!(lhs < rhs)` بنویسیم که چون حالت `lhs == rhs` را هم در بر می گیرد این حالت را حذف کرده و نهایتاً آن را به صورت زیر می توانیم بنویسیم :

```

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

```

۳.۲.۱۷ operators true/false

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

```

۱ if (t9)
۲     t9 = t9 + 1;

```

حال اینجا به عنوان مثال

```

۱ public static bool operator true(Time t)
۲ {
۳     return t.h != 0 || t.m != 0;
۴ }
۵ public static bool operator false(Time t)
۶ {
۷     if (t)
۸         return false;
۹     return true;
۱۰ }

```

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 .

[؟] [؟]

جلسه ۱۸

واسط ها

باوان دیوانی آذر - ۱۳۹۹/۲/۱

۱.۱۸ چالش ۱

فرض کنید ما کلاس `Student` ، مانند کد زیر را داریم :

```
۱ public class Student
۲ {
۳     public string Name {get; private set;}
۴     public int Id {get; private set;}
۵     public double GPA {get; private set;}
۶
۷     public Student(string name, int id, double gpa)
۸     {
۹         this.Name = name;
۱۰        this.Id = id;
۱۱        this.GPA = gpa;
۱۲    }
۱۳ }
```

نمونه کد ۱۵۰: تعریف کلاس `Student`

همچنین در تابع اصلی Main یک آرایه از جنس Student داریم ، میخواهیم این آرایه را بر حسب شماره دانشجویی Id و معدل GPA مرتب کنیم.

چالش: این کار را چگونه با نوشتن فقط یک تابع انجام بدهیم؟

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

```

۱ private static void IdSort(Student[] students)
۲ {
۳     for (int i = 0; i < students.Length; i++)
۴         for (int j = i + 1; j < students.Length; j++)
۵             if (students[i].Id < students[j].Id)
۶                 Swap(students, i, j);
۷ }

```

نمونه کد ۱۵۱: تعریف تابع IdSort

```

۱ private static void GPASort(Student[] students)
۲ {
۳     for (int i = 0; i < students.Length; i++)
۴         for (int j = i + 1; j < students.Length; j++)
۵             if (students[i].GPA < students[j].GPA)
۶                 Swap(students, i, j);
۷ }

```

نمونه کد ۱۵۲: 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 واسط به نام `IStudentComparer` تعریف میکنیم ، برای تعریف واسط از کلید واژه `Interface` استفاده میکنیم .

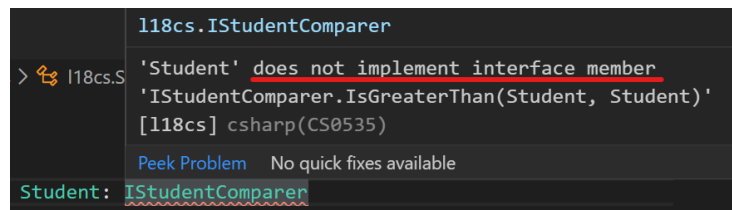
```

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

```

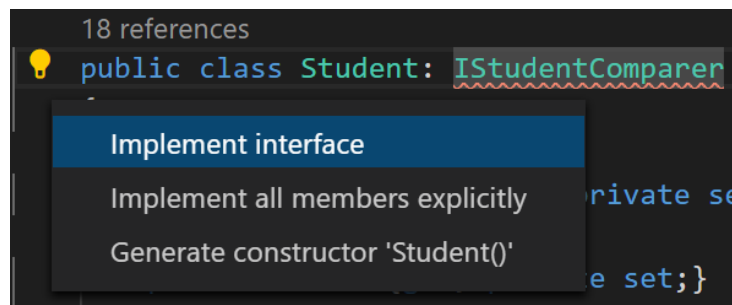
نمونه کد ۱۵۳: تعریف واسط `IStudentComparer`

سپس در کلاس `Student` از واسطمان ارث بری میکنیم . خب همانطور که می بینید با انجام این کار با یک خطا روبرو می شویم .



شکل ۱۰۱۸: خطای شماره یک

برای رفع این خطا از لامپ زردی که در سمت چپ خطی که خطا را داریم قرار گرفته ، کمک می گیریم و با زدن گزینه `Implement interface` خطا رفع می شود .



شکل ۲۰۱۸: رفع خطای شماره یک

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

```

۱ public bool IsGreaterThanOr(Student other)
۲ {
۳     throw new System.NotImplementedException();
۴ }

```

نمونه کد ۱۵۴: تعریف تابع IsGreaterThanOr

اگر دقت کنید ما به یک مشکل برخورد می کنیم . ما برای حل چالشمان میخواستیم فقط یک تابع برای مرتب کردن آرایه بر حسب چند ویژگی بنویسیم ولی در این تابع ما فقط میتوانیم این مقایسه را بر حسب یک ویژگی انجام دهیم . پس برای حل این مشکل دو کلاس تعریف می کنیم که از واسط IStudentComparer ارث بری می کنند .

```

۱ class StudentIdComparer: IStudentComparer
۲ {
۳     public bool IsGreaterThanOr(Student s1, Student s2)
۴         => s1.Id < s2.Id;
۵ }

```

نمونه کد ۱۵۵: تعریف کلاس StudentIdComparer

```

۱ class StudentGPAComparer: IStudentComparer
۲ {
۳     public bool IsGreaterThanOr(Student s1, Student s2)
۴         => s1.GPA < s2.GPA;
۵ }

