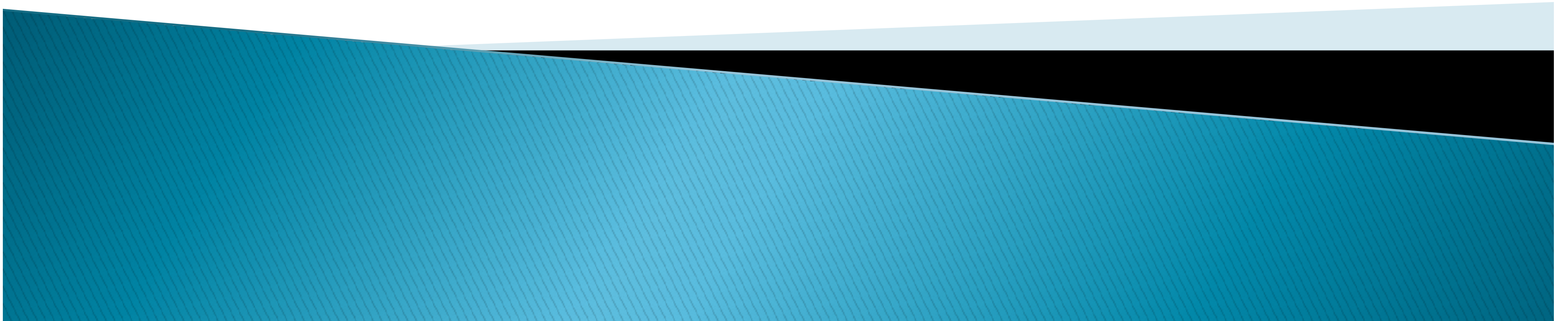
بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Object Oriented Programming (OOP)

جلسه دوم

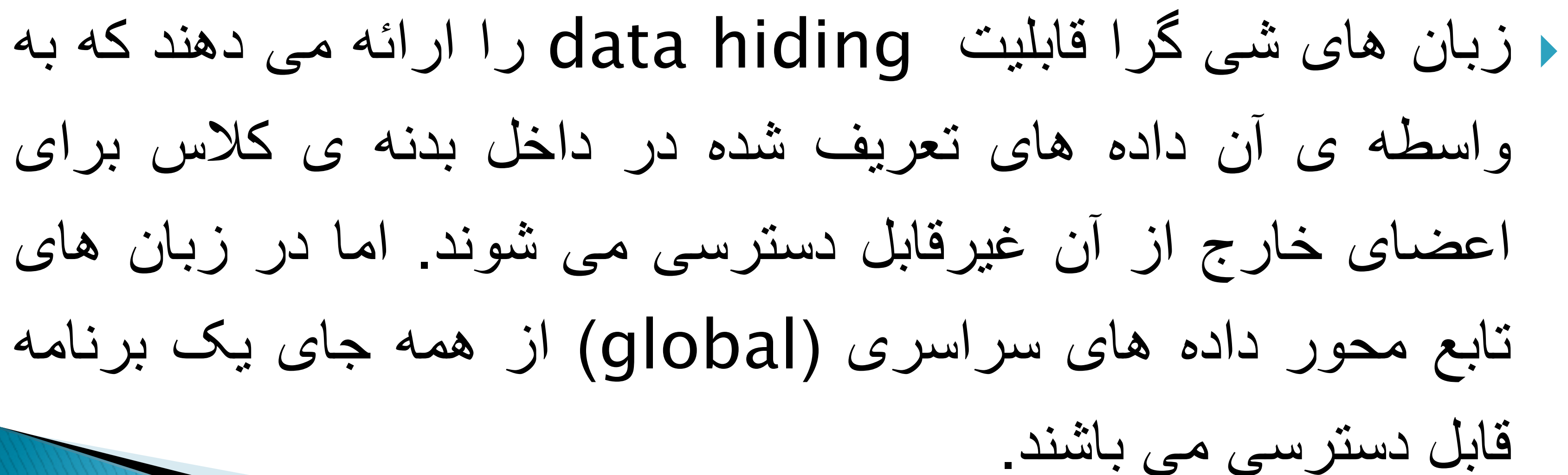
ملکی



برتری زبان های شی گرا بر زبان های تابع محور یا procedure-oriented

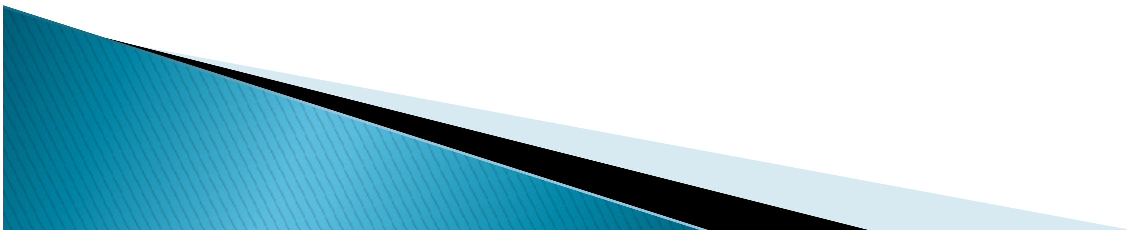
- زبان های شی گرا توسعه و نگهداشت نرم افزار را آسان می سازند. در صورتی که مدیریت کد برنامه های نوشته شده با زبان های تابع محور به محض بزرگ تر شدن پروژه بسیار طاقت فرسا می شود.

