به نام خدا

برنامه نویسی شیئ گرا

(C# .NET)

جزوه مهندس نیرومند
در درس «برنامه‌نویسی پیشرفته»

Object-Oriented concepts — مفاهیم شیئ‌گرایی

- **Class**
- **Objects**
- **Properties**
- **Methods**
- **Events**

Class: templates for objects

- **کلاس**ها قالبی برای شیئ‌ها هستند.
- کلاس‌ها پایه و اساس زبان‌های برنامه‌نویسی شیئ‌گرا هستند.

Objects: instance of a class

- یک **شیئ** یک نمونه از کلاس است.

Properties: characteristics of an object

- **صفت**، به ویژگی‌های یک شیئ گفته می‌شود.

Methods: actions performed by the object.

- **متود**، به عملیاتی که توسط یک object انجام می‌شود گفته می‌شود.
- **نکته:** تمامی متودها باید داخل کلاس‌ها تعریف شوند.

Events: message sent by an object to signal an activity

- **رویداد**، به پیغامی گفته می‌شود که توسط یک شیئ برای انجام یک عملیات فرستاده (signal) می‌شود.

C Sharp Programming Language:

- C# توسط مایکروسافت، در سال ۲۰۰۰ ابداع و عرضه شد.
- C# بهبود یافته و تسهیل یافته زبان C++ است.
- C# ترکیبی از C++ و Java است.
- C# یک زبان مدیریت شده است.

Namespace: (فضای نام)

Used to organize the program and prevent name collision

برای دسته‌بندی برنامه و کلاس‌های برنامه و جلوگیری از تصادم یا برخورد نام‌های مشابه از فضای نام استفاده می‌شود.

مثالی برای جلوگیری از برخورد نام‌های مشابه: (دو کلاس به یک نام اما در فضای نام‌های مختلف)

```
class1: yz.customer
class2: mn.xy.customer

xyz.customer  cu = new xyz.customer();
mn.xy.customer  cu = new mn.xy.customer();
```

- هر فضای نام شامل کلاس‌هایی مرتبط با یکدیگر است.
- هر فضای نام ممکن است از چندین فضای نام دیگر تشکیل شده باشد.
- همان مفهوم package در جاوا است، برای دسته بندی کلاس‌ها به کار می‌روند.

- **.NET Framework**:

Provides the tools and features necessary for building and running applications

Net framework. ابزارهای لازم جهت ساخت و اجرای برنامه های کاربردی را فراهم می‌کند.

.NET Framework Components: **(اجزای دات.نت فریم‌ورک)**

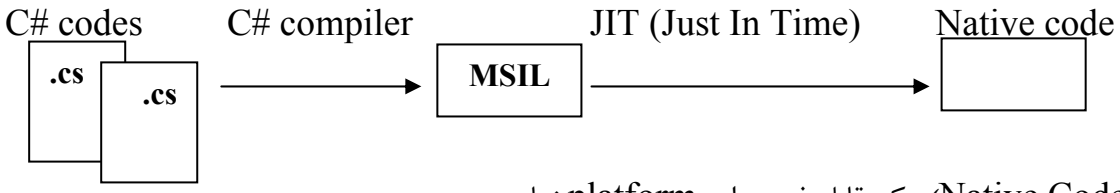
1) **Common Language Runtime (CLR)**

Platform for running .NET apps (similar to the JVM in java)

JVM: java virtual machine

☺ CLR یک محیط امن برای اجرای تمامی برنامه‌های .NET است.

روند اجرای برنامه‌ها:



کد محلی (Native Code) : کد قابل فهم برای platform نهایی

روند اجرای کدهای C# بر روی بسترهای مختلف:

برنامه‌های نوشته شده به زبان C# در لحظه کامپایل به یک زبان میانی به نام MSIL (Microsoft Intermediate Language) یا زبان میانی مایکروسافت تبدیل می‌شوند. خروجی برنامه به platform (بستر) نهایی منتقل می‌شود که بر روی آن Net framework. مربوط به همان platform نصب است. کامپایلرJIT که جزئی از Net framework. است کدهای نهایی را به کدهای قابل فهم برای platform نهایی تبدیل می‌کند.

- Net framework. مربوط به هر platform توسط مایکروسافت یا مؤسسات دیگر ساخته می‌شود.

2) Class Library (CL)

A collection of classes

شامل مجموعه‌ای از کلاس‌ها

به طور مثال:

کلاس‌هایی برای کارهای تحت وب

کلاس‌هایی برای اتصال به بانک اطلاعاتی

کلاس‌هایی برای اجاد فرم‌های ویندوزی

3) **Common Type System (CTS)**
   Types supported by .NET

تعریف CTS:

- شامل انواع داده‌ای پایه‌ای است و برای هماهنگی بین زبان‌های .NET استفاده می‌شود.
- این بخش از Net framework. شامل انواع داده‌ای عمومی است که تمامی انواع داده‌ای استفاده شده در زبان‌های .NET. به آنها تبدیل می‌شوند (یا در اصطلاح، map می‌شوند). به طور مثال کلمه کلیدی integer در زبان VB.NET به نوع داده‌ای int32 تبدیل می‌شود و یا کلمه کلیدی int در C# به همین صورت.

.NET Languages:

1) VB.NET
2) C#.NET
3) C++.NET
4) J#.NET

انواع برنامه‌ها در C#:

- Windows Applications → System.windows.Forms
  System.windows.Drawing

- Web Applications → System.Web
  System.UI

- XML Web Service → System.Web
  System.Services

C# is a "type-safe" language. (Type-safe code can only access memory that it is authorized to access)

C# یک زبان type-safe است یعنی به بخشی از حافظه دسترسی دارد که اجازه دسترسی داشته باشد.