```

نمونه کد ۱۵۶: 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 ISGreaterThanOr واسطمان استفاده می کنیم .

```

۱ private static void Sort(Student[] students, IStudentComparer cmp)
۲ {
۳     for(int i=0; i<students.Length; i++)
۴         for(int j=i+1; j<students.Length; j++)
۵             if (cmp.IsGreaterThanOr(students[i], students[j]))
۶                 Swap(students, i, j);
۷ }

```

نمونه کد ۱۵۷: 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 تابعمان به عنوان ورودی می دهیم .

```
۱ Sort(students, new StudentGPAComparer());
۲ Sort(students, new StudentIdComparer());
```

نمونه کد ۱۵۸: 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 .

```
۱ static class StudentComparer
۲ {
۳     public static StudentIdComparer StudentIdComparer = new StudentIdComparer();
۴     public static StudentGPAComparer StudentGPAComparer = new StudentGPAComparer();
۵ }
```

نمونه کد ۱۵۹: تعریف کلاس StudentComparer

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

```
۱ Sort(students, StudentComparer.StudentGPAComparer);
۲ PrintStudents(students);
۳
۴ Sort(students, StudentComparer.StudentIdComparer);
۵ PrintStudents(students);
```

نمونه کد ۱۶۰: تست تابع Sort

```
۱ private static void PrintStudents(Student[] students)
۲ {
۳     foreach(var s in students)
۴         Console.WriteLine($"{s.GPA} {s.Id} {s.Name} ");
۵ }
```

نمونه کد ۱۶۱: تعریف تابع PrintStudents

۲.۱۸ چالش ۲

خب ، اگر تست کنید متوجه می شوید که تابعمان به درستی کار می کند ، حالا فرض کنید که کلاس `Teacher` داریم.

```

۱ public class Teacher
۲ {
۳     string Name;
۴     int Id;
۵     double Rating;
۶
۷     public Teacher(string name, int id, double rating)
۸     {
۹         this.Name = name;
۱۰        this.Id = id;
۱۱        this.Rating = rating;
۱۲    }
۱۳ }

```

نمونه کد ۱۶۲: 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 .

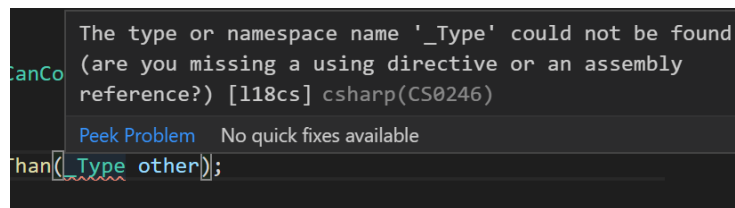
```

۱ public interface ICanCompare
۲ {
۳     int IsGreaterThan(_Type other);
۴ }

```

نمونه کد ۱۶۳: تعریف واسط `ICanComparer`

همانطور که می بینید به یک خطا برخورد می کنیم .



شکل ۳.۱۸: خطای شماره دو

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

```

۱ public interface ICanCompare<_Type>
۲ {
۳     int IsGreaterThan(_Type other);
۴ }

```

نمونه کد ۱۶۴: تعریف واسط `ICanComparer`

خوشبختانه خطا رفع شد. الان باید کلاس های `Teacher` و `Student` این واسط را ارث بری کنند .

```

۱ public class Teacher : ICanCompare<Teacher>
۲ {
۳     string Name;
۴     int Id;
۵     double Rating;
۶
۷     public Teacher(string name, int id, double rating)
۸     {
۹         this.Name = name;
۱۰        this.Id = id;
۱۱        this.Rating = rating;
۱۲    }
۱۳     public int IsGreaterThan(Teacher other) => this.Name.CompareTo(other.Name);
۱۴ }

```

نمونه کد ۱۶۵: تعریف کلاس `Teacher`

```

۱ public class Student : ICanCompare<Student>
۲ {
۳     string Name;
۴     int Id;
۵     double GPA;
۶
۷     public Student(string name, int id, double gpa)
۸     {
۹         this.Name = name;
۱۰        this.Id = id;
۱۱        this.GPA = gpa;
۱۲    }
۱۳    public int IsGreaterThanOr(Student other) => this.Name.CompareTo(other.Name);
۱۴ }

```

نمونه کد ۱۶۶: تعریف کلاس Student

اکنون باید تابع Sort را بنویسیم .

```

۱ private static void Sort<_Type>(_Type[] students)
۲ where _Type: ICanCompare<_Type>
۳ {
۴     for(int i=0; i<students.Length; i++)
۵         for(int j=i+1; j<students.Length; j++)
۶             if (students[i].IsGreaterThanOr(students[j])>0)
۷                 Swap(students, i, j);
۸ }

```

نمونه کد ۱۶۷: تعریف تابع Sort

خب حال نوبت به تست کردن تابعمان می‌رسد .

```

۱ Sort(teachers);
۲ Sort(students);

```

نمونه کد ۱۶۸: تست تابع Sort

خوشبختانه تابعمان به درستی کار می‌کند. همانطور که می‌بینید این راه کمی طولانی بود ، پس بزودی قرار است که با توجه به نکته زیر یک راه بسیار کوتاه‌تر به شما معرفی کنم .

۳.۱۸ نکات

نکته ۱: متغیرهای آرایه و لیست ، خود یک تابع `Sort` دارند (اسم این تابع برای آرایه `Array.Sort` و برای لیست `Sort` است) و فقط لازم است که کلاسی را که می‌خواهیم شیء‌هایش مرتب شوند ، این واسط `IComparer` را ارث بری کرده باشند .