- زبان های شی گرا قابلیت data hiding را ارائه می دهند که به واسطه ی آن داده های تعریف شده در داخل بدنه ی کلاس برای اعضای خارج از آن غیرقابل دسترسی می شوند. اما در زبان های تابع محور داده های سراسری (global) از همه جای یک برنامه قابل دسترسی می باشند.



تفاوت بین زبان شی گرا و زبان مبتنی بر شی

- زبان های مبتنی بر شی (object-based) تمامی ویژگی ها و اصول OOP را به استثنای Inheritance یا وراثت رعایت می کنند. JavaScript و VBScript از این دسته از زبان ها هستند.

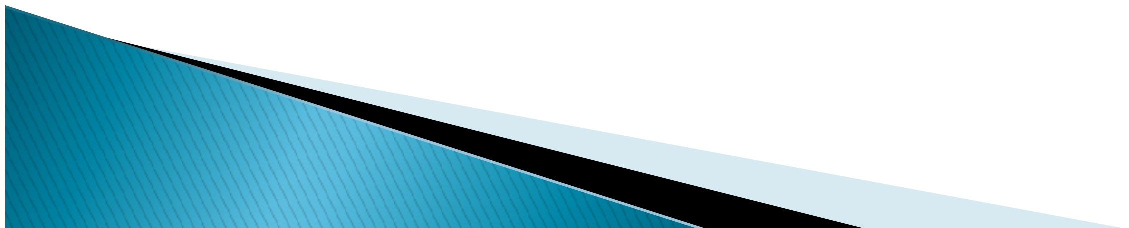


فیلدها ، متدها و سازنده ها

- فیلد متغیری است که داخل یک کلاس تعریف شده و امکان ذخیره مقداری را به ما می دهد....

- سازنده تابعی است که دارای سطح دسترسی public است و هم نام با نام کلاس می باشد و چون مکانیزمی ندارد که خروجی را برگرداند پس هیچ خروجی را بر نمی گرداند ولی می تواند ورودی داشته باشد.

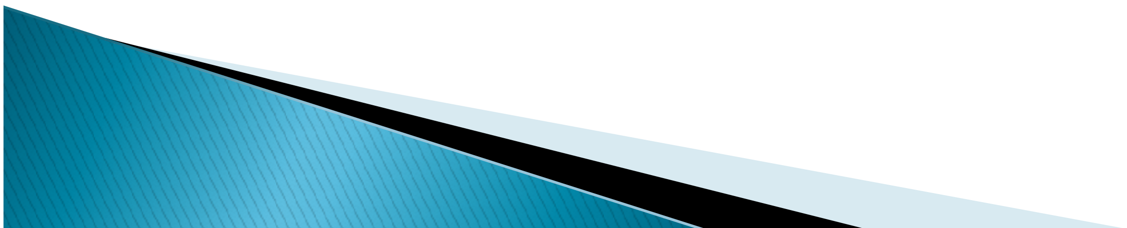
- به محض اینکه از روی کلاس یک شی ساخته شد تابع سازنده خود بخود اجرا می شود.



سازنده

```
public class Person
{
    public Person(string firstName, string lastName)
    {
        this.FirstName = firstName;
        this.LastName = lastName;
    }
```

```
person p = new Person("Hossein", "Ahmadi");
```

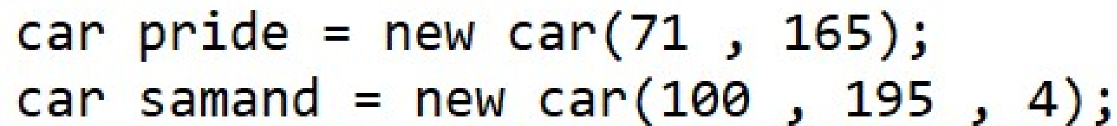


کلاس با چند سازنده

یک کلاس می تواند چندین تابع سازنده داشته باشد.

```
private void button1_Click(object sender, EventArgs e)
{
    Student st = new Student();
▲ 1 of 2 ▼ Student(int id, string name, string lastname)
}
```

```
public class car
{
   public car(int motor , int wheel)
     {
        this.Motor = motor;
        this.Wheel = wheel;
     }
   public car(int motor , int wheel , int airbag)
     {
        this.Motor = motor;
        this.Wheel = wheel;
        this.Airbag = airbag;
     }
}
```