C# is a strongly-typed language. (Every value or variable must be declared as a specific type)

C# یه زبان Strongly-Typed است. یعنی هر مقدار و هر متغیر باید به عنوان یک نوع داده خاص تعریف شده باشد.

ساده ترین برنامه به زبان C#:

```
using System;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

☺نکته:

۱- متود Main آغاز اجرای برنامه است.

۲- هر دستور به یک (;) ختم می‌شود.

۳- {} مشخص کننده یک Block از اطلاعات هستند، هر کد بین این علامات متعلق به Block است.

۴- هر متغیر در هر Block فقط در همان Block قابل استفاده است.

۵- C# یک زبان Case Sensitive است یعنی به حروف بزرگ و کوچک حساس است.

Main ≠ main

انواع Comments یا توضیحات در C#:

۱- مقابل کد

```
Console.WriteLine("Hello World"); // any comment
```

۲- در یک خط جداگانه

```
// any comment
```

۳- یک بلاک توضیحات:

```
/* comments */
```

Commentها نه کامپایل می‌شوند و نه اجرا و هیچ تاثیری در روند اجرای برنامه ندارند، فقط برای یادآوری به خود برنامه‌نویس یا برنامه‌نویس‌های بعدی مفید هستند.

انواع داده‌ای در C#:

Keyword	.NET Framework Type	Value Range	Example
bool	System.Boolean	True/False	`bool result=true;`
char	System.Char	0 to 65536 bit	`char letter='a';`
decimal	System. Decimal	-7.9e28 to 79e28	`decimal val=1.23M;`
double	System. Double	-1.79e308 to 1.79e.308	`double x=1.23M;`
float	System.Single	-3.40e38 to 3.40e38	`float val=1.23F;`
int	System.int32	-2.147.483.648 to 2.147.483.647	`int val=12;`
string	System.String	All valid string characters	`string str="Hi Hamid";`

نکاتی در مورد تعریف داده‌ها:

۱- هنگام تعریف داده‌ی Char از علامت ' ' در اطراف مقدار استفاده می‌کنیم.

۲- هنگام تعریف داده‌ی string از علامت " "در اطراف مقدار استفاده می‌کنیم.

۳- هنگام تعریف اعداد اعشاری اگر نوع داده Decimal انتخاب شد در انتهای عدد کاراکتر M و اگر نوع داده Float انتخاب شد در انتهای عدد کاراکتر F درج می‌شود.

۴- برای تعریف یک نوع داده هم می‌توان از کلمه کلیدی مربوط به زبان C# استفاده کرد و هم از کلمه کلیدی .NET Framework.

انواع تعریف متغیر در C#.NET :

```
int x=23;
System.Int32 x=23;
int x=new int ();
```

مزیت حالت سوم این است که نیاز به مقداردهی اولیه ندارد بلکه با مقدار پیش‌فرض مقداردهی می‌شود.

۵- چندین متغیر می‌توانند در یک خط تعریف شوند.

۶- متغیرهای محلی باید حتماً مقداردهی شوند.

نحوه تعریف متغیر:

ابتدا کلمه کلیدی مربوط به نوع متغیر را درج می‌کنیم سپس نامی‌که برای متغیر انتخاب کردیم سپس یک مساوی و بعد مقدار متغیر را به آن نسبت می‌دهیم.

مثال:

```
Type     name = value;
 int     pica = 10;
```

مثال: برنامه‌ای بنویسید که یک متغیر از نوع string با مقدار it is my first application تعریف کند و در خروجی چاپ کند؟

```
static void Main(string[] args)
{
            string x = "It is my first application";
            Console.WriteLine(x);
            Console.Read();
}
```

برای باز ماندن پنجره خروجی (که نهایتاً با فشار دادن دکمه Enter بسته می‌شود)

دستور دریافت مقدار از ورودی:

در فضای Console در حالی که در ابتدای برنامه، فضای نام سیستم را وارد کرده‌ایم در بدنه برنامه از دستور زیر استفاده می‌کنیم:

```
Console.ReadLine();
```

دستور نمایش اطلاعات در خروجی:

در فضای Console در حالی که در ابتدای برنامه، فضای نام سیستم را وارد کرده‌ایم در بدنه برنامه از دستور زیر استفاده می‌کنیم:

```
Console.WriteLine();
```

```
Console.Write("Hamid Reza");
Console.Write("Niroomand");
```

خروجی:

```
Hamid Reza Niroomand
```

```
Console.WriteLine("Hamid Reza");
Console.WriteLine("Niroomand");
```

خروجی:

```
Hamid Reza
Niroomand
```

دستورات شرطی	Conditional Statements

۱- if

۲- switch

سه ساختار کلی دستور if :

if (شرط ۱)
{
دستورات در صورت
برقراری شرط ۱
}
else if (شرط ۲) {
دستورات در صورت
صحت شرط ۲
}
	if (شرط) {
	دستورات در صورت
	برقراری شرط
	}
	else {
	دستورات در صورت
	عدم برقراری شرط
	}
		if (شرط)
		{
		دستورات در صورت
		برقراری شرط
		}

☺ نکته:

۱- بلافاصله بعد از if پرانتز باز قرار می‌گیرد.

۲- حتما باید در داخل پرانتز if یک مقدار boolean ذکر گردد.

۳- مقایسه تساوی دو عبارت با (==) و انتساب با (=) انجام می‌شود.

۴- اگر بعد از if یک خط کد قرار گیرد نیازی به {} نیست.

۵- اگر چندین شرط در if داشته باشیم برای and کردن از(&&) و برای or از (||) استفاده می‌کنیم.

۶- اگر if، {} نداشته باشد و بعد از آن بیش از یک خط کد باشد فقط خط اول به عنوان بدنه if در نظر گرفته می‌شود و بقیه کدها بدون ارتباط با if اجرا می‌شوند.

مثال: برنامه‌ای بنویسید که یک کاراکتر از ورودی بخواند در صورتی که این کاراکتر y بود در خروجی چاپ کند you mean Yes و اگر n بود در خروجی چاپ کند you mean "No" و در غیر این صورت همان کاراکتر را چاپ کند؟