پس برای حل چالش ۲ ، فقط لازم بود کلاس های `Teacher` و `Student` از واسط `IComparer` ارث بری کنند.

نکته ۲: برای تعریف فیلد باید حتما از پراپرتی استفاده کنید .

۴.۱۸ خلاصه بندی

هر زمان چند کلاس زیرمجموعه چیزی باشند ، می‌توانیم از واسط استفاده کنیم .

مثال ۱ : فرض کنید که می‌خواهید برای هر یک از اشکال یک کلاس بنویسید . به همین خاطر می‌توانید یک واسط به نام `IShape` تعریف کنید و این واسط به لیستی از جنس رأس بگیرد و تابع‌هایی برای بدست آوردن محیط و مساحت داشته باشد .

مثال ۲ : فرض کنید که می‌خواهید فضای بیمارستان را تعریف کنید . در نتیجه شما یک گروه بیمار دارید و انواع مختلف دکتر، به عنوان مثال گروهی جراح زیبایی هستند ، گروهی جراح مغز و غیره . برای انجام این کار می‌توانیم از دو واسط `IPerson` و `IDoctor` استفاده کنیم ، بطوریکه کلاس بیمار از واسط `IPerson` ارث بری کند و کلاس دکتر از `IDoctor` و `IPerson` ارث بری کند .

برای مشاهده مثال‌های بیشتر به مراجع [؟،؟] مراجعه کنید .

تعدادی واسط هم خودشان تعریف شده‌اند و ما می‌توانیم از آنها ارث بری کنیم . ما در این جلسه با واسط `IComparer` آشنا شدیم و در جلسه بعدی قرار است با واسط‌های بیشتری مثل `IEnumerator` آشنا شویم .

۵.۱۸ تمرینات اضافی

برای تسط بیشتر می‌توانید تمرین‌های زیر را انجام دهید .

تمرین ۱: فرض کنید که کلاسی به نام `Point` دارید که سه ویژگی `X` و `Y` و `Manitude` را دارد ، همچنین یک آرایه از جنس این کلاس هم داریم . یک تابع `Sort` بنویسید که این آرایه را بر حسب سه ویژگی گفته شده مرتب کند .

تمرین ۲: سوالات و توضیحات در دو لینک [؟،؟] وجود دارند ، پس شما سعی کنید تست کیس‌ها را پاس کنید .

جلسه ۱۹

Interface IEnumerable، IDisposable

آزاده دارایی مقدم - ۱۳۹۹/۲/۶

جزوه جلسه ۱۹ ام 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 جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهشمند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید. مطالبی که در ادامه آمده فقط جنبه راهنمایی شیوه استفاده از لاتک می‌باشد. خواهشمند است این پاراگراف و مطالب بعدی را از نسخه جزوه‌ای که تحویل می‌دهید، حذف کنید.

۱.۱۹ Generic Interface

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

مجموعه کلاس های عمومی یک اینترفیس تعریف کنیم.

```

۱ internal interface IShape
۲ {
۳     void Draw();
۴     double GetArea();
۵ }

```

نمونه کد ۱۶۹: اینترفیس عمومی در سی شارپ

```

۱ internal class Circle: IShape
۲ {
۳     private Point point;
۴     private int v;
۵     public double GetArea() => this.v * this.v * Math.PI;
۶     public void Draw()
۷     {
۸         Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
۹     }
۱۰    public Circle(Point point, int v)
۱۱    {
۱۲        this.point = point;
۱۳        this.v = v;
۱۴    }
۱۵ }

```

نمونه کد ۱۷۰: کلاس circle

```

۱ internal class Triangle: IShape
۲ {
۳     private Point point1;
۴     private Point point2;
۵     private Point point3;
۶     public double GetArea()
۷     {
۸         return (this.point1.X*(this.point2.Y - this.point3.Y)
۹             + this.point2.X*(this.point3.Y - this.point1.Y)
۱۰            + this.point3.X*(this.point1.Y - this.point2.Y))/2;
۱۱     }
۱۲     public void Draw()
۱۳     {
۱۴         Console.WriteLine(" Drawing Triangle at {point1} ,{point2} ,{point3} ");
۱۵     }
۱۶     public Triangle(Point point1, Point point2, Point point3)
۱۷     {
۱۸         this.point1 = point1;
۱۹         this.point2 = point2;
۲۰         this.point3 = point3;
۲۱     }
۲۲ }

```

نمونه کد ۱۷۱: کلاس Triangle

```

۱ using System;
۲ using System.Collections.Generic;
۳ using System.IO;
۴ class Program
۵ {
۶     static void Main(string[] args)
۷     {
۸         Triangle t = new Triangle(new Point(2,4),
۹             new Point(3, -6),
۱0            new Point(7, 8));
۱۱         Circle c = new Circle(new Point(5, 4), 4);
۱۲         DrawShapesWithStats(t);
۱۳         DrawShapesWithStats(c);
۱۴     }
۱۵     static void DrawShapesWithStats(IShape shape)
۱۶     {
۱۷         shape.Draw();
۱۸         Console.WriteLine(shape.GetArea());
۱۹     }
۲۰ }

```

نمونه کد ۱۷۲: تابع main

```

Output:Drawing Triangle at (2,4),(3,-6),(7,8)
27
Drawing Circle at (5,4), with radius: 4
50.26548245743669

```

۲.۱۹ Generic Constraints

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

محدودیت new() مشخص می کند که یک آرگمان در تعریف کلاس عمومی باید دارای یک سازنده بدون پارامتر عمومی باشد. هنگامی که یک کلاس عمومی نمونه های جدیدی از Type را ایجاد می کند، همانطور که در مثال زیر نشان داده شده است، محدودیت جدید را روی پارامتر Type اعمال می کنیم.

