زبان ساخت یافته:

زبان برنامه نویسی هستند که در آن برنامه نویسقدم ها و روال هایی را که لازم است تا برنامه به جواب برسد را مشخص می کند.

در این روش از برنامه نویسی انجام یک روال به روال های کوچکتر تقسیم می شود. به این ترتیب یک برنامه با شکسته شدن به ریز برنامه های کوچکتر سعی می کند تا عملکرد موردنظر را پیاده سازی کند.

مثال:زبان Basic,C،پاسکال

تمرین:

برنامه ای بنویسید که نمرات یک دانشجو را گرفته و معدل آن را حساب کند.

1)تعریف زیرروال جهت دریافت نمرات

2)تعریف زیر روال جهت جمع نمرات و محاسبه معدل

3)تعریف زیرروال برای چاپ خروجی و نمایش به صورت جدول

4)تعریف یک زیرروال برای اتصال به چاپگر و چاپ آن

هر اندازه که زیروال کوچکتر باشد درک و کدنویسی آن برای برنامه نویس راحتر می شود.

(هر زیر روال معمولا شامل 30 خط کد برنامه نویسی است.)

برنامه نویسی با نوشتن زیر روال در واقع بخشی از یک سیستم را پیاده سازی می کند.برنامه نویسان مختلف

می توانندبا قرار دادن زیر روال هایی که بر روی آنها کار کرده اند در نهایت به سیستم نهایی ختم گردد.در زبان ساخت یافته توابع کتابخانه ای فراوانی وجود داردکه به برنامه نویس در پیاده سازی روال کمک می کند.

مثال:در مسئله قبل در بخش چاپ خروجی توابع کتابخانه ای در زبان پاسکال یا C پیاده سازی شده است.

-در برنامه نویسی شی گرا توسعه برنامه راحت تر است.

-در ساخت یافته در ک مسئله بهتر است.

-ساخت یافته هنوز منسوخ نشده ولی برنامه نویسی شی گرا کاربردی تر است.

زبان های برنامه نویسی رویه ای:در این زبان برنامه نویسی برنامه نویس این امکان را دارد که با تقسیم برنامه به اجزایی پردازش مناسب را بر روی داده ها اعمال کند و این رویه ها هستند که ساختار کلی برنامه را تشکیل می دهند.

مزایا:

1-ساختار برنامه از طریق رویه ها پیاده سازی می شود.

2-با وجود توابع کوچکتر علاوه بر درک آسان خطایابی و اشکال زدایی آن آسان است .

معایب:

1-به دلیل جدا بودن داده ها از رویه ها در صورتیکه در یک تابع نحوه رفتار درستی با داده ها تعریف نشده باشد ممکن است با خطا مواجه شویم.

2-به دلیل سلسله مراتبی بودن ،تغییر در یک رویه باعث تغییرات کلی در برنامه شده ،در نتیجه نگهداری و پشتیبانی از این برنامه ها دشوار است.

برنامه نویسی ماژولار:

برنامه به تعدادی جزء (Component) یا ماژول تقسیم می شود. برخلاف برنامه نویسی رویه ای داده و رویه ها از هم جدا هستند .ماژول این دو را با هم ترکیب می کند..

در واقع ماژول داده ها و رویه هایی است که بر روی داده هااعمال می شوند.

نکته:تفاوت زبان رویه ای و ماژولار در جدابودن داده ها و رویه ها از یکدیگراست.

معایب:

1-امکان تغییرات افزایشی بر روی یک ماژول بدون دسترسی مستقیم به کدها امکان پذیر نمی باشد.

2-نمی تواند یک ماژول مبنایی برای ماژول دیگر باشد.

زبان برنامه نویسی شی گرا:

شیء(Object):

1-هر چیزی که مشخصه و رفتاری از خود بروز می دهد.

2-جزئی از یک نرم افزار است که رفتار و حالت را در خود نگهداری می کند.اشیاء به برنامه نویس این امکان را می دهد تا با استفاده از انها نرم افزار را به مدلی از دنیای واقعی تبدیل کند .

شی دارای دو پارامتر است:1-مشخصه 2-رفتار می باشد.

به طور مثال مجموعه اشیاء خودرو که شامل شی های زیر می باشدکه مقدار آنها متفاوت می باشد.

تعدادسیلندر ،مدل،رنگ ،شماره موتورو...

کلاس:یک کلاس رفتار و مشخصه های مشترک یک نوع شی که به اشتراک می گذارد را تعریف می کند.

نکته:در واقع کلاس یک الگو (قالب Template)می باشد که بر اساس آن شی تعریف می گردد.

نمونهInstanse: هر شی از روی کلاس ساخته می شود و درواقع بیانگر کلاس است.

هر گاه مشخصه های یک کلاس مقدار دهی شوند یک شی خاص و متمایز تعیین می گرددکه به آن نمونه گویند.

نکته:مشخصه ها را به عنوان متغیر تعریف می کنیم.