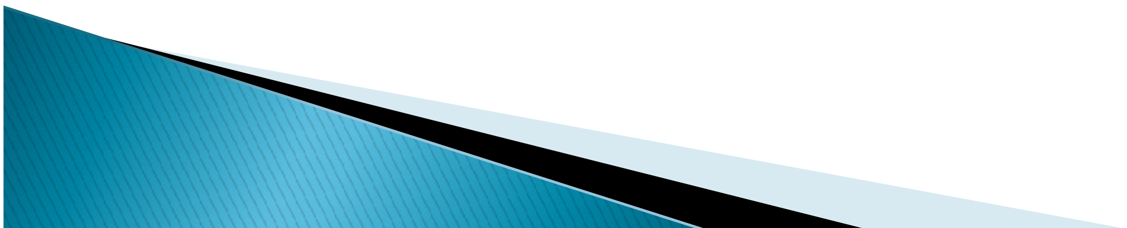


```
car pride = new car(71 , 165);
car samand = new car(100 , 195 , 4);
```

سازنده پیش فرض

- سازنده پیش فرض، سازنده ایست که هیچ پارامتری را به عنوان ورودی نمی گیرد.

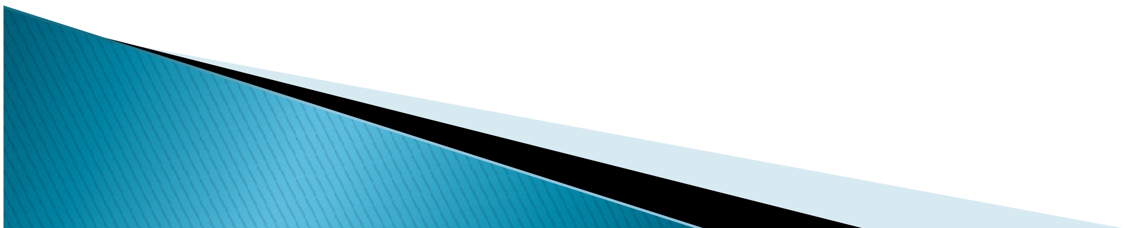
- مناسب ترین جا برای تخصیص مقادیر پیش فرض به اعضای کلاس، سازنده ی پیش فرض می باشد.