به عنوان مثال در نمونه کد بالا برای استفاده از محدودیت ها و new() در کلاس Program، میتوان تابع DrawShapeWithStats را این گونه نوشت:

```

۱ static void DrawShapesWithStats<T>(T shape) where T: IShape, new()
۲ {
۳     T s1 = new T();
۴     shape.Draw();
۵     Console.WriteLine(shape.GetArea());
۶ }

```

نمونه کد ۱۷۳: Generic Constraints

۳.۱۹ IDisposable

زمانی که شیء ای روی یک کلاس ایجاد می کنیم، برای متغیر تعریف شده فضایی در Stack ایجاد می شود، شیء در حافظه Heap که یک حافظه مدیریت شده است ایجاد می شود و آدرس حافظه Heap در Stack قرار می گیرد. حافظه Heap حافظه ای است که دائماً توسط سرویسی به نام Collector Garbage مدیریت می شود و اشیاء بدون استفاده آن، توسط این سرویس حذف می شوند.

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

به عنوان مثال در کلاس Circle داریم:

```

۱ internal class Circle: IShape, IDisposable
۲ {
۳     private Point point;
۴     private int v;
۵     public void Dispose()
۶     {
۷         Console.WriteLine(" Clearing Circle ");
۸     }
۹     public double GetArea()
۱0    {
۱1        return this.v * this.v * Math.PI;
۱2    }
۱3    public void Draw()
۱4    {
۱5        Console.WriteLine(" Drawing Circle at {point} ,with radius: {v} ");
۱6    }
۱7    public Circle(){ }
۱8    public Circle(Point point, int v)
۱9    {
۲0        this.point = point;
۲1        this.v = v;
۲2    }
۲3 }

```

نمونه کد ۱۷۴: تابع main

StreamReader Class ۴.۱۹

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

```

۱ static void Main()
۲ {
۳     string line;
۴     using (StreamReader reader = new StreamReader("file.txt"))
۵     {
۶         line = reader.ReadLine();
۷     }
۸     Console.WriteLine(line);
۹ }

```

نمونه کد ۱۷۵: 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.

```

۱ using System;
۲ using System.Diagnostics;
۳ internal class Timer: IDisposable
۴ {
۵     private Stopwatch s = new Stopwatch();
۶     private string Name;
۷     public Timer(string name)
۸     {
۹         this.Name = name;
۱۰        s.Start();
۱۱    }
۱۲    public void Dispose()
۱۳    {
۱۴        s.Stop();
۱۵        Console.WriteLine(" Elapsed Time({this.Name}): {s.Elapsed.ToString()} ");
۱۶    }
۱۷ }

```

نمونه کد ۱۷۶: 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 ۱۷۷: اندازه گیری زمان اجرای برنامه خواندن فایل

Elapsed Time(ReadAllLines): 00:00:00.1567477

Elapsed Time(StreamReader): 00:00:00.0001820

۶.۱۹ IEnumerable

Enumerator شی ای است که قابلیت بازگرداندن هر مورد از یک لیست و گروه داده ای را به صورت یک به یک و ترتیبی که از آن درخواست میشود را دارد. Enumerator به طبقه بندی موارد آگاه است و پیگیری می کند که کجای رشته قرار دارد. و بعد از آن مقدار مورد جاری را بنا به درخواست باز می گرداند. اینترفیس IEnumerable توسط یک کلاس IEnumerable پیاده سازی می شود. IEnumerable یک نوع است که یک متد به نام GetEnumerator دارد که این متد یک enumerator برمی گرداند. Current مقدار جایگاه جاری در رشته را برمی گرداند. MoveNext متدی است که enumerator را به موقعیت بعدی مکانی پیش می برد. و یک مقدار بولین برمی گرداند که نشان دهنده معتبر بودن موقعیت بعدی و یا انتهای رشته می باشد. موقعیت ابتدایی enumerator، قبل از اولین مورد در رشته است. پس تابع MoveNext باید قبل از اولین دسترسی به Current فراخوانی شود. Yield یک عنصر مجموعه را برمی گرداند و موقعیت مکان نما را به عنصر بعدی هدایت می کند.

در نمونه کد زیر کاربردی از این اینترفیس را میبینم.

```

۱ using System.Collections.Generic;
۲ public class PowersOf2
۳ {
۴     static void Main()
۵     {
۶         foreach (int i in Power(2, 8))
۷         {
۸             Console.Write(" {0} ", i);
۹         }
۱۰    }
۱۱    public static IEnumerable<int> Power(int number, int exponent)
۱۲    {
۱۳        int result = 1;
۱۴
۱۵        for (int i = 0; i < exponent; i++)
۱۶        {
۱۷            result = result * number;
۱۸            yield return result;
۱۹        }
۲۰    }
۲۱ }

```

نمونه کد ۱۷۸: کاربرد yield در اینترفیس IEnumerable

Output: 2 4 8 16 32 64 128 256

```

۱ using System;
۲ using System.Collections.Generic;
۳ class Program
۴ {
۵     static void Main()
۶     {
۷         List<int> list = new List<int>();
۸         list.Add(1);
۹         list.Add(5);
۱۰        list.Add(9);
۱۱        List<int>.Enumerator e = list.GetEnumerator();
۱۲        Write(e);
۱۳    }
۱۴    static void Write(IEnumerable<int> e)
۱۵    {
۱۶        while (e.MoveNext())
۱۷        {
۱۸            int value = e.Current;
۱۹            Console.WriteLine(value);
۲۰        }
۲۱    }
۲۲ }

```

نمونه کد ۱۷۹: کاربرد GetEnumerator در اینترفیس IEnumerable

Output: 1 5 9

[؟] [؟] [؟]

جلسه ۲۰

چگونگی کارکرد مموری

سعید شهباب زاده - ۱۳۹۹/۲/۸

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

۱.۲۰ Memorymanager Example

در این جلسه مثالی در باره چگونگی کارکرد مموری در سیستم عامل زده شد که در مورد آن صحبت می شود.