```
static void Main(string[] args)
{
   string character;
   character = Console.ReadLine();
   if (character == "y")
      Console.WriteLine("you mean Yes");
   else if (character == "n")
      Console.WriteLine("you mean \"No\"");
   else
      Console.WriteLine(character);
   Console.Read();
}
```

ساختار کلی دستور switch:

switch (variable) {
case value1:
Body Statements
break;
 	
case value2:
Body Statements
break;
.
.
.
.
default:
Body Statements
break;
}
	switch (متغیر) {
	case ۱مقدار:
	دستورات در صورت برقراری مقدار۱
	break;
	case ۲مقدار:
	دستورات در صورت برقراری مقدار۲
	break;
	.
	.
	.
	default:
	دستورات در صورت عدم برقراری مقادیر مختلف
	
	break;
	}

مثال: با استفاده از دستور DateTime.Now.DayOfWeek.ToString() میتوان نام روز جاری هفته را به دست آورد.

نام جاری را در یک string ذخیره کنید و با استفاده از دستور switch نام فارسی آن را در خروجی چاپ کنید؟

```
static void Main(string[] args)
      {
          string res = "";
          string day = System.DateTime.Now.DayOfWeek.ToString();
          switch(day)
          {
              case "Saturday":
                  res = "شنبه";
                  break;
              case "Sunday":
                  res = "شنبه یک";
                  break;
              case "Monday":
                  res = "شنبه دو";
                  break;
              case "Tuesday":
                  res = "شنبه سه";
                  break;
              case "Wednessday":
                  res = "شنبه چهار";
                  break;
              case "Thursday":
                  res = "شنبه پنج";
                      break ;
              case "Friday":
                  res = "جمعه";
                  break ;
          }
        Console.WriteLine(res);
```

```
}
```

Iteration Statements	حلقه‌ها در C#

کاربرد: اجرای یک قطعه کد بیش از یک بار تا زمانی که یک شرط برقرار باشد.
مثال: درج روزهای ماه در یک منوی کرکره‌ای
بیشترین استفاده: در کار با آرایه‌ها و خواندن اطلاعات از database

انواع حلقه:

۱- while
۲- do…while
۳- for
۴- foreach

ساختار کلی حلقه while:

```
while (شرط)
{
   Body Statements
}
```

ساختار کلی حلقه do…while:

```
do {
   Body Statements
} while (شرط)
```

☺ نکته: do…while تنها حلقه‌ای است که ضمانت می‌کند که بدنه حلقه حداقل یک بار اجرا شود.

ساختار کلی حلقه for:

```
for (عملیات اولیه ; شرط توقف ; عملیات در هر دور )
{
   Body Statements
}
```

☺ نکته: احتمال فراموش کردن "افزایش شمارنده حلقه" در این حلقه نسبت به سایرین کمتر است.

دستور به دست آوردن روز و ساعت جاری:

```
DateTime.Now.DayOfWeek.ToString();
```

دستور تبدیل عدد به کاراکتر اسکی:

```
Convert.ToChar(int);
```

مثال: برنامه‌ای بنویسید که حروف انگلیسی (بزرگ و کوچک) را به ترتیب را در دو خط جداگانه چاپ کند؟

```
static void Main(string[] args)
{
    int i;
    for (i = 65; i <= 90; i++)
        Console.Write(Convert.ToChar(i)+" ");
    Console.WriteLine();
    for (i = 97; i <= 122; i++)
        Console.Write(Convert.ToChar(i)+" ");
    Console.Read();
}
```

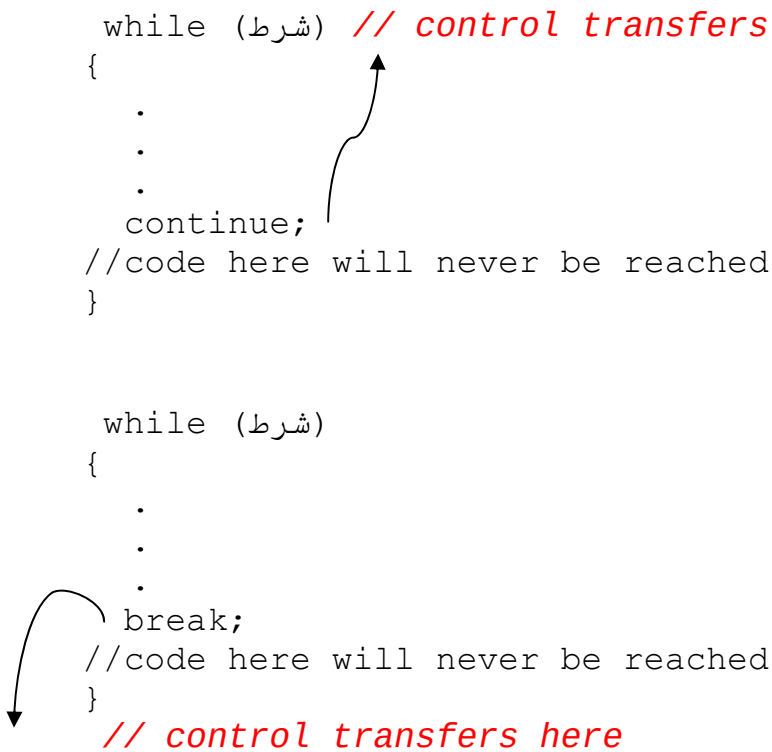
مثال: برنامه‌ای بنویسید که جدول ضرب ۱ تا ۱۰ را در خروجی چاپ کند؟