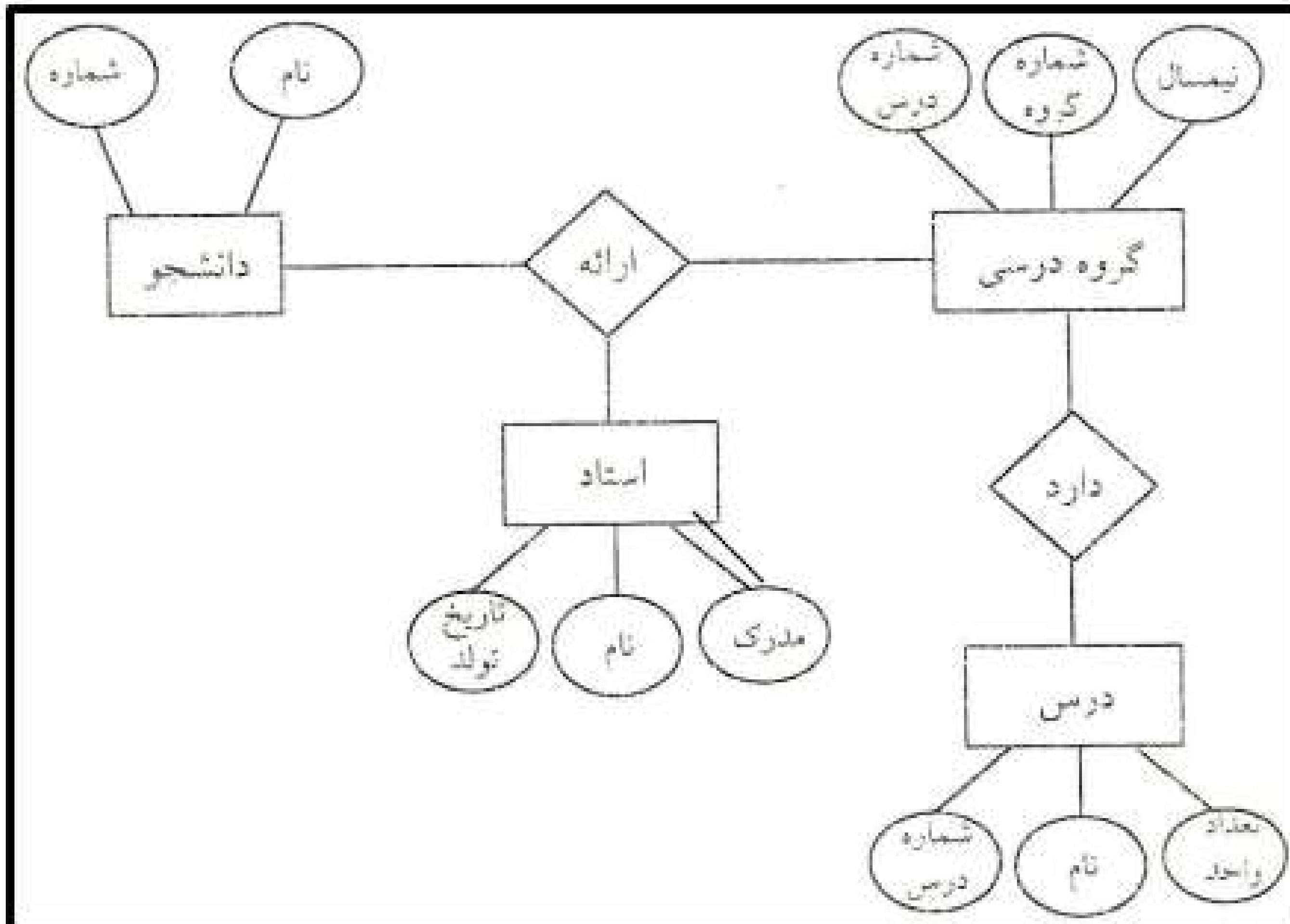
تابع مخرب

- تابعی است که وقتی object می خواهد از حافظه خارج شود خود به خود اجرا می شود و از اینرو هیچ ورودی ندارد.

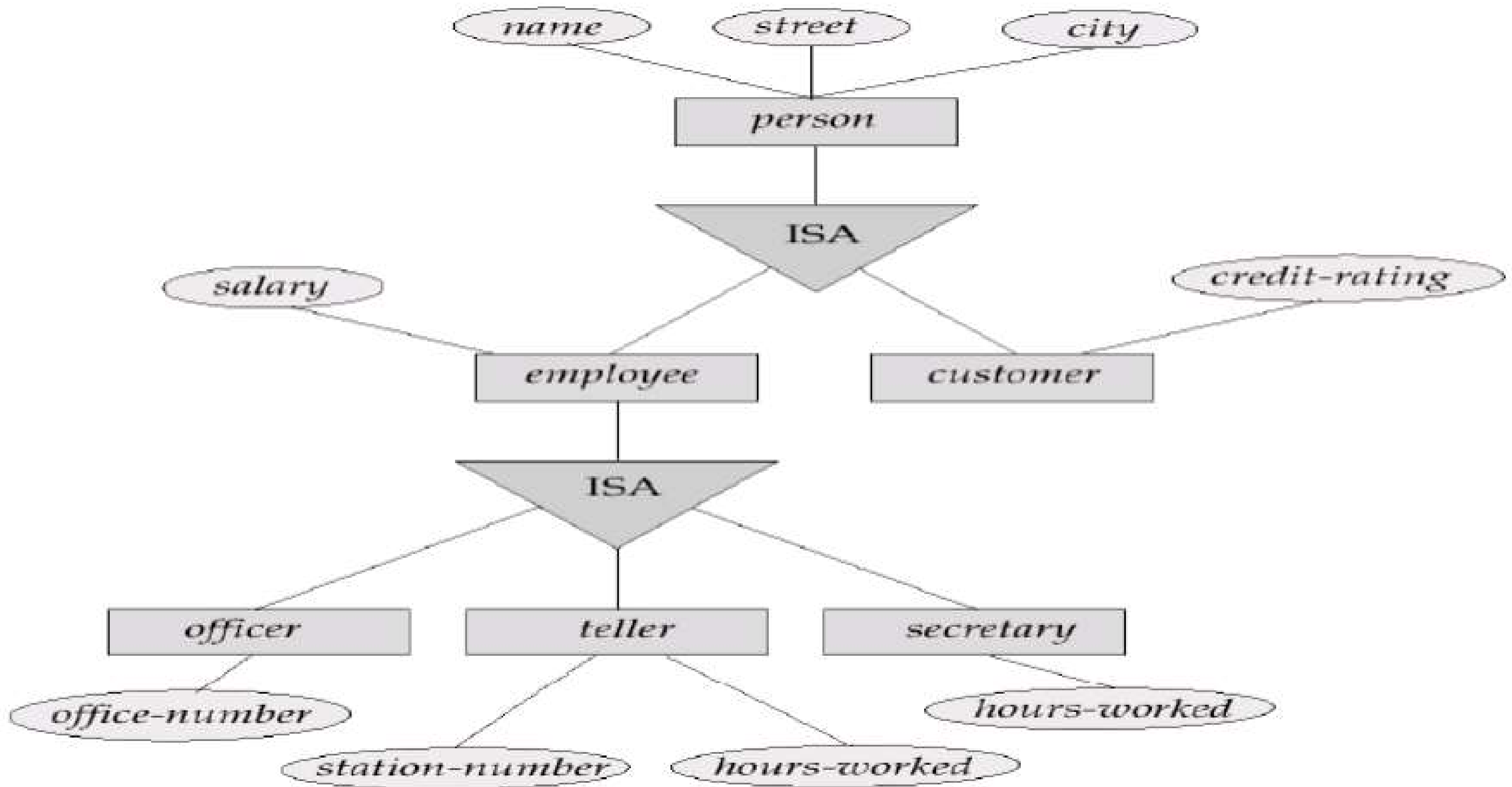
- در واقع فضایی که سازنده اختصاص داده بود تابع مخرب از شی می گیرد و آزاد می کند.