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

```

۱ public class Memorymanager
۲ {
۳     private int size;
۴     private int[] Memory;
۵     private int FirstFree;
۶     private const Int16 NoMoreValues = Int16.MaxValue;
۷     public MemoryManager(int size)
۸     {
۹         this.size = size;
۱۰        this.Memory = new int[size];
۱۱        this.FirstFree = 0;
۱۲        this.Memory[0] = (int) new IntBlock((Int16)size, NoMoreValues);
۱۳    }
۱۴ }

```

نمونه کد ۱۸۰: کلاس Memorymanager

در این کلاس ما بلاک هایی از مموری خواهیم داشت که در ارایه Memory ذخیره خواهند شد

سایز ارایه به وسیله متغیر size مشخص میشود متغیر FirstFree ادرس اولین بلاک خالی در مموری را در خود خواهد داشت و متغیر NoMoreValue همانطور که از اسمش مشخص است برای آن است که بدانیم آیتم دیگری نخواهیم داشت.

همچنین یک ساختار به نام IntBlock را می سازیم برای راحت تر شدن تبدیل های اینتجر که به صورت زیر است:

```

۱ internal struct IntBlock
۲ {
۳     private Int32 @value;
۴
۵     public Int16 size {
۶         get => SplitNumbers(@value).size;
۷         set => CombineNumbers(value, this.next);
۸     }
۹
۱۰    public override string ToString() => $"{next}({size});
۱۱
۱۲    public Int16 next
۱۳    {
۱۴        get => SplitNumbers(@value).next;
۱۵        set => CombineNumbers(this.size, value);
۱۶    }
۱۷
۱۸    public static explicit operator int(IntBlock b) => b.value;
۱۹
۲۰    public static implicit operator IntBlock(int b) => new IntBlock(b);
۲۱
۲۲    public IntBlock(Int32 num) => this.value = num;
۲۳
۲۴    public IntBlock(Int16 size, Int16 next)
۲۵    {
۲۶        @value = CombineNumbers(size, next);
۲۷    }
۲۸    private static int CombineNumbers(Int16 size, Int16 next)
۲۹    {
۳۰        int result = size;
۳۱        result = result << 16;
۳۲        result = result | (UInt16)next;
۳۳        return result;
۳۴    }
۳۵
۳۶    private static (Int16 size, Int16 next) SplitNumbers(int num)
۳۷    {
۳۸        int size = num;
۳۹        size = size >> 16;
۴۰        int next = ((int)Math.Pow(2, 16) - 1) & num;
۴۱        return ((Int16)size, (Int16)next);
۴۲    }
۴۳ }

```

نمونه کد ۱۸۱: کلاس IntBlock

متد CombineNumbers دو عدد از نوع اینتجر ۱۶ بیتی به عنوان ورودی میگیرد و این دو عدد را باهم ترکیب کرده و یک اینتجر ۳۲ بیتی به عنوان خروجی تحویل میدهد. این کار به خاطر جای دادن دو عدد که سایز و ادرس بلاک خالی بعدی است در یک عدد اینتجر هست که بتوانیم در یک خانه ارایه مموری دو عدد داشته باشیم.

متد SplitNumbers هم دقیقاً برعکس متد CombineNumbers هست و میتواند یک عدد اینتجر ۳۲ بیتی را به دو عدد درون یک تاپل برگرداند.

علاوه بر ساختار IntBlock یک کلاس دیگر به نام MemoryBlock داریم که همانطور که از اسمش مشخص است بلاک های مموری را مشخص میکند و دارای دو عدد که ادرس شروع و پایان آن بلاک هست و دارای یک ارایه که محتوای آن بلاک را مشخص میکند.

در زیر میتوانیم این کلاس را نیز مشاهده کنیم:

```

۱ class MemoryBlock
۲ {
۳     private int[] Data;
۴     public int Start;
۵     public int End;
۶
۷     public override string ToString()
۸     {
۹         return $"{this.Start}->{this.End}";
۱۰    }
۱۱
۱۲    public int Size => End - Start + 1;
۱۳    public MemoryBlock(int[] data, int start, int end) =>
۱۴        (Data, Start, End) = (data, start, end);
۱۵ }

```

نمونه کد ۱۸۲: کلاس MemoryBlock

حالا که کلاس های IntBlock و MemoryBlock را دیدیم میتوانیم به سراغ کلاس Memoryman-ager و متد های آن برویم

این کلاس یک متد به نام AcquireMemory دارد که در واقع با گرفتن یک سایز، یک بلاک از مموری با همان سایز را تحویل میدهد.