```
static void Main(string[] args)
{
    int i,j;
    for (i = 1; i <= 10; i++)
    {
        for (j = 1; j <= 10; j++)
            Console.Write(i*j+"\t");
        Console.WriteLine();
    }
    Console.Read();
}
```

مثال: برنامه‌ای بنویسید که عددی را از ورودی گرفته و فاکتوریل آن را حساب کند؟

```
static void Main(string[] args)
{
    int a = Int32.Parse(Console.ReadLine());
    int fact = 1;
    for (int i = a; i >= 1; i--)
        fact = fact * i;
    Console.WriteLine(fact);
    Console.Read();
}
```

دستورات break و continue برای کنترل اجرای روند برنامه:



آرایه‌ها — Arrays

Array: a sequence of data of the same type

آرایه ترکیبی است از داده ها با یک نوع داده مشابه. مثل مجموعه‌ای از نام‌های خانوادگی مشتریان.

☺ نکته:

۱- همه اعضای آرایه از یک نوع هستند.
۲- هر عضو با یک اندیس شناخته و فراخوانی می‌شود.
۳- اندیس‌ها از صفر شروع می‌شوند. (در اصطلاح گفته می‌شود: آرایه‌ها در این زبان Zero-Based هستند)
۴- آرایه در C# یک نوع داده نیست بلکه Reference Type است بنابراین برای تعریف آن از کلمه کلیدی new استفاده می‌شود.

تعریف آرایه:

Type [rank] name = new Type [number of elements];

مثال:

```
int[] rid = new int[10];          یک بعدی

int[,] rid = new int[3, 5];       دو بعدی
```

مقداردهی اولیه به عناصر آرایه:

۱- Implicit (ضمنی)

۲- Explicit (صریح)

مثال:

آرایه‌ای با ۳خانه که مقدار پیش‌فرض تمامی آنها صفر است:

```
int[] x1 = new int[3];                //درست
int[] x2 = new int[] { 8, 2, -3 };    //درست
int[] x3 = new int[3] { 8 };          //نادرست
```

```
int[] xy = new int[3];
xy[0] = 8; //مقداردهی
xy[1] = 2;
xy[2] = -3;
```

مقداردهی اولیه آرایه دو بعدی:

```
int [ , ]  y = new  int [2،3] {{1,2,3},{4,5,6}};
```

1	2	3
4	5	6

نحوه ذخیره سازی عناصر آرایه در حافظه:

1
2
3
4
5
6

پشت سر هم با یک تقسیم منطقی (Logic Division) بین دو بعد

صفات و متودهای آرایه:

یک آرایه در C# یک شیئ به حساب می‌آید و همان طور که گفته شد هر شیئ دارای صفات و متودهایی است. استفاده از صفت Rank برای به دست آوردن تعداد ابعاد آرایه و صفت Length برای به دست آوردن تعداد خانه‌ها (یا طول آرایه).

نحوه استفاده:

```
ArrayName.Rank
ArrayName.Length
```

با استفاده از متود GetUpperBound() و ارسال شماره بُعد به این متود می‌توان باند بالای آرایه در آن بعد را به دست آورد.

با استفاده از متود GetLowerBound() و ارسال شماره بعد به این متود می‌توان باند پایین آرایه در آن بعد را به دست آورد.

مثال:

```
int[] x = { 8, 2, -3 };
Console.WriteLine(x.Rank);          //prints=>1
Console.WriteLine(x.Length);        //prints=>3
```

حلقه foreach:

حلقه‌ای بسیار ساده برای مرور تک تک اعضای آرایه است و تنها برای خواندن وآنالیز عناصر آرایه به کار می‌رود.

ساختار کلی:

foreach (**نوع عنصر** **نام متغیر محلی آرایه** in نام آرایه)

{

عملیات خواندن و آنالیز

}

☺ نکته:

متغیر محلی حلقه در هر دور یک مقدار ثابت دارد و قابل تغییر نیست، بنابراین نمی‌توان از این حلقه برای مقداردهی به عناصر آرایه استفاده کرد.

مرتب سازی حبابی	**Bubble Sorting**

در این روش تک تک عناصر آرایه با عنصر بعدی خود مقایسه می‌شوند اگر عنصر بعدی کوچکتر از عنصر فعلی بود جای هر دو عنصر عوض می‌شود. با این کار عنصر بزرگ‌تر همچون یک حباب به انتهای آرایه منتقل می‌شود. در مرحله بعد عنصر آخر در مقایسه‌ها شرکت نمی‌کند و همان روال قبلی طی می‌شود. در هر مرحله یک عنصر از مقایسه خارج می‌شود تا زمانی که کل عناصر مرتب شوند.

☺ نکته: به دلیل مقایسه‌های بیش از حد، این الگوریتم کمترین بهینگی را دارد.

مثال: تعریف یک آرایه یک بعدی با ۵ عنصر و مرتب سازی حبابی آن:

```
static void Main(string[] args)
{
    int [ ] arr = new int [5] {6, 4,-5, 1, 2};
    int i, j, temp, count = 0;
    foreach (int c in arr)
    {
        Console.WriteLine(c + "\t");
        for (j = 1; j < arr.Length; j++)
        {
            for (i = 0; j < arr.Length - j; i++)
            {
                if (arr[i] > arr[i + 1])
                {
                    temp = arr[i + 1];
                    arr[i + 1] = arr[i];
                    arr[i] = temp;
                }
            }
        }
    }
    Console.ReadLine();
}
```

دو متود مهم کلاس Array در فضای نام System:

1- Sort() method