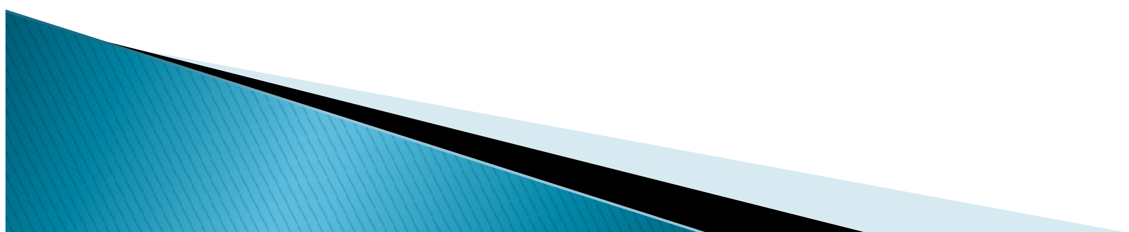
یادآوری



مقدماتی از وراثت



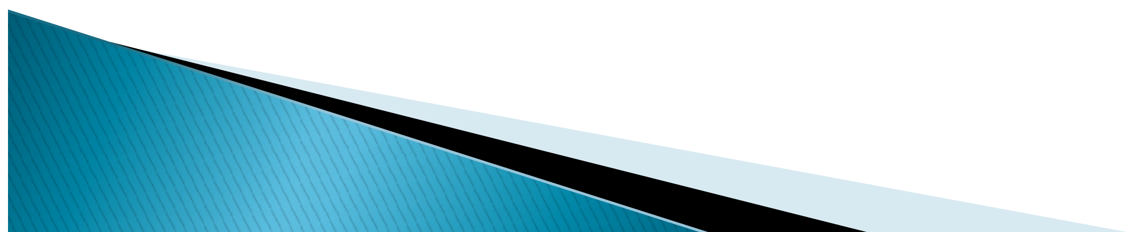
- وراثت برنامه‌نویس را قادر می‌سازد تا با ارث‌بری از صفات و متدهای یک یا چند کلاس موجود، کلاس‌های جدیدی را ایجاد نماید.



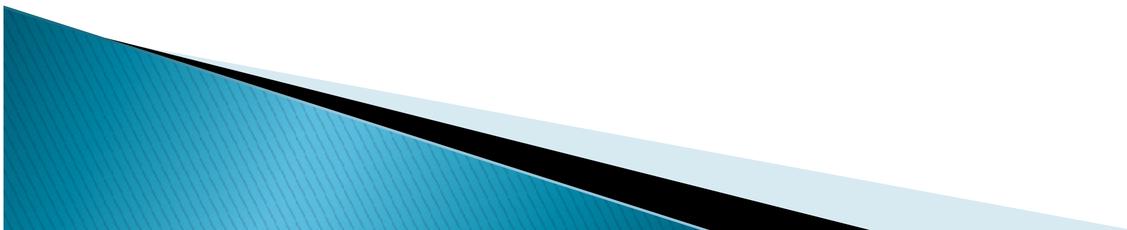
ضرورت مباحث انتزاع و...

- کلاس پایه ای داریم که این کلاس نباید مورد استفاده قرار بگیرد و تنها باید از کلاس های فرزند قابلیت ایجاد شئ وجود داشته باشد یا کلاسی نوشته ایم و نباید اجازه ایجاد کلاس فرزند از روی آن کلاس داده شود.

- این قابلیت ها بخصوص در مواقعی که در تیم شما، افرادی از کدهای نوشته شده توسط شما استفاده می کنند یا کدی را برای استفاده از سایر برنامه نویس ها بر روی اینترنت منتشر می کنید کاربرد دارند.



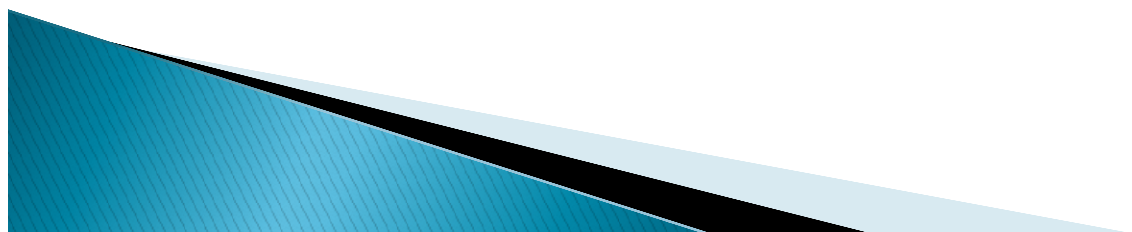
- راه حل برای این شرایط استفاده از کلاس های abstract و کلاس های sealed می باشد.



انتزاع (abstraction)

وقتی از منظر انتزاعی یا Abstraction به اشیا و مسائل نگاه می کنیم به این معنی است که:

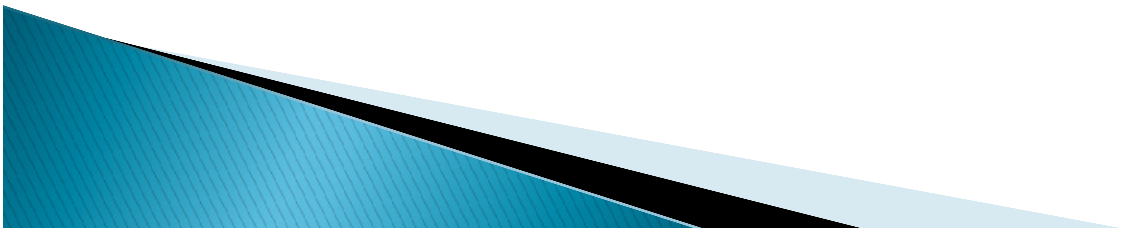
- به چیز های ضروری و اساسی دقت می کنیم.
- از جزییات و موارد بی ربط به اساس مساله پرهیز می کنیم.



تفاوت انتزاع و کپسوله سازی

یخچال و فریزر

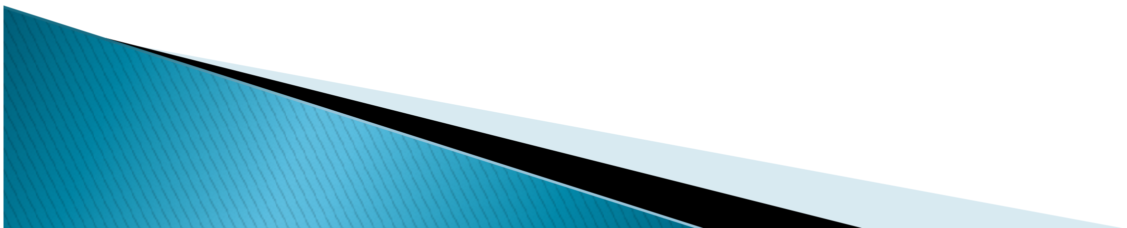
- انجماد یخچال و فریزر یک مفهوم است. ما فقط می دانیم که خنک میکند یا نه ! (Abstraction)
- چگونگی کار کردن آن را لازم نیست کاربر بداند (Encapsulation)



abstract class

- کلاس های انتزاعی (Abstract Class) که با کلمه کلیدی abstract مشخص می شوند، کلاس های پایه و مادر در یک سلسله مراتب درختی کلاس ها می باشند. به عبارت دیگر این کلاس ها، کلاس مرجع بوده و بقیه کلاس ها به ترتیب از روی این کلاس به ارث می روند.

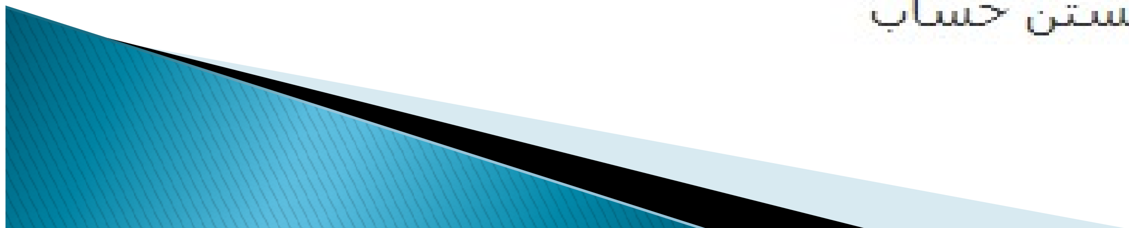
- **کلاسی که نمی توانیم از آن نمونه سازی کنیم و فقط در ارث بری میتوانیم از آن استفاده کنیم.**



مثالی جهت فهم انتزاع و کلاس های انتزاعی

کلاس حساب بانکی (مثلا بانک ملت) :

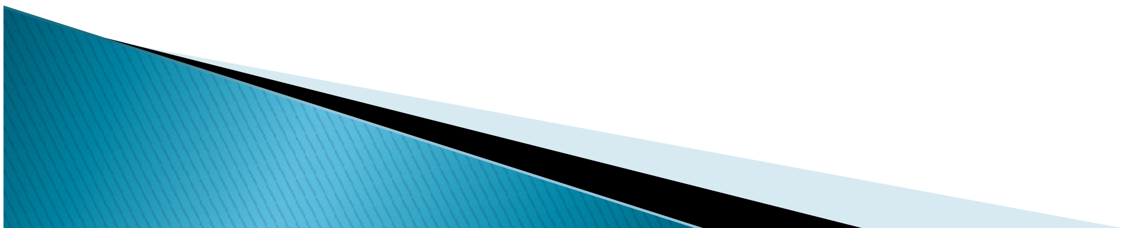
- صفت ها (attributes)
  - شماره حساب
  - نام صاحب حساب
  - موجودی
  - نوع حساب (جام، کوتاه مدت، طلایی)
- رفتار ها (behaviors)
  - افزایش موجودی
  - کاهش موجودی
  - افتتاح حساب
  - بستن حساب



توضیح...