میتوانیم این متد را ببینیم:

```

۱ public MemoryBlock AcquireMemory(int size)
۲ {
۳     int loc = FirstAvailable(size);
۴     return new MemoryBlock(data:this.Memory, start:loc, end:loc+size-1);
۵ }

```

نمونه کد ۱۸۳: متد AcquireMemory

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

```

۱ public void DeleteMemory(MemoryBlock mb)
۲ {
۳     var current = this.FirstFree;
۴     while (true)
۵     {
۶         IntBlock bi = this.Memory[current];
۷         if (bi.next == NoMoreValues)
۸         {
۹             this.Memory[current] = (int) new IntBlock(bi.size, (Int16) mb.Start);
۱0            this.Memory[mb.Start] = (int) new IntBlock((Int16) mb.Size, NoMoreValues);
۱۱            break;
۱۲        }
۱۳    }
۱۴ }

```

نمونه کد ۱۸۴: متد DeleteMemory

این کلاس همچنین متد های CombineNumbers و SplitNumbers را داراست که نحوه پیاده سازی آنها را پیش تر دیدم.

ما همچنین به یک متد برای بررسی بلاک های خالی مموری نیاز داریم ، این متد FreeBlocks نام دارد و پیاده سازی آن به صورت زیر است.

```

۱ public IEnumerable<IntBlock> FreeBlocks()
۲ {
۳     int current = this.FirstFree;
۴     while (current != NoMoreValues)
۵     {
۶         IntBlock cb = this.Memory[current];
۷         yield return cb;
۸         current = cb.next;
۹     }
۱۰ }

```

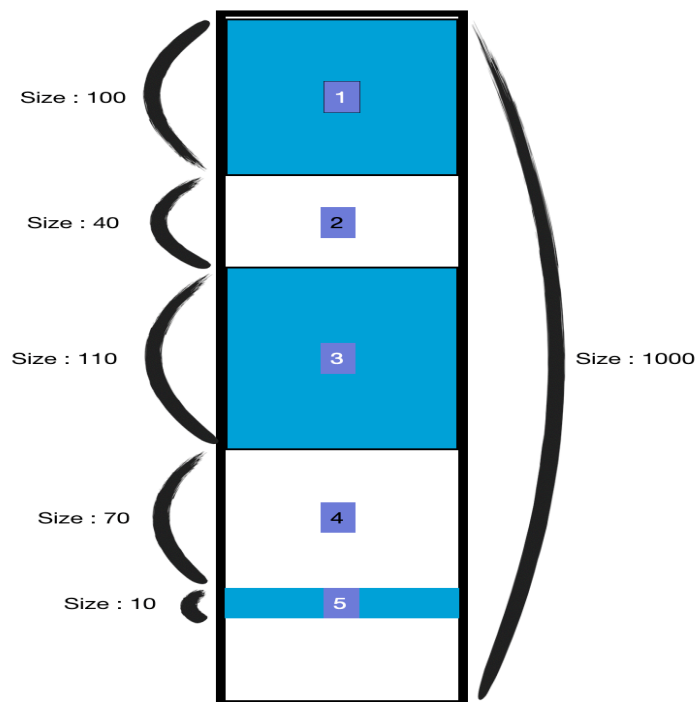
نمونه کد ۱۸۵: متد FreeBlocks

این متد از نوع IEnumerable است به این معنی که میتوانیم بر روی آن foreach بزنیم وعناصر را یکی یکی با دستور yield return برگردانیم.

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

عنصر از آن آرایه را برمیگرداند و از متد خارج میشود و بعد دوباره به متد برمیگردد از ادامه دستور return yield شروع میکند تا دوباره به آن برسد و عنصر بعدی را برمیگرداند

متد دیگری که کلاس MemoryManager داراست متد FirstAvailable است که به عنوان ورودی یک سایز میگیرد و در مموری اولین بلاک خالی که ظرفیت لازم یا همان سایز مورد نظر را دارد پیدا میکند و ادرس آن را برمیگرداند



برای مثال در تصویر بالا اگر یک بلاک با اندازه ۵۰ بخواهیم متد FirstAvailable ادرس بلاک ۴م که عدد ۲۵۰ است را برمیگرداند

حالا می توانیم نحوه پیاده سازی متد را ببینیم :

```

۱ public int FirstAvailable(int size)
۲ {
۳     int current = this.FirstFree;
۴     int last = this.FirstFree;
۵     int location = NoMoreValues;
۶     while(current != NoMoreValues)
۷     {
۸         IntBlock cb = this.Memory[current];
۹         IntBlock lb = this.Memory[last];
۱۰        if (cb.size >= size)
۱۱        {
۱۲            location = current;
۱۳            if (last == current)
۱۴                this.FirstFree = current + size;
۱۵            else
۱۶                this.Memory[last] = (int) new IntBlock( lb.size,
۱۷                    cb.size > size ?
۱۸                    (Int16)(current + size) :
۱۹                    cb.next);
۲۰
۲۱            if (cb.size > size)
۲۲                this.Memory[current + size] = (int) new IntBlock(
۲۳                    (Int16)(cb.size - size), cb.next
۲۴                );
۲۵
۲۶            break;
۲۷        }
۲۸        last = current;
۲۹        current = cb.next;
۳۰    }
۳۱
۳۲    if (current == NoMoreValues)
۳۳        throw new InvalidOperationException();
۳۴
۳۵    return location;
۳۶ }

```

نمونه کد ۱۸۶: متد FirstAvailable

و در آخر یک متد هم برای چاپ کردن بلاک های خالی در کنسول داریم که می توانیم پیاده سازی آن را مشاهده کنیم:

```

۱ public void PrintBlocks()
۲ {
۳     foreach(var info in this.FreeBlocks())
۴         Console.WriteLine(info);
۵ }