```
Array.Sort (نام آرایه);
```

2- Reverse() method

```
Array.Reverse (نام آرایه);
```

جستجو در آرایه:

۱- آرایه مرتب ← جستجوی دودویی

در یک آرایه مرتب صورت می‌گیرد. عنصر مورد نظر با عنصر وسط آرایه مقایسه می‌شود، اگر برابر بود، جستجو خاتمه می‌یابد. اگر عنصر مورد جستجو از عنصر وسط بزرگتر بود جستجو در بخش بالایی آرایه و اگر کوچکتر بود در بخش پایینی صورت می‌گیرد.

۲- آرایه نا مرتب ← جستجوی ترتیبی

عنصر مورد جستجو با تک تک عناصر آرایه مقایسه می‌شود. (به دلیل مقایسه‌های بیش از حد، بهینگی ندارد)

روش انتخابی: کوچکترین عنصر آرایه پیدا شده با عنصر اول جا به جا می‌شود، در مرحله بعد بقیه عناصر آرایه برای یافتن کوچک‌ترین عنصر، جستجو می‌شوند و جای آن با عنصر دوم آرایه عوض می‌شود و این روند تا مرتب سازی کامل ادامه می‌یابد.

کار با رشته‌ها:

۱- رشته‌ها متغیر از نوع ارجاع (Reference) هستند. این بدان معناست که متغیر شامل اشاره‌گری به محل ذخیره سازی رشته است نه شامل خود رشته.

۲- برای درج کاراکترهای خاص در بین رشته قبل از آنها یک (\) درج می‌کنیم. علاوه بر آن برای رفع مشکل کوتیشن‌ها می‌توان به اول رشته علامت@ اضافه کرد.

```
string str = "My\"name\" is Nicky";
string str = @"My ""name"" is Nicky";
```

۳- برای تبدیل متغیر غیر رشته‌ای به رشته‌ای می‌توان از متود Convert.ToString() استفاده کرد.

۴- برای تبدیل حروف به بزرگ یا کوچک می‌توان از متود ToUpper() و ToLower() استفاده کرد.

۵- برای جایگزینی یک رشته از متودReplace() و برای حذف ازRemove() استفاده می‌شود.

۶- برای درج رشته از متود Insert() استفاده می‌شود.

۷- برای جدا کردن یک رشته از بین یک رشته دیگر، از متود SubString() استفاده می‌شود.

مثال: برنامه ای بنویسید که یک رشته طولانی از ورودی دریافت کند و تعداد کلمه is در این رشته را در خروجی نمایش دهد؟

```
class Program
{
    static void Main(string[] args)
    {
        string str = Console.ReadLine();
        string search = "is";
        int count = 0;
        for (int i = 0; i <= str.Length - search.Length; i++)
        {
            if (str.Substring(i, search.Length) == search)
                count++;
        }
        Console.WriteLine(count);
    }

}
```

خواندن یک فایل به عنوان ورودی:

۱- فراخوانی فضای نام IO در فضای سیستم در بالای برنامه `using System.IO;`

۲- تعریف متغیزی به نام path به عنوان مسیر فایل متنی

۳- استفاده از کد زیر برای خواندن خط به خط فایل متنی

```
string path = @"C:\test.txt";
using (StreamReader sr = File.OpenText(path))
{
   string s = "";
   while ((s = sr.ReadLine()) != null)
   {
      //Body Statements
   }
}
```

آرایه‌هایی با طول نا مشخص:

گاهی اوقات تعداد عناصر برای ما مشخص نیست به طور مثال وقتی بخواهیم تعداد نامشخصی عدد از ورودی دریافت کنیم در این موارد باید از ArrayList استفاده کرد که در فضای نام System.Collections قراردارد.

نحوه استفاده:

1- using System.Collections;

2- ArrayList Arr = new ArrayList(); //تعریف آرایه

3- Arr.Add (new element); // افزودن عنصر جدید به آرایه

4- Arr [index number] // دسترسی به یک عنصر خاص

☺ نکته: نوع آرایه از روی نوع عناصر تشخیص داده می‌شود.

تشریح مباحث شیئ‌گرایی در#C

Namespace:

A logical construct used to organize programs & libraries in a hierarchical manner.

***فضای نام**، یک ساختار منطقی برای سازماندهی برنامه‌ها و کتابخانه‌ها به یک روش سلسله‌مراتبی است.*

ساختار کلی تعریف و ایجاد یک فضای نام:

```
namespace      NamespaceName {
      //Namespace members → namespace classes
 }
```

Nest namespace: (فضای نام داخلی)

۲روش برای تعریف یک فضای نام به عنوان زیر مجموعه فضای نام دیگر وجود دارد.

۱- تعریف یکی در داخل دیگری

```
namespace  myNamespace {
        namespace myInnerNamespace {
            // Namespace members → namespace classes
    }
}
```

۲- تعریف یک فضای نام با جدا کردن آن توسط یک نقطه از فضای نام خارجی

```
namespace   myNamespace.myInnerNamespace {

}
```

Class:

A template for an object. *یک **کلاس**، قالبی برای یک شیئ است*

A class is a UTD (User-Defined Type)

کلاس یک **نوع داده تعریف شده توسط کاربر** است.
در حقیقت وقتی یک کلاس تعریف می‌کنیم یک **نوع داده سفارشی** تعریف کرده‌ایم.

ساختار کلی تعریف یک کلاس:

```
[Modifier] class  className {
    // Class member → methods, properties, events
}
```

☺ **نکاتی در مورد کلاسها و فضای نامها**:

۱- یک فضای نام می‌تواند در چندین فایل پخش شود.

۲- برای ایجاد یک ساختار چند لایه‌ای می‌توان فضای نام‌ها را در زیر مجموعه یکدیگر تعریف کرد.

۳- کلاس‌ها حتما در بین یک فضای نام تعریف می‌شوند.

۴- مهم‌ترین کاربرد فضای نام جلوگیری از برخورد نام‌های تکراری (Name Collision) است.

۵- در حالت کلی برای استفاده از یک کلاس باید قبل از نام آن، نام فضای نام مربوط به آن را درج کنیم.

به طور مثال برای ارسال یک string به خروجی باید از چنین کدی استفاده کنیم:

```
System.Console.WriteLine (string);
```

اما می‌توان با استفاده ازکلمه کلیدی using از تکرار فضای نام در ابتدای نام کلاس کرد.