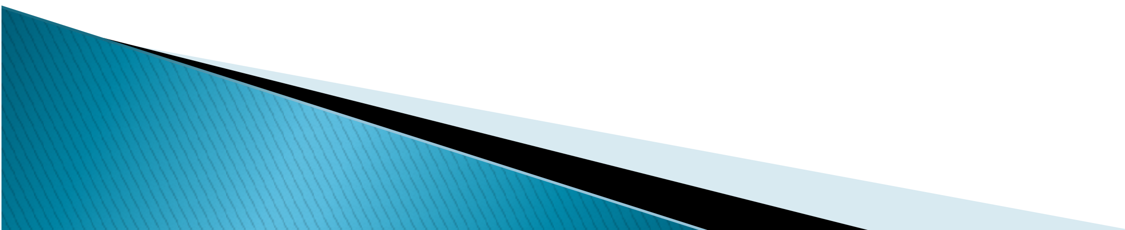
- فرض کنید در همان مثال حساب بانکی، بخواهیم یک موجودیت انتزاعی تر از این کلاس خلق کنیم که وابستگی به یک بانک خاص نداشته باشد.

- برای این منظور **می بایست دیدگاه انتزاعی خود را یک پله ارتقا دهیم** و آنچه که موجب وابستگی این کلاس به یک بانک خاص می باشد (در این مثال نوع حساب که ممکن اسن برای هر بانک متفاوت باشد) را حذف می کنیم.

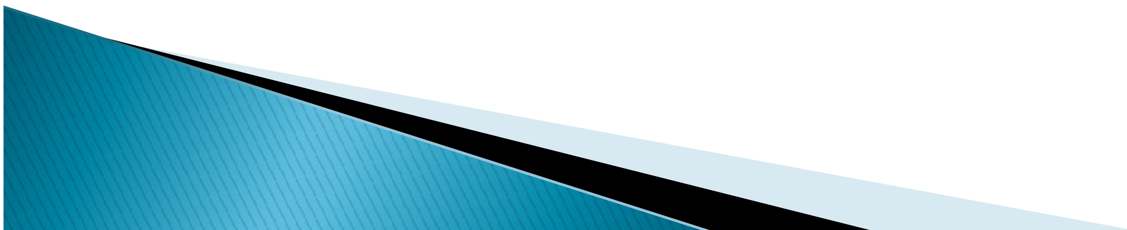


abstract method

اجبار برای کلاس های فرزند تا عملکردهایی را پیاده سازی کنند.

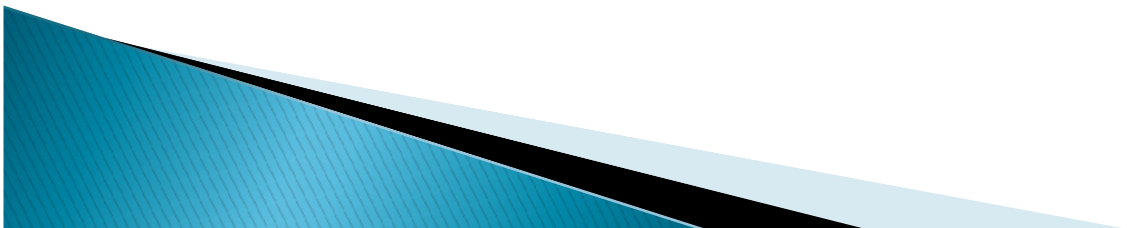


- در واقع کلاس های فرزند در یک سری رفتارها مشترکند ولی هر کدام از این کلاس های فرزند این رفتارهای مشترک را به نحوی متفاوت انجام می دهند.



نکته

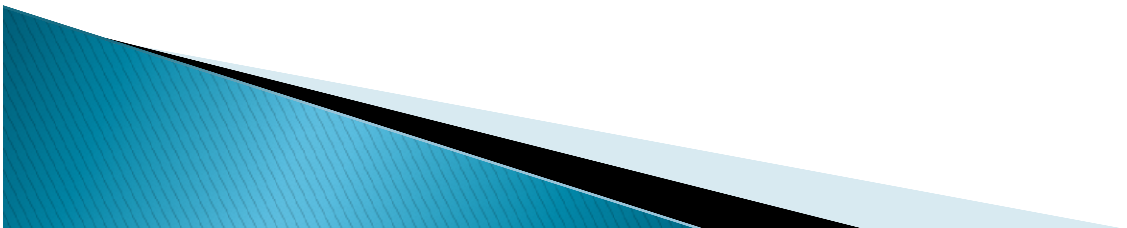
- اگر در یک کلاس، یک متود abstract تعریف شده باشد، خود کلاس نیز **حتما** باید abstract باشد، اما کلاسی را که به صورت abstract تعریف می کنیم، ضروری نیست که متود abstract داشته باشد.