```

نمونه کد ۱۸۷: متد PrintBlocks

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

```

۱ static void Main(string[] args)
۲ {
۳     MemoryManager mm = new MemoryManager(size:100);
۴     mm.PrintBlocks();
۵
۶     var m10 = mm.AcquireMemory(10);
۷     mm.PrintBlocks();
۸
۹     var m20 = mm.AcquireMemory(85);
۱۰    mm.PrintBlocks();
۱۱
۱۲    mm.DeleteMemory(m10);
۱۳    mm.PrintBlocks();
۱۴
۱۵    var m3 = mm.AcquireMemory(8);
۱۶    mm.PrintBlocks();
۱۷
۱۸    IntBlock b = new IntBlock(5, 10);
۱۹    int[] memory = new int[10];
۲۰    memory[5] = (int) b;
۲۱ }

```

نمونه کد ۱۸۸: تابع Main

Output:

(۱۰۰.۳۲۷۶۷)

(۹۰.۳۲۷۶۷)

(۵.۳۲۷۶۷)

(۵.۰)

(۱۰.۳۲۷۶۷)

(۵.۸)

(۲.۳۲۷۶۷) (عدد ۳۲۷۶۷ به معنی آن است که بلاکی بعد از آن بلاک وجود ندارد)

جمع بندی :

در این جلسه با نحوه کار مموری در سیستم عامل های مختلف با استفاده از یک مثال آشنا شدیم همچنین نحوه کار با IEnumerable و راحت تر شدن کار به وسیله آن را دیدیم .

با آرزوی موفقیت و سلامتی

جلسه ۲۱

اشاره گر به تابع

مهدیه نادرری - ۱۳۹۸/۲/۱۳

۱.۲۱ اشاره گر به تابع^۱ 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. در نتیجه ما می توانیم مانند مثال زیر یک ورودی را به عنوان تابع روی متغیرهای دیگر صدا بزنیم. سپس با فراخوانی مناسب تابع اصلی نتیجه ی دلخواه را دریافت کنیم.

^۱function pointer

```

۱ def add(a, b):
۲     return a+b
۳
۴ def mul(a, b):
۵     return a*b
۶
۷ def sub(a, b):
۸     return a-b
۹
۱۰ def combine(list_a, list_b, fn):
۱۱     list_result = []
۱۲     for i in range(0, len(list_a)):
۱۳         result = fn(list_a[i], list_b[i])
۱۴         list_result.append(result)
۱۵     return list_result
۱۶
۱۷ def print_pretty(list):
۱۸     for i in range(0, len(list)):
۱۹         print(f"[{i}]={list[i]}")
۲۰
۲۱ def main():
۲۲     a = [1, 2, 3, 4, 5]
۲۳     b = [2, 3, 4, 5, 7]
۲۴     c = combine(a, b, add)
۲۵     print_pretty(c)
۲۶
۲۷ if __name__ == "__main__":
۲۸     main()

```

نمونه کد ۱۸۹: ارسال یک تابع به عنوان ورودی تابع دیگر در پایتون

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

```

۱ def negative(a):
۲     return -1 * a

```

۲.۲۱ اشاره گر 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 واژه `ی`

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

```

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.

```

۱ static void Sort(int[] list)
۲ {
۳     for (int i = 0; i < list.Length; i++)
۴     {
۵         for (int j = i+1; j < list.Length; j++)
۶         {
۷             if(list[i] < list[j])
۸                 (list[i], list[j]) = (list[j], list[i]);
۹         }
۱0    }
۱1 }

```

تاپل انجام میشود. $(list[i], list[j]) = (list[j], list[i])$ یک روش برای Swap() هست که با جابجایی دو قسمت

حالا اگر بخواهیم تابع Combine() را برای نوع جدیدی مانند Student بنویسیم، اول لازم است کلاس آن را در فایل جدید Student.cs بسازیم. حال برای اینکه تابع Sort() برای آن بنویسیم باید:

```

۱ static void Sort(Student[] list, IStudentComparer cmp)
۲ {
۳     for (int i = 0; i < list.Length; i++)
۴     {
۵         for (int j = i+1; j < list.Length; j++)
۶         {
۷             if(cmp.IsGreater(list[i], list[j]))
۸                 (list[i], list[j]) = (list[j], list[i]);
۹         }
۱0    }
۱1 }

```

اینترفیس IStudentComparer پیاده سازی بشود که می خواهیم برای مقایسه ی دو Student تابعی داشته باشد. و از آن در کلاس ها استفاده کنیم. برای پیاده سازی آن میتونیم از کلاس جدیدی در Main استفاده کنیم. و کلاس StudentComparer که اینترفیس مورد نظرا دارد، بسازیم. به طور دلخواه ویژگی مورد مقایسه در IsGreater() را GPA در نظر 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])`

```

۱ namespace l21cs
۲ {
۳     internal interface IStudentComparer
۴     {
۵         bool IsGreater(Student s1, Student s2);
۶     }
۷ }

```

```

۱ namespace l21cs
۲ {
۳     public class StudentComparer : IStudentComparer<Student>
۴     {
۵         public bool IsGreater(Student s1, Student s2)
۶         {
۷             return s1.GPA > s2.GPA;
۸         }
۹     }
۱۰ }

```

اگر بخواهیم Sort() برای هر نوع داده ای، اجرا شود باید تغییراتی در کد های قبل اعمال کنیم. (نمونه کد ۲۰۱۸).

```

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

```

```

۱ internal interface IStudentComparer<Type>
۲ {
۳     bool IsGreater(Type s1, Type s2);
۴ }

```

کاری که تاکنون 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.