```
using System;
Console.WriteLine (string);
```

۶- در عبارتی مثل `System.IO.File.OpenText (path)` عبارت قبل از پرانتز همیشه یک متود است، قبل از متود همیشه نام کلاس است و عبارت‌های قبل از نام کلاس، نام فضای نام‌های مشتمل است.

Object:

Object is an instance of a class. Object is a reference type.

*یک **شیئ**، به یک نمونه از یک کلاس گفته می‌شود.*

ایجاد یک شیئ از کلاس:

نام کلاس نام شیئ = new نام سازنده کلاس ();

مثال:

Customer cu1 = new Customer();

☺ نکته: اگر چندین کلاس با یک نام در فضای نام‌های مختلف وجود داشت برای ایجاد یک شیئ ازکلاس‌ها باید ابتدا نام فضای نام مشخص شود.

مثال:

```
namespace Shop
{
    public class Customer
    {

    }
}
namespace Sales
{
    public class Customer
    {

    }
}
```

ایجاد شیئ از دو کلاس Customer که در بالا تعریف شده اند:

```
Shop.Customer cu1 = new Shop.Customer();
Sales.Customer cu2 = new Sales.Customer();
```

Method:

A runnable entity that contains the executable codes for a C# program.

یک موجودیت قابل اجرا که شامل کدهای قابل اجرا یک برنامه C# است.

- همه کدهای قابل اجرا در بدنه یک متود قرار می‌گیرد و متودها فقط در داخل یک کلاس قابل تعریفند.

ساختارکلی تعریف یک متود:

[Access modifier] returnType methodName ([*parameters*]) {

// The executable codes

}

[سطح دسترسی] نوع برگشتی نام متود ([لیست پارامترها]) {

}

مثال:

```
public int AddOne(int x)
{
    return ++x;
}
```

شرح اجزای متود:

۱- تعیین کننده سطح دسترسی: اختیاری است اگر تعیین نشود نوع پیش‌فرض private در نظر گرفته می‌شود.

۲- نوع مقدار برگشتی: متودها هر نوع داده‌ای را می‌توانند برگردانند، حتی یک نوع داده تعریف شده توسط کاربر. متودی که مقداری را بر نمی‌گرداند void در نظر گرفته می‌شود.

۳- نام متود: اگر چند جزئی باشد بهتر است و ابتدای هر کلمه با کاراکتر بزرگ درج شود.

۴- پارامترها: نوع و مقدار هر پارامتر باید تعریف شود. اگر بیش از یکی باشد با کاما از هم جدا می‌شوند.

۵- بدنه متود: برای برگرداندن مقدار در یک متود از کلمه کلیدی return استفاده می‌شود. هر متغیری که در داخل بدنه متود تعریف شود برای آن متود متغیر محلی محسوب می‌شود و از خارج از متود قابل دسترسی نیست. فقط متغیری که return می‌شود از بیرون قابل مشاهده است.

Access Modifier:

Identifies the visibility of class member such as method, property or a data.

سطح دسترسی یک عضو کلاس را مشخص می‌کند. (اعضایی مثل متودها، صفات و یا داده‌ها)

- **public**: available at any place *(قابل دسترسی در همه جا)*
- **private**: available only within the class itself *(قابل دسترسی فقط در داخل خود کلاس)*

- **internal**: visibility is public in the containing assembly but private outside of it.

سطح دسترسی در اسمبلی جاری، public و در خارج از آن private است.

- **protected**: available within the class or a derived class.

قابل دسترسی در داخل کلاس جاری و کلاس‌های مشتق شده از کلاس جاری.

- **protected internal**: visibility is public in the containing assembly but protected outside of it.

دسترسی در اسمبلی جاری، public و در خارج از آن protected است.

فراخوانی یک متود:

```
namespace ConsoleApplication1
{
    public class MathOperations
    {
        public int AddOne(int x)
        {
            return ++x;
        }


        class Program
        {
            static void Main(string[] args)
            {
                int a = 4, b = 0;
                MathOperations n = new MathOperations();
                b = n.AddOne(a); // فراخوانی

                Console.WriteLine(b);
                Console.WriteLine(a);

                Console.ReadLine();
            }
        }
    }
}
```

خروجی:

b=5 , a=4

انواع فراخوانی:

۱- call by value: در این روش، *دقیقاً مقدار یک متغیر* به متود ارسال می‌شود و روی متغیری که ارسال شده است، تغییری ایجاد نمی‌شود. (به طور پیش فرض، این نوع فراخوانی در C# به کار گرفته می‌شود.)

۲- call by reference: در این روش *آدرس محل ذخیره سازی متغیر* به متود ارسال می‌شود. در نتیجه هر نوع تغییر در پارامتر ورودی باعث تغییر مقدار متغیری که در هنگام فراخوانی ارسال شده، می‌شود. برای این نوع فراخوانی از کلمه کلیدی ref قبل از نام متغیردر هنگام تعریف متود وفراخوانی استفاده می‌شود.

دلیل استفاده از فراخوانی با ارجاع:

از آنجا که یک متود تنها قادر به برگرداندن یک مقدار است می‌توان از فراخوانی با ارجاع این عیب را رفع کرد.

مثالی از فراخوانی با ارجاع (خروجی‌ها را با مثال قبل مقایسه کنید):

```
namespace ConsoleApplication1
{
    public class MathOperations
    {
        public int AddOne(ref int x)
        {
            return ++x;
        }


        class Program
        {
            static void Main(string[] args)
            {
                int a = 4, b = 0;
                MathOperations n = new MathOperations();
                b = n.AddOne(ref a);
                Console.WriteLine(b);
                Console.WriteLine(a);