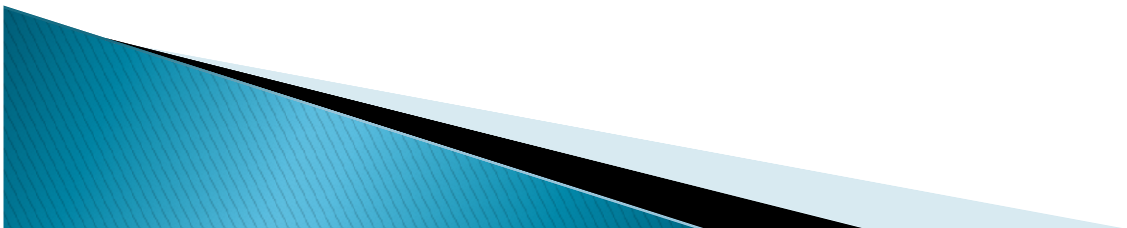
interface

- Interfaceها در #C نیز تا حد زیادی مشابه Abstract Classها بوده و در این ویژگی که نمی توان از روی آن یک نسخه یا شی ساخت با هم یکسان هستند.

با این حال، Interfaceها حتی از کلاس های Abstract نیز مفهومی تر هستند، و حتی نمی توان دستوراتی را برای متدهای تعریف شده تعیین کرد.

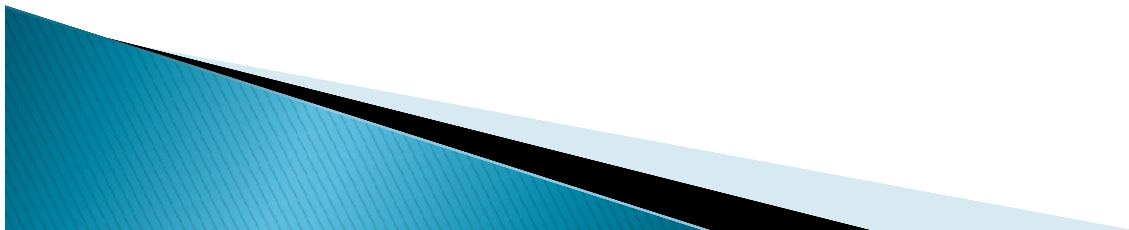


- نکته مهم درباره Interfaceها این است که از آنجایی که C# امکان به ارث بری چندگانه (multiple inheritance) را نمی دهد، یعنی یک کلاس از بیش از یک کلاس پایه به ارث برود، اما اجازه ای Interfaceهای چندگانه را می دهد.



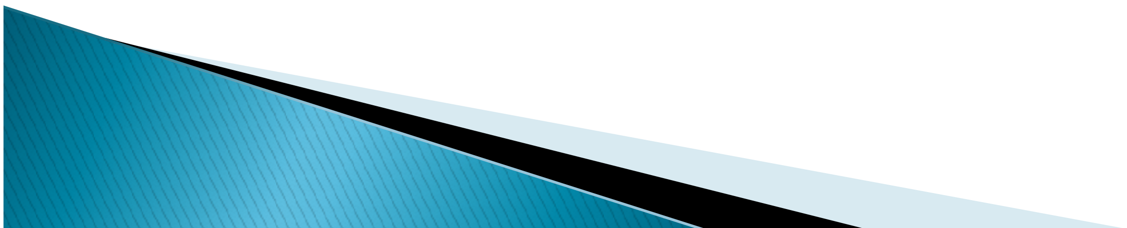
نکته

- سطح دسترسی access modifiers مثل public، private، protectedو ... در ابتدای نام Interface قرار داده نمی شود. زیرا این خواص در Interface ها مجاز نبوده و آن ها همه به صورت پیش فرض public هستند.

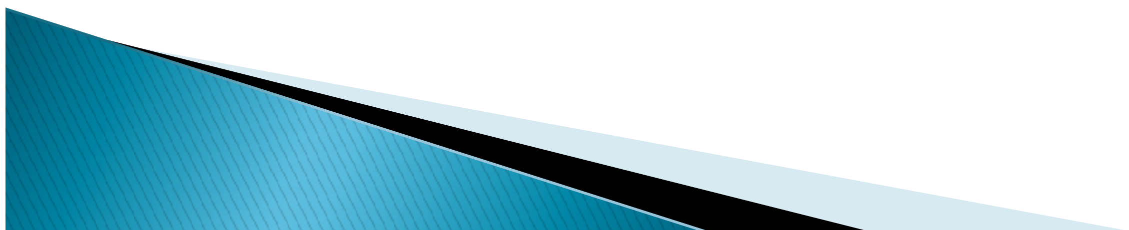


Structures

- همان‌طور که می‌دانید، کلاس‌ها reference type هستند. این بدان معنا است که اشیای کلاس از طریق یک reference قابل دسترسی هستند. از این‌رو reference typeها با value typeها که مستقیماً قابل دسترسی‌اند، متفاوت هستند.



- اما دسترسی مستقیم به یک شیء (به شکلی مشابه با value type ها) نیز گاهی می‌تواند سودمند باشد. یکی از دلایل این‌کار، افزایش بهره‌وری است.
- دسترسی به اشیاء از طریق reference باعث به وجود آمدن overhead(سربار) می‌شود و این overheadها فضا اشغال می‌کنند.
- برای رفع این نگرانی، سی‌شارپ structure را ارائه داده است.



- یک structure مشابه با class است با این تفاوت که structure، value type اما کلاس reference type است.
- structure ها نمی‌توانند از structureها یا از classهای دیگر ارث‌بری کنند.