```

۱ delegate bool student_comparer(Student s1, Student s2));

```

```

۱ static void sort<_Type>(Student[] list, student_comparer stdcmp)
۲ {
۳     for(int i=0; i<list.Length; i++)
۴         for(int j=i+1; j<list.Length; j++)
۵             if (stdcmp(list[i], list[j]))
۶                 (list[i], list[j]) = (list[j], list[i]);
۷ }

```

دو تابع جدید برای مقایسه ی اعضای مجموعه نوشته ایم، یکی بر اساس GPA و دیگری بر اساس Id.

```

۱ static bool StdCmpGPA(Student s1, Student s2) => s1.GPA > s2.GPA;
۲ static bool StdCmpId(Student s1, Student s2) => s1.id > s2.id;

```

وقتی دو تابع همنام باشند، یکی از آنها overload دیگری است. در یکی از overload های Sort() میتوان از شیئی از کلاس StudentComparer استفاده کرد و در دیگری از تابع IsGreater() آن و البته از هر تابع مشابه دیگر. در مقایسه ی اینکه نوشتن این برنامه با اینترفیس بهتر است یا دلگیت می توان اشاره کرد که اگه اینترفیسی لازم داشتیم که چند متود داشت ، استفاده از دلگیت خیلی جالب نبود. برای دیدن نتیجه مرتب کردن لیست ، لازم است تابعی برای چاپ کردن بنویسیم.

```

۱ private static void print<_Type>(_Type[] stdlist)
۲ {
۳     Console.WriteLine("-----");
۴     foreach(var s in stdlist)
۵         Console.WriteLine(s);
۶ }
۷ });

```

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

```

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

```

توابع 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 هربار همون طور باقی میماند. باید تابعی بنویسیم که لیست را بی نظم کند. میتوانیم از همان تابع `Sort()` استفاده کنیم که تابع ورودی آن اعضا لیست را بصورت رندوم مقایسه می کند. یعنی عددی که برمیگرداند تصادفی باشد. و با ویژگی های `GPA` و `Id` ارتباطی نداشته باشد.

```

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

```

۱ Student[] stdlist = new Student[]{
۲     new Student(98521234, 8.12),
۳     new Student(97532412, 8.14),
۴     new Student(98234324, 8.13),
۵     new Student(97989899, 8.11),
۶ };
۷
۸ sort(stdlist, new StudentComparer());
۹ print(stdlist);
۱۰ sort(stdlist, RndCmp);
۱۱
۱۲ sort(stdlist, new StudentComparer().IsGreater);
۱۳ print(stdlist);
۱۴ sort(stdlist, RndCmp);
۱۵
۱۶ sort(stdlist, StdCmpGPA);
۱۷ print(stdlist);
۱۸ sort(stdlist, RndCmp);
۱۹
۲۰ sort(stdlist, StdCmpId);
۲۱ print(stdlist);

```

همانطور که می بینید توابع صدا زده شده در Main() هم از الگوی خاصی می کنند. یک لیست مشخص را مرتب می کنند، آن را چاپ کرده و بعد دوباره بهم میریزند. البته با یک تغییر کوچک میتوان گفت طرح اصلی آن این است که اول بهم بریزد بعد سورت و سپس چاپ کند. در نتیجه میتوانیم تابعی بنویسیم که این سه کار را پشت سر هم انجام دهد. پس تابع RunSortExpriment() 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 بگیرد. و delegate های زیر

```

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

```

را برای آن تعریف میکنیم.

```

۱ static void RunSortExperiment<_Type>(
۲     string label,
۳     _Type[] list,
۴     sort_delegate<_Type> sortfn,
۵     print_delegate<_Type[]> printfn,
۶     StudentComparer cmp)
۷     {
۸         sortfn(list, RndCmp);
۹         sortfn(list, cmp);
۱0        Console.WriteLine(label);
۱۱        printfn(list);
۱۲    }

```

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:

```

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

```

یا بطور خلاصه تر

```

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

```

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

```

۱ List<Student> std_list = new List<Student>(stdlist);
۲ var s97532412 = std_list.Find( (Student s) => {
۳     return s.id == 97532412;
۴ });

```

در فریم ورک دات نت delegate هایی از پیش تعیین شده وجود دارد که میتوانند هر تعداد پارامتر با انواع مختلف داشته باشند از جمله Func و Action. مثلاً در تابع Sort() به جای استفاده از Stu- dentComparer، می‌توانیم stdcmp <Student, Student, bool> Func 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 تعریفشان کنیم. و برخی از آن هارا حذف کنیم.

```

۱ delegate int binary_op_int(int a, int b);
۲ delegate void sort_delegate<_Type>(_Type[] list, Func<_Type, _Type, bool> stdcmp);
۳ delegate void print_delegate<_Type>(_Type[] stdlist);

```

حال میتوانیم تابع RunSortExprimment() را بازنویسی کنیم.

```

۱ static void RunSortExperiment<_Type>(
۲     string label,
۳     _Type[] list,
۴     sort_delegate<_Type> sortfn,
۵     Action<_Type[]> printfn,
۶     Func<_Type, _Type, bool> cmp)
۷ {
۸     sortfn(list, RndCmp);
۹     sortfn(list, cmp);
۱0    Console.WriteLine(label);
۱1    printfn(list);
۱2 }

```

و در Main() تغییرات لازم را ایجاد نماییم.

```

۱ RunSortExperiment("InterfaceMethod",
۲     stdlist,
۳     sort,
۴     print,
۵     new StudentComparer().IsGreater);
۶ RunSortExperiment("StdCmpGPA", stdlist, sort, print, StdCmpGPA);
۷ RunSortExperiment("StdCmpId", stdlist, sort, print, StdCmpId);

```