                Console.ReadLine();
            }
        }
    }

}
```

خروجی:

b=5 , a=5

Data Members:

Data Members are fields or variables that belong to a class.

*در حقیقت **اعضای داده‌ای**، فیلدها یا متغیرهای تعریف شده در بدنه یک کلاس هستند.*

```
public class MathOperations
{
    private int x = 5;
}
```

☺ دو قاعده کلی:

پیشنهاد می‌شود:

۱- متودها public تعریف شوند.

۲- اعضای داده‌ای private تعریف شوند. (البته اگر سطح دسترسی آن‌ها تعیین نشود، به طور پیش‌فـرض در این حالت هستند)

نکته: اعضای داده‌ای به طور پیش‌فرض مقداردهی اولیه می‌شوند.

Properties:

Properties look like data members but they are in fact Methods.

صفات، *مانند* ***data member****ها به نظر می‌رسند اما در حقیقت متود هستند.*

ساختار کلی تعریف یک صفت:

```
[Modifier] returnType propertyName
{
    get {

        return variable;
      }
    set {

      }
}
```

☺ نکته:

۱- صفات، امکان کنترل بیشتری روی مقادیر داده‌ای فراهم می‌کنند. امکاناتی فراتر از انتساب و خواندن مقدار، امکان تست صحت مقدار، ثبت درDatabase و فقط-خواندنی یا فقط-نوشتنی کردن مقادیر و....

۲- بدنه کد باید شامل بلاک‌های set یا get یا هر دو باشد.

۳- اگر بخش set یک صفت حذف شود فقط-خواندنی می‌شود و اگر بخش get حذف شود فقط-نوشتنی می‌شود.

۴- مقدار یک صفت به بخش set ارسال می‌شود و با کلمه کلیدی value می‌توان به مقدار دسترسی داشت.

۵- به بلاک set ممکن است mutator (تغییر دهنده) و به بلاک get ممکن است accessor (دسترسی دارنده) گفته شود.

مثالی از تعریف دو صفت ID و Name برای کلاس Customer و مقداردهی به آن‌ها:

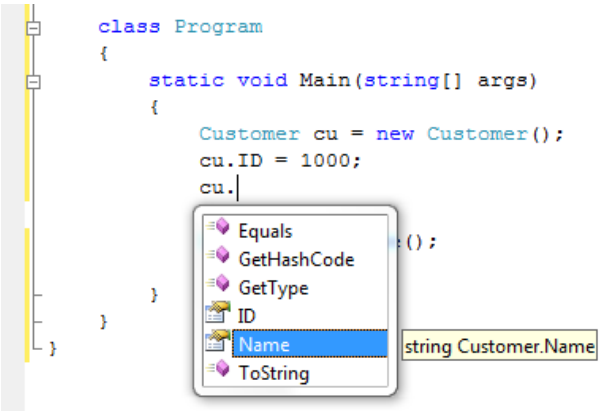
```
namespace ConsoleApplication1
{
    public class Customer
    {
        private int id;
        private string NAME;

        public int ID // تعریف صفت شماره شناسایی
        {
            get
            {
                return id;
            }
            set
            {
                ID = value;
            }
        }
        public string Name  // تعریف صفت نام
        {
            get
            {
                return NAME;
            }
            set
            {
                NAME = value;
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Customer cu = new Customer(); // تعریف یک شیئ از کلاس مشتری
            cu.ID = 1000;  // مقداردهی به صفت شماره شناسایی
            cu.Name = "Hamid Reza Niroomand"; // مقداردهی به صفت نام

            Console.ReadLine();
        }
    }
}
```

به تصویر زیر دقت کنید:



Method Overloading:

Overloaded Methods have the same name but different signatures.

***متودهای Overload شده**، نام‌های مشابهی دارند، اما امضای متفاوتی.*

- Signature یا امضای متود عبارت است از:
  i. نام پارامترهای متود
  ii. تعداد پارامترها
  iii. ترتیب پارامترها
  iv. نوع هر پارامتر

☺ نکته: اینکه از بین دو متود همنام، کدام متود مورد نظر ماست از signature که هنگام فراخوانی به متود ارسال می‌کنیم تشخیص داده می‌شود.

انواع Data Memberها:

1- Instance data member
2- Static data member

شرح: هنگام تعریف یک عضو داده‌ای آن داده بدون ایجاد شیئ قابل استفاده نبود در حقیقت آن داده مرتبط بود با آن شیئ. گاهی اوقات نیاز نیست به هر شیئ data member خاص نسبت دهیم، در این مواقع، آن داده را از نوع static تعریف می‌کنیم. یک داده static مثل متغیر global است.

Static data members (اعضای داده‌ای استاتیک) :

- belong to the class itself not to an instance of the class (a specific object)

یک عنصر داده ای از نوع static به خود کلاس تعلق دارد نه به یک نمونه از کلاس.

- are referenced by the class name

از طریق نام کلاس قابل دسترسی هستند.

- exist even when no instance of class exist

وجود دارند حتی اگر هیچ نمونه‌ای از کلاس وجود نداشته باشد.

☺ به طور مثال، متود WriteLine و ReadLine از نوع static هستند و برای استفاده از آن‌ها **نیازی به ایجاد شیئ نیست.**

Constructors: (سازنده‌ها)

A constructor is a Method that is called when a class is instantiated.

یک ***سازنده***، متودی است که هنگام مقداردهی اولیه به یک object فراخوانی می‌شود.