مثال:

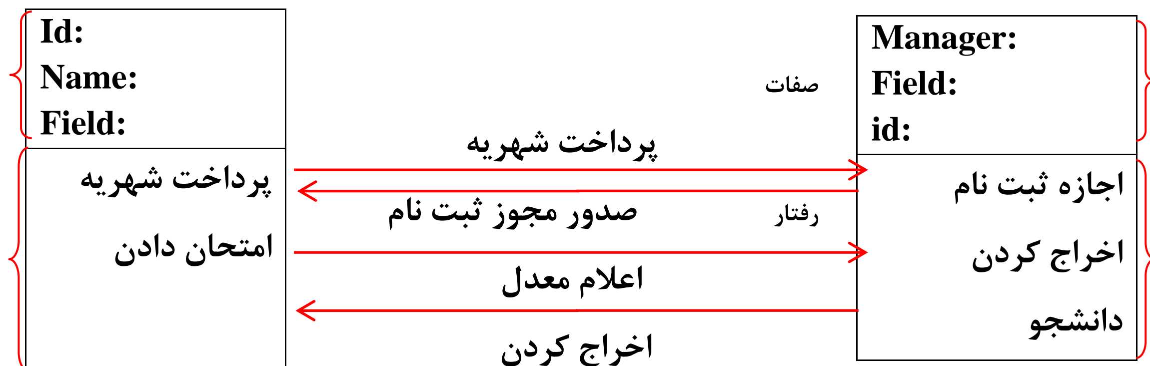
a = 0; ⟵ نمونه

صفات: مشخصه های قابل مشاهده یک کلاس را گویند.مثل رنگ ، مو ، وزن و...(برای شی انسان)

رفتار(Behavior): عملی است که توسط شی هنگامی که پیغامی به آن ارسال می شود و یا هنگامی که تغییرات رخ بدهد انجام می گردد.

آموزش

دانشجوstudent



متد(Method) : متدها مجموعه کدی هستند که در یک کلاس یک عمل خاص را انجام می دهند.

مثال: اگر کلاسی جهت کار با پایگاه داده تعریف شده باشد انگاه جهت برقرای ارتباط با پایگاه داده باید متدی را که به منظور اتصال به بانک اطلاعاتی که نوشته شده را فراخوانی می کنیم.

C#:

در سال 2001 توسط شرکت مایکروسافت ارائه شد هراه با بسته نرم افزاری net . که برای اولین بار مطرح می گردید ارائه شد و بعدها توسط ISO(International...)وECMA (European Computer Manufactures Association)مورد تایید قرار گرفت.

برنامه های تولید شده توسط net. توسط CLR(Common Language time) اجرا می شوند.

وظایف CLR:

1-مدیریت حافظه(Memory managment)

2-جمع آوری زباله ها(Garbage Collection)

3-رسیدگی به خطاها(Expection Handeler)

جمع آوری زباله ها(Gc): در زبان c# نگرانی از بابت مدیریت اشیاء وجود ندارد.بدین معنا که پس از ایجاد یک شیءو در صورتیکه بعد از مدتی از آن شیء استفاده نکنیم به طور اتوماتیک آن شی از حافظه پاک می شود.

نحوه اجرای برنامه(کامپایل کردن برنامه):

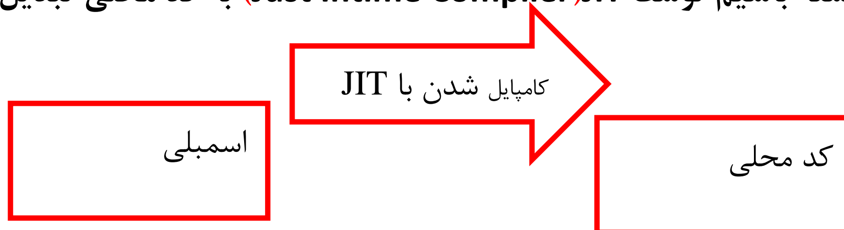
1-کد نوشته شده به زبان C#

کامپایل

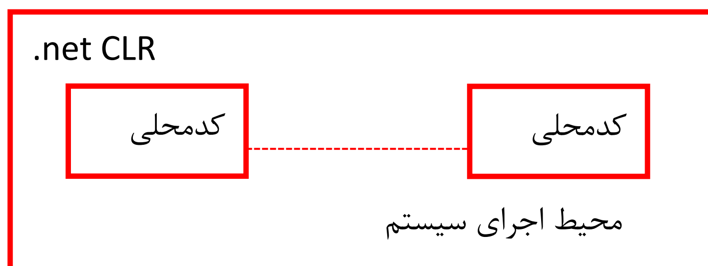
2-کد به زبان C# ← کد اسمبلی

کد به MSIL(Microsoft Intermediate Language) تبدیل شده و در یک فایل اسمبل ذخیره می گردد.

3-هنگامی که قصد اجرا داشته باشیم توسط JIT(Just Intime Compiler) به کد محلی تبدیل می شود.



4-کد محلی تولید شده به وسیله JIT به هراه سایر برنامه ها که با .net نوشته شده اند تحت کنترل CLR در می آیند.



معرفی مفاهیم در زبان C#:

در زبان C# بیشتر از 2500 کلاس از قبل تعریف شده و در اختیار برنامه نویس است که می تواند استفاده کند.

-Name space(فضای نام):

کاربرد آن جهت دسته بندی کلاسها و سهولت دسترسی برای استفاده است.

مثال:

System.web
System.Data

System .Data. sqlClient.sqlConnection

Solution: با لاترین سطح دسته بندی در محیط VS.net گویند .و فایلی را تولید می کند با پیوندsln. وظیفه آن نگهداری اطلاعات و روابط بین پروژه می باشد.

نکته:در محیط طراحی یا فرم اگر دکمه F7را بفشاریم وارد محیط کدنویسی می شویم.

کاربردUsing: نوشتن نام کامل کلاس کار دشواری است و برای رفع این مشکل با یک بار نوشتن اسم name space آن کلاس با کلمه using از تکرار جلوگیری می کنیم.

نحوه تعریف متغیر در زبان C#:

1-int a;

2-system.int32 a;

نحوه تعریف کلاس درC#:

```
Class name
{
    صفات (Data Member )        ⟶        هر متغیر که در محدوده کلاس تعریف شود
    متدها
}
```

مثال:در محیط عملیاتی که بانک باشد .حساب بانکی یک کلاس است.

نکته: زبان برنامه نویسی C# زبان حساس به حروف است.

نکته: استاندارد در تعریف کلاس این است که اسم کلاس با حروف بزرگ شروع می کنیم.

نکته: اسم متغیر ها در C# با حروف کوچک شروع می شود ولی اگر دو کلمه ای یا چند کلمه ای باشد کلملت دوم به بعد با حروف بزرگ نوشته می شود.

```
Class BankAccount
{                                            (عمومی)
Public    string    Account name;       نام حساب
Public    int       Accountfee;         کارمزد حساب
private   int       AccountBalance;
private   int       AccountNumber;      شماره حساب
                                             (خصوصی)
}                                            (محدوده کلاس)
```

کنترل اجازه دسترسی:کلاس دارای یک محدوده می باشد که با{} مشخص می گردد. و آنچه بین {}باشد درون محدوده کلاس محسوب شده و خارج از آن به عنوان برون کلاس شمرده می شود.

برای اینکه بتوانیم بر روی فیلدها و متدها قابلیت دسترسی و کنترل داشته باشیم از کلمات کلیدی public یا private استفاده می کنیم.

Public: اگر متد یا فیلد به این صورت تعریف شود در محدوده خارج از کلاس قابل استفاده می باشد.

Private: اگر فیلد یا متدی به این صورت تعریف شود تنها در داخل محدوده کلاس قابل دسترس است.

کپسوله سازی(encapsulation):

به مفهوم قرار دادن چیزها در یک محدوده می باشدو کپسوله سازی 2 کاربرد دارد.

1-باعث کنارهم قرار دادن داده ها و متدها در کنار یکدیگر (پشتیبانی از طبقه بندی)

2-کنترل دستیابی به متدها و داده ها (کنترل استفاده از کلاس)

نکته:اگر نحوه دسترسی به یک فیلد در کلاس معین نگردد (public یا private)به صورت پیش فرض آن فیلد به صورت private در نظر گرفته می شود .

نحوه تعریف فیلدی که دارای مقدار ثابتی برای تمامی اشیا است.

1-public int static interest rate; نرخ سود

2-با استفاده از کلمات کلیدی Readonly و Const می توان فیلدهایی با مقادیر ثابت تعریف کرد.

Public int Readonly day of weak=7; تعداد روزهای هفته

نکته: تعریف کردن فیلدها به صورت public کار درستی نیست ولی فیلدهایی که دارای مقدار ثابت هستند . تعریف کردن به صورت public ایرادی ندارد.

★نحوه تعریف یک شی از کلاس:

به 2 روش می توان تعریف کرد:

روش 1

Bank Account cost Account

نام کلاس نام شی

Cost Account =new Bank Account کلمه کلیدی به رنگ آبی

();

روش 2

Bank Account costAccount =new Bank Account();

★نحوه دسترسی به فیلدهای یک شی تعریف شده :

الف)فیلدهای public:

Cost Account.accountname="jari

اسم شی   فیلد از نوع public

نحوه تعریف متد در زبان C#:

متدها مجموعه ای از کدها هستند که می توانیم برای انجام کاری خاص آنها را فراخوانی کرد.

(متد مانند تابع است که یک ورودی و خروجی دارد که خروجی می تواند نداشته باشد)

عمومی ترین متدها در زبان C# برای تنظیم مقادیر (set) یا برگرداندن مقادیر (get) متغیرهای private هستند.

نکته:مثل متغیرها که به صورت private یا public هستند متدها هم به این صورت هستند.

```
Class BankAccount
{
Private account number;
Public int getAccountnumber( );
{
Return  accountnumber;
}
Public void set account number(int accnum)
{
Account number=accnum;
}
}
```

آرگومان های ورودی

برای آنکه خارج ازدسترسی داشته باشیم   خروجی تابع

خروجی ندارد   تنظیم مقدار   ورودی

نحوه مقداردهی به متدها:

```
BankAccount CostAccount=new BankAccount();
costAccount.Account num=""
cost Account.set Account number(12345);
int x;
x=cost Account . get account number();
```

مقدار account number را می خوانیم و در x می ریزیم

نکته:اگر نام متغیر ما با نام تابع یا متد ما یکی باشد خطا می دهد و برای رفع آن باید کلمه کلیدی this را اضافه کنیم.

مثال:

یک تابع است — آرگومان ورودی

```
Public void set Account number(int Account number)
{
This .account number=account number;
}
}
```

This می گوید برو داخل کلاس و این متغیر را پیدا کن

مثال:

```
Public class Account
{
Public string    Account name;
Public   int     Account fee;
Private  int     Account Balance;
Private  int     Account number;
Public   void    setAccountnumber(int acc num)
{
Account number=acc num;
}
```

```
Public  int  getAccountnumber()
{
Return  account number;
Static void main(string[]args)
{
Bank Account.custAccount=new Bank Account();
Cust Account.account name="jari";
Cust Account .account fee=10;
Cust Account. Set Account number(12345);
Console.writeline(Cust Account.Account name);
Console.writeline(Cust Account.Account fee);
Console.writeline(Cust Account.get Account number());
Console.readkey();
}
```

معرفی متد ویژه در کلاس به نام سازنده(Constructer)

هدف این متد این استکه با استفاده از این متد می توان هنگام تعریف یک شی فیلدهای آن (اعم از public یا private) را مقدار دهی کرد.این متد باید هم نام کلاس باشد.

Bank Account **CustAccount**=new BankAccount();

کلاس — یک شی از کلاس ایجاد شد

یک شی از کلاس BankAccount ایجاد شد.

```
Class    BankAccount
{
Public   void    BankAccount(string accname,int accnum)
{
Accountname=accname;
Accountnumber=accnum;
```

متد سازنده

}

}

BankAccount CustAccount=new BankAccount("Ali",12345);

علاوه بر تعریف شی فیلدهایی که در سازنده معرفی شده اندمقداردهی می گردد.

نکته:اگر در سازنده برخی از فیلدها به صورت صریح (آشکار)مقداردهی نگردد.مقدار پیش فرض آنها صفر یا falseیا null خواهد بود.

صفر → Int ,float,...

null → رشته ای

false → Boolean

مثال:

```
Console.writeline(custAccount.accountfee);
```

خروجی = 0

روش های Comment گذاری:

// → درج توضیح در یک خط

/* → درج توضیح در چند خط

*/ → برای پایان توضیح چند خطی

وراثت(inharitance):

بیان کننده رابطه ی بین کلاس ها می باشد.با استفاده از وراثت می توان کلاسهای جدیدی را از کلاس های موجود ایجاد کرد .کلاس جدید صفات و رفتار کلاس موجود را به خودش اختصاص داده یا با استفاده از آنها ویژگی یا صفات را اصلاح یا جایگزین می کند.

وراثت به دلیل استفاده مجدد از کلاسها (یکی از نکات مهم در مهندسی نرم افزار)موجب کاهش مشکلات در طراحی سیستم است.

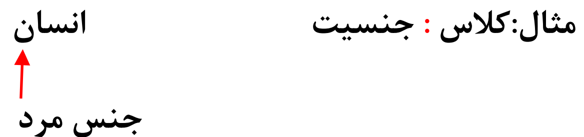
کلاس پایه(Base class):

کلاس موجودی که می توان صفات و رفتار از ان به ارث برد

مثل :کلاس انسان

کلاس مشتق(drived class) :

کلاس جدیدی که صفات و رفتار را از کلاس پایه به ارث می برد.



نکته 1:کلاس مشتق شده خاص تر از کلاس پایه است و گروه کوچکتری از اشیاء را شامل می شود

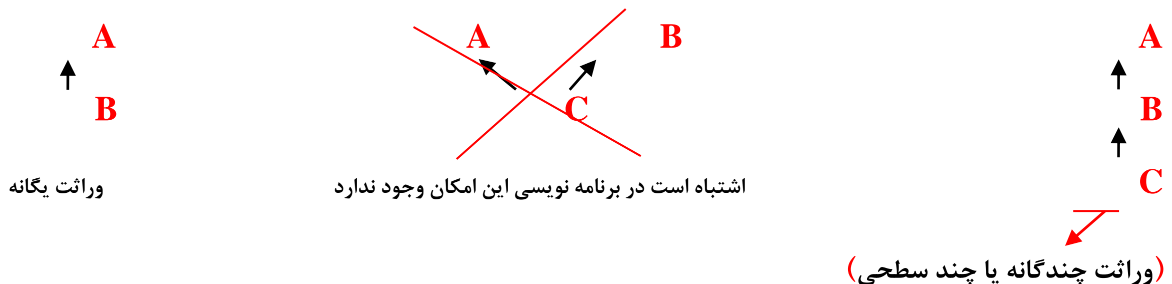
نکته2:کلاس مشتق شده علاوه بر صفات و رفتار کلاس پایه می تواند شامل صفات و رفتار خاص خود باشد.

وراثت چندگانه(multiple inharitance):

هرگاه یک کلاس مشتق شده خود به عنوان یک کلاس پایه برای کلاس دیگری باشد در این حالت وراثت چندگانه رخ داده است .در این وراثت کلاس مشتق از چندین کلاس به ارث برده است.

وراثت یگانه(Single inharitance):

هر کلاس مشتق شده صفات و رفتار را از یک کلاس به ارث می برد.

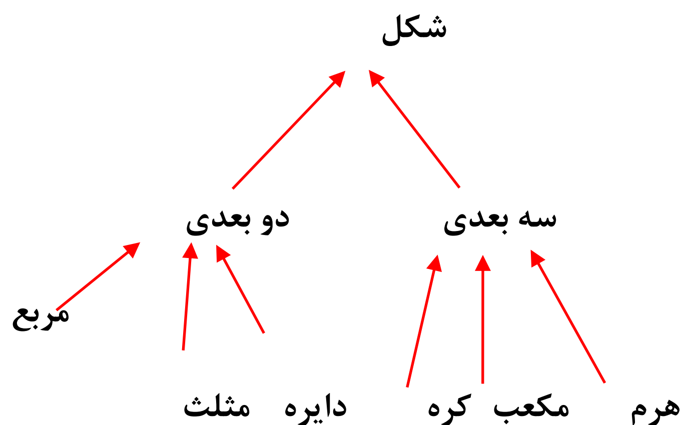


نکته:وراثت چندگانه مشکل بوده و احتمال بروز خطا در سیستم وجود دارد.

مثال:اگر کلاس چهار ضلعی را به عنوان کلاس پایه در نظر بگیریم آنگاه کلاس مستطیل یک کلاس مشتق شده از کلاس چهار ضلعی است .می توان گفت که هر مستطیل یک چهار ضلعی است. ولی امکان ندارد هر مستطیل یک چهارضلعی باشد.

نکته:وراثت ایجاد یک ساختار سلسله مراتبی است.

(بخشی از مراحل سلسله مراتبی کلاس شکل):



نحوه تعریف کلاس مشتق شده در #C:

```
Class Drivedclass  Baseclass
{
فیلدها
متدها
}
```

نکته:در #C از وراثت یگانه پشتیبانی می کند.

```
Class  SavingAccount : BankAccount
{
Public  double  interestRate;نرخ سود
Public  SavingAccount (string name,int num,int balance,double        سازنده
rate):Base(name,num)  ──►  دومتدسازنده
{
Account balance=balance;
```

```
interestRate=rate;
}
Public double mounty Rate()  →  برای محاسبه سود
{
Return interestRate*Account Balance;
}
}
```

نکته: سازنده ی کلاس مشتق شده باید سازنده ی کلاس خود را فراخوانی کند برای این کار از کلمه کلیدی base استفاده می کنیم.

نکته: اگر صریحا سازنده کلاس پایه را در سازنده کلاس مشتق شده فراخوانی نکنیم خود کامپایلر سعی می کند تا سازنده پیش فرض کلاس پایه را در کلاس مشتق فراخوانی کند.

نکته: در زبان برنامه نویسی C# کلاس system.object ریشه تمامی کلاس ها می باشد به عبارت دیگر همه کلاس ها به صورت ضمنی از کلاس object به ارث می برند به همین دلیل است که 4 متد Gettype(),Enqual(),Tostring(),Gethashcode() در تمامی کلاس ها یافت می شود.چون این متدها از کلاس object به ارث می برند.

نکته:

```
Class  BankAccount : system.object  →  از ارث می برد.
{
CustAccount.  لیست فیلدهاو متدها
}
```

نکته: در C# می توان متدها و فیلدها را به صورت محافظت شده protected تعریف کرد آنگاه آن فیلد و متد در آن کلاس و تمامی کلاسهایی که ازآن مشتق شوند دسترسی خواهند داشت و خارج از کلاس دسترسی نداریم .

مثال:

```
Using  system;
Class   parentclass()
{
Public  parentclass()  →  سازنده
{
Console.writeline("parent  constractor");
}
Public  void  print()  →  متد
```

```
{
Console.writeline("I'm   parentclass");
}
}
Class  childclass  :  parentclass
}
Public  childclass()   →                متد سازنده کلاس مشتق شده
{
Console.writeline("child  constractor");
}
Public  static  void   main()
{
Childclass  child = new   childclass();
Child.print();      شی
}
  فراخوانی متد print از parentclass
```

خروجی ⬅ parent constractor

Child constractor

I'm   parentclass

نکات:

1-کلاس فرزند توانایی های کلاس والد را دارد .

2-در این مثال با اینکه در Childclass متد print تعریف نشده بود اما آن را از parentclass به ارث برده بود.

3-هنگام ساختن یک شی از کلاس مشتق شده ابتدا به صورت اتوماتیک یک نمونه از کلاس والد ساخته می شود.

```
Class BankAccount
{
Public BankAccount (string name,int num)
{
}
Public BankAccount()
{
}
```

مثال:

```
Using  system;
```

```
Class   parentclass()
{
String  parentshring;
Public  parentclass()
{
Console.writeline("parent  constractor");
}
Public  void  print()
{
Console.writeline("I'm   parentclass");
}
Public   parentclass(string  mystring)
{
Parent  string = mystring;
}
Class  childclass  :  parentclass
}
Public  childclass():base("Drived  from")
{
Console.writeline("child  constractor");
}
Public  void   print()
{
Base.print();
Console.writeline("I'm  childclass");
}
Public  static  void  main()
{
Childclass   child=new   childclass();
Child.print();
((parent)child).print();
}
```

```
Class  parent
{
```

فیلد از نوع رشته

دوتا سازنده
- بدون آرگومان ورودی
- برای مقداردهی فیلدها

متدprint

به ارث می برد

```
Class   child:parent
{
Public  child(){}
Public   print() {base.print();}
Public  static   void   main()

{
Child    child = new    child();
Child.print();
((parent)child).print();
}
```

متد در کلاس پایه

نکته:متد همانند یک تابع می باشد و یک ورودی و یک خروجی دارد.

ورودی ← متد ← خروجی

نکته: هرگاه خواسته باشیم تابعی که در کلاس والد تعریف شده است را در کلاس فرزند با تعریف دیگری ارائه دهیم .در این صورت تابع تعریف شده در کلاس فرزند تابع هم نام در کلاس والد را مخفی می کند و دیگر تابع والد فراخوانی نمی شود .

نکته:هرگاه بخواهیم به متدی در کلاس پایه دسترسی پیدا کنیم باید آن را با کلمه کلیدی **base** فراخوانی کنیم.

روش دیگر جهت دسترسی به تابع در کلاس پایه بصورت زیر است:

متد هم نام.(نام شئ ایجاد شده از کلاس مشتق(کلاس والد)) ← به متد کلاس پایه دسترسی پیدا می کند.

مثال:

((Parent) child).print (); ← یعنی **child** در واقع تبدیل به همان **parent** شده و از متد **print** استفاده می کند.

Parent: کلاس والد

child: شئ

نکته:به این حالت **Boxing** گویند.

Boxing یا Casting:

برای تبدیل نوع های مختلف در زبان #C می توان از پرانتز و سپس ذکر نوع اصلی استفاده کرد.

مانند مثال قبل

مثال:

```
double a=13.4;        // کلاس والد
int b;                // کلاس مشتق
B= (int) a;           // A به int تبدیل می شود.
Console.writeline (b);
```

خروجی ← 13

نکته: به این حالت **unboxing** گویند.

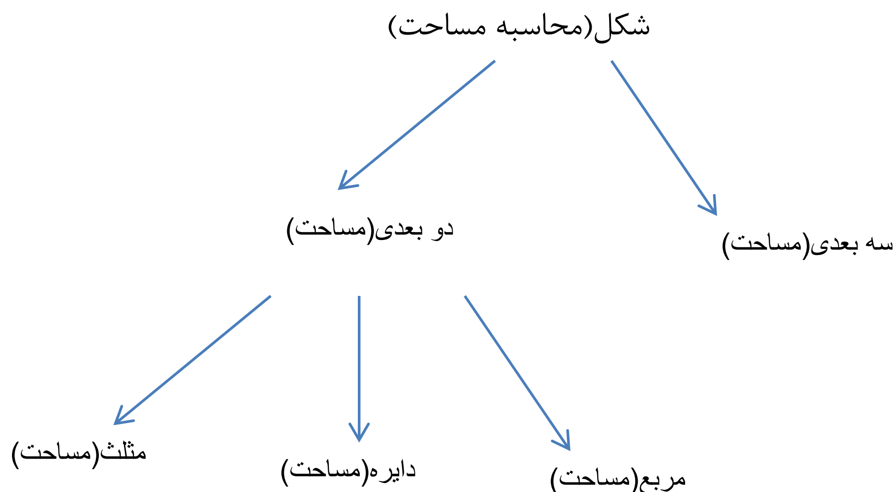
نکته: اگر کلاس مشتق به کلاس والد تبدیل شود **Boxing** و در غیر اینصورت یعنی والد به مشتق تبدیل شود **unboxing** گویند.

Boxing ← تبدیل مقدار نوع داده ای به نوع مرجع

Unboxing ← تبدیل نوع مرجع به نوع اصلی

چند ریختی یا چند شکلی(Polymorphism):

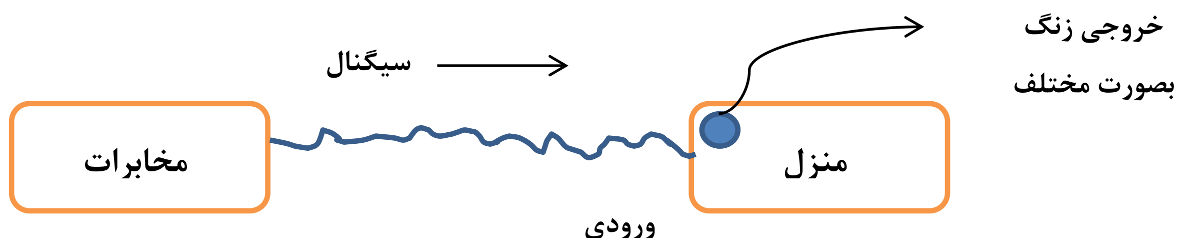
به معنای توانایی استفاده کردن از فرم های مختلف یک نوع است.البته بدون توجه به جزئیات آن.در برنامه نویسی یعنی بتوان یک متد را بیش از یکبار پیاده سازی کنیم.



یک تابع مساحت است که برای همه همنام است و به شیوه های مختلف پیاده سازی شده است.

مثال:

هنگام دریافت سیگنال از مخابرات،مخابرات به نوع تلفنی که در انتهای خط وجود دارد توجهی ندارد.مخابرات تنها نوع پایه ای بنام **phone** می شناسد و فرض می کند که هر **instance** از این نوع می داند که چگونه صدای تلفن را درآورد.



با استفاده از چند ریختی ما می توانیم در کلاس پایه و مشتق دو متد هم نام را که به یکدیگر هیچگونه ارتباطی نداشتند را مرتبط کنیم.در واقع این مطلب بیان کننده این است که دو پیاده سازی مختلف از یک نوع متد ارائه داده ایم .

مثال:متد print در مثال قبل از قانون چندریختی استفاده می کند.

```
Class shape
{
Protected int x,y;
Public void virtual M()
{
Console.writeline("Base");
}
}
Class Rectangle:Shape
{
```

```
Public void override M()
{
Console.writeline("Pholymorphism");
}
Public static void Main()
{
Rectangle r=new Rectangle();
r.M();
}//end Main  ⟶  پایان Main
}//end class  ⟶  پایان class
```

نکته:کلمه کلیدی virtual بیان کننده این مفهوم است که این متد اولین پیاده سازی متد مطرح شده است.(که معمولا در کلاس والد است.)برای اینکه متد Polymorphism را پشتیبانی کند باید کلمه کلیدی virtual را در کلاس پایه تعریف کنیم.

نکته:در کلاس های مشتق قبل از اسم متد هم نام کلمه کلیدی override را اضافه می کنیم.

نکته:در زبان C# متدها بصورت پیش فرض virtual نمی باشد.(در زبان جاوا هر متد بصورت پیش فرض virtual می باشد.)

نکته:نمی توان یک متد private را به همراه کلمات کلیدی virtual و override تعریف کرد.اگر این کار انجام شود خطای زمان کامپیال دریافت می کنیم.

نکته:دو متد باید مثل هم باشند.یعنی اسم نوع پارامترها و نوع برگشتی آنها یکسان باشد.

نکته:دو متد باید اجازه دستیابی یکسان ارائه دهند.

مثال:اگر یکی از متدها public باشد متد دیگر نیز باید public باشد.

مثال:برنامه ای بنویسید که طول و عرض یک مستطیل را از ورودی گرفته و محیط و مساحت آنرا محاسبه و چاپ کند.(با تعریف کلاس پیاده سازی شود.)

```
Class Rectangle
{
Public int x,y;
Public void Mohit()
{
Console.writeline((x+y)*2);
}
Public void Masahat()
{
Console.writeline(x*y);
}
}
Static void Main()
{
Console.writeline("please enter x,y:");
Rectangle r=new Rectangle();
r.x=int.parse(console.readline());
r.y=int.parse(console.readline());
r.Mohit();

r.Masahat();
console.readkey();
}
}
}
```

آرگومان ورودی ندارد.

چون متد سازنده تعریف نکردیم.

نکته:برای تبدیل نوع رشته به int از تابع زیر استفاده می کنیم:

```
r.x=int.parse(console.writeline());
```

برای تبدیل نوع رشته به int

از نوع رشته

نکته:فقط می توانید متد virtual را override کنید.اگر در متد کلاس پایه virtual نباشد و شما سعی کنید آنرا override کنید یک خطای زمان کامپایل در یافت می کنید.

نکته:اگر کلاس مشتق شده متد را با استفاده از کلمه کلیدی override تعریف نکنید متد کلاس پایه override نمی گردد.بلکه متد مورد نظر در کلاس مشتق یک متد کاملا متفاوت است که تنها اسم آن با متد کلاس پایه یکی است.البته باعث می شود که کامپیالر به شما اخطار دهد که برای رفع آن می توانید از کلمه کلیدی new استفاده کنید.

مثال:

```
Public new void M()
{}
```

نکته:اگر دو متد با یکدیگر همنام باشند مانند متد M در مثال زیر و چون دو کلاس ما از یکدیگر مشتق شده اند بنابراین نباید همنام باشند.و اگر خواستیم همنام باشند و از چندریختی تبعیت نکنند.حتما باید از کلمه کلیدی new استفاده کنیم.و اگر کلاس ما مشتق نشده باشد ایرادی ندارد که دو متد مورد نظر همنام باشند.

مثال:

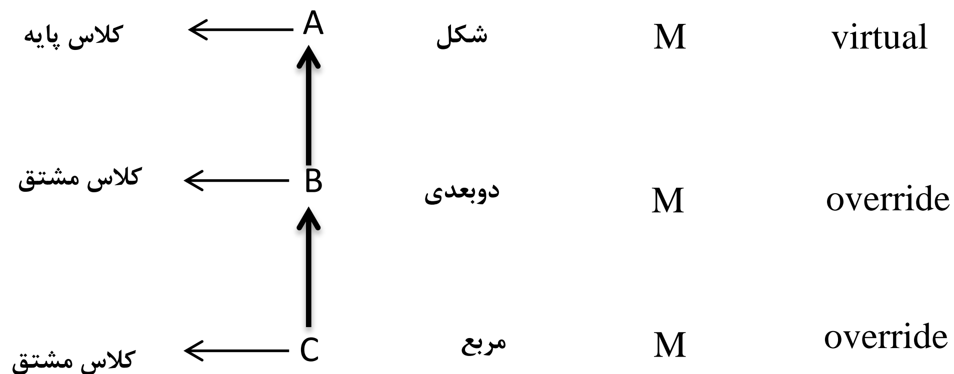
```
Class Shape
{
Public        void M()   ---->   کلاس پایه
{
Console.writeline("Base");
```

```
}
}
Class Rectangle
{
Public new void M()
{
Console.writeline("Polymophism");}}
```

Public new void M() ⟶ کلاس مشتق

نکته: یک متد override بصورت ضمنی virtual است و می تواند در کلاس مشتق شده از این کلاس override شود.ولی نمی توانید صریحا و با استفاده از کلمه کلیدی virtual مشخص کنید که یک virtual,override است.

مثال:



مثال: با توجه به مثال قبل اگر بخواهیم کلاس C را از B مشتق کنیم بصورت زیر تعریف می شود.

```
Class circle:Rectangle
{}
```

Rectangle ⟶ ارث می برد.

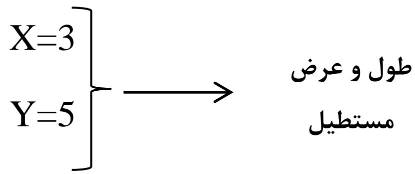
فقط از یک کلاس به ارث می برد و نیازی نیست که اسم کلاس پایه اصلی ذکر شود.

مانند :

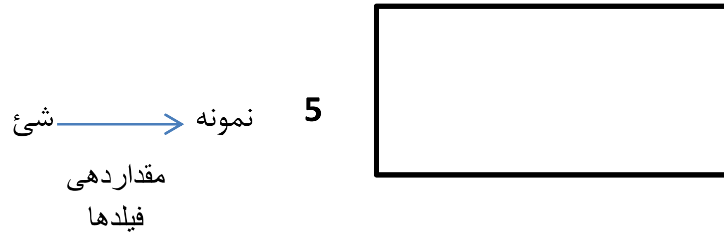
circle:Rectangle:shape ✗

Cirle:Rectangle ✓

نکته:زمانی که یک شئ فیلدهایش مقدار بگیرد تبدیل به instance یا نمونه می گردد.

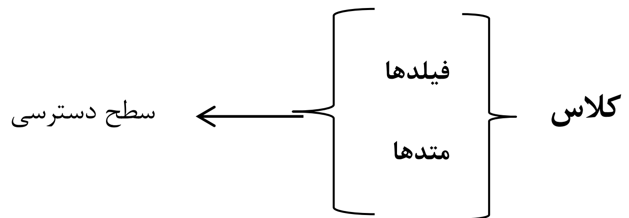


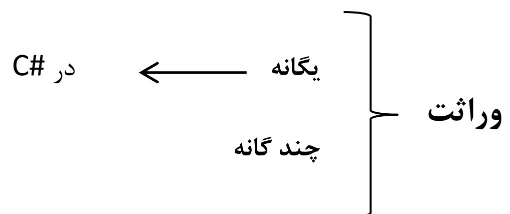
فقط مساحت مستطیل فوق محاسبه می شود ویک نمونه است



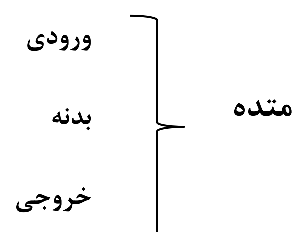
مثال: برنامه ای بنویسید که نام کارمند،میزان ساعت کاری در ماه،میزان حق العمل هر ساعت را دریافت کرده و حقوق ماهیانه را محاسبه و چاپ کند.(با استفاده از مفاهیم کلاس)

یادآوری:





کلاس پایه و مشتق ← پلی مورفیسم



Interface(واسط):

ارث بردن از یک کلاس مکانیسم قدرتمندی است.قدرت واقعی وراثت از ارث بردن از یک Interface بدست می آید. تقریبا مثل یک قرداد است.(فقط اسم است.)و این امکان را می دهد که اسم متد کاملا از پیاده سازی آن جدا گردد.

تا اینجا باید اسم متد همراه با پیاده سازی آن باشد.ولی از اینجا به کمک Interface امکان جداسازی تعریف متد از پیاده سازی آن فراهم می گردد.

در واقع Interface یک نوع قراردادی است که به موجب آن زیر کلاسهایی که از آن به ارث

می برند موظف به پیاده سازی بندهای این قرارداد می باشند.در Interface کلیات این قرارداد می آید و جزئیات در زیر کلاسها مطرح می گردد.

جهت تعریف یک Interface در زبان C# از کلمه کلیدی Interface استفاده می شود.

در یک Interface مانند یک کلاس متدها تعریف می گردد با این تفاوت که نیازی به تعیین سطح دسترسی(یعنی کلمات Public,Private,Protected)نمی باشد.و به جای پیاده سازی بدنه آن بلافاصله بعد از تعریف متد از(;)استفاده می کنیم.

مثال:نحوه تعریف Interface

```
Inter face Iwrite
{
Int I; ✖
Void print();}
```

نکته:در Interface تعریف فیلد نداریم.

Int I;⟶ خطا می دهد.

نکته:در زبان **C#** برای تعریف Interface،در ابتدای اسمی که انتخاب می کنیم I قرار می دهیم.

نکته: برای اینکه بتوانیم از وراثت چندگانه پیروی کنیم از interface استفاده می کنیم.

نکات:

1- نمی توان از یک interface یک نمونه ایجاد کرد.

```
Iwrite i=new Iwrite();
i.Print();
```

خطا می دهد.
(اشتباه است.)

اگر مجاز به این کار بودیم فراخوانی متدی که اصلا پیاده نشده معنی نداشت.

2- در یک interface نمی توان فیلد تعریف کرد.چون فیلد در واقع پیاده سازی یک صفت از یک شئ است.حتی نمی توان فیلد Static در interface تعریف کرد.

3- در interface نمی توان سازنده تعریف کرد.چون سازنده برای مقداردهی اولیه به یک نمونه از کلاس بکار می رود.در حالی که ما نمی توانیم نمونه ای از interface تعریف کنیم.

4- نمی توان از مشخصه های Public,Private,Protected استفاده کنیم.و تمامی متدها در interface بصورت ضمنی Public هستند.

پس در کلاسی که از interface به ارث می بردمتدها باید صریحا بصورت Public تعریف گردد.

نکته: در interface بصورت ضمنی Public است.ولی در کلاسها بصورت ضمنی Private است.پس حتما باید در کلاس Public تعریف گردد.

5- یک interface نمی تواند از یک کلاس ارث برد.چون در صورت ارث بری باید پیاده سازی را به ارث برد که این ممکن نیست.

نکته: چون کلاس فیلد دارد و interface فیلد ندارد.پس یک interface نمی تواند از یک کلاس ارث برد.

مثال:

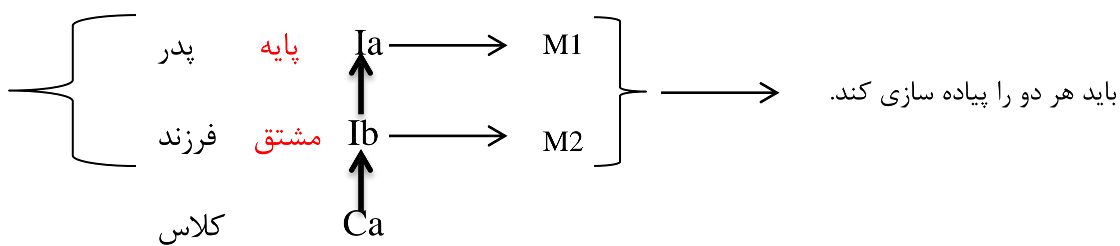
```
Using system;
Interface Iwrite
{
Void print();
```

```
Void M();
}
Class Demo:Iwrite
{
}
```

6- کلاسی که از یک Interface به ارث می برد باید تمامی متدهای Interface را پیاده سازی کند.(نوع برگشتی،اسم متد و پارامترهای ورودی باید کاملا منطبق با متدهای Interface باشد.)

7- یک کلاس فقط می تواند از یک کلاس دیگر به ارث برد.ولی یک کلاس می تواند از چندین Interface ارث بری کند.(پشتیبانی از ارث بری چندگانه)

8- یک Interface می تواند از یک Interface دیگر ارث برد.در اینصورت کلاس ما که از Interface فرزند ارث بری می کند ملزم است که نمامی متدها در Interface فرزند و پدر را پیاده سازی کند.