```
public class Customer
{
    string fn;
    string ln;
    int id;
    public Customer() // default constructor
    {
        id = 8890;
    }
    public Customer(string firstName, string lastName)// second constructor
    {
        id = 8890;
        fn = firstName;
        ln = lastName;
    }

}
```

```
////////////////////// in Main method we have:
Customer cu1 = new Customer();
Customer cu2 = new Customer("Hamid Reza" , "Niroomand");
```

☺ نکته:

۱- نام سازنده باید حتما هم نام کلاس باشد.

۲- سازنده نیازی به تعیین نوع مقدار برگشتی ندارد.

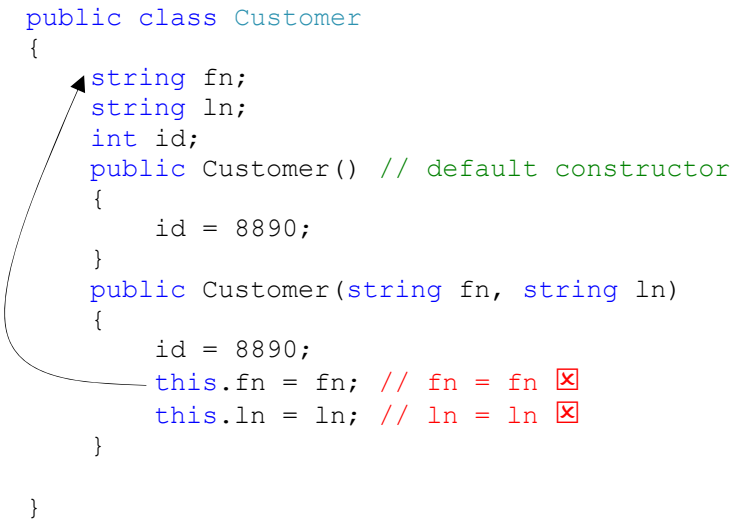
۳- سازنده‌ای که Zero parameters است سازنده پیش‌فرض کلاس است.

۴- اگر برای یک کلاس، سازنده تعریف نشود کامپایلر یک سازنده پیش‌فرض برای آن تعریف می‌کند تا Data memberها را مقداردهی اولیه کند. (مقدار اولیه Reference typeها Null است، آرایه‌ها و رشته‌ها از این نوع هستند.)

۵- اگر یک سازنده با بیش از یک آرگومان تعریف شود نمی‌توان از آن کلاس مثل حالت معمول به صورت صفر آرگومانی شیئ ایجاد کرد. در حقیقت کامپایلر برای کلاس‌هایی که یک سازنده با بیش از صفر آرگومان داشته باشند، سازنده صفر آرگومانی تولید نمی‌کند و آن را نیز کاربر باید تعریف کند.

☺ **نکته**: کلمه کلیدی this به شیئ جاری (Current Object) اشاره دارد.

دو متغیر fn که در برنامه زیر، همنام هستند، با this، متمایز شده‌اند:



```
public class Customer
{
    string fn;
    string ln;
    int id;
    public Customer() // default constructor
    {
        id = 8890;
    }
    public Customer(string fn, string ln)
    {
        id = 8890;
        this.fn = fn; // fn = fn ☒
        this.ln = ln; // ln = ln ☒
    }

}
```

Inheritance (وراثت)

the ability for one class to inherit behaviours or services from another class.

- مهم‌ترین مزیت وراثت، Reusability (قابلیت استفاده مجدد) است.
- هر کلاسی که در C# ایجاد می‌شود، حتماً از یک کلاس دیگر به ارث می‌برد. اگر کاربر برای یک کلاس،والد خاصی تعریف نکند، آن کلاس از کلاس System.Object به ارث می‌برد.
- با تعریف یک کلاس به عنوان فرزند کلاس دیگر، تمام «متودها، سفات و Eventهای public کلاس پدر برای فرزند نیز قابل استفاده خواهد بود. ضمن اینکه شما قادرید سفات، متودها و Eventهای بیشتری برای فرزند تریف کنید.
- هنگام استفاده از وراثت، شما یک رابطه بین دو کلاس برقرار می‌کنید.
  رابطه IS-A یا Parent/Child
- با استفاده از وراثت می‌توان یک کلاس پایه و عمومی (base or generalized) تعریف کرد و کلاس‌هایی را از آن مشتق کرد.
- رابطه بین کلاس مشتق شده و کلاس اصلی، با عبارت ISA توصیف می‌شود.

چند مثال:

Car ISA Vehicle
SalesManager ISA Employee ISA Person

- کلاس مشتق شده می‌تواند رفتار کلاس اصلی را ویرایش کند مثلا یکی از متودهایش را در اصطلاح Override کند.

ساختار کلی مشتق کردن یک کلاس از کلاس دیگر:

[Access Modifier] class *className* [: *baseClassName*]

مثال:

```
public class Manager : Employee
{
    //...........
}
public class Employee : Person
{
    //...........
}
public class Person
{
    //...........
}
```

خاصیت Single Inheritance:

در C# هر کلاس به طور مستقیم فقط از یک کلاس پایه به ارث می‌برد.

وراثت و سازنده‌ها:

وقتی یک شیئ ایجاد می‌شود سازنده کلاس مربوطه فراخوانی می‌شود. اولین کاری که هر سازنده انجام می‌دهد این است که سازنده والد (parent) خود را فراخوانی می‌کند. در این حالت سازنده پیش‌فرض (صفرآرگومانی) کلاس پایه فراخوانی می‌شود مگر اینکه کاربر تعیین کند کدام سازنده کلاس پایه فراخوانی شود.

مثال:

```
public class Manager : Employee
{
    //..........
}
public class Employee : Person
{
    int id_emp;
    public Employee()
    {
        id_emp = 3000;
    }
    public Employee(string FN, string LN): base(FN, LN)
    {
        id_emp = 3000;
    }
}
public class Person
{
    string fn;
    string ln;
    int id;
    public Person()
    {
        id = 3000;
    }
    public Person(string FN, string LN)
    {
        fn = FN;
        ln = LN;
        id = 3000;
    }

}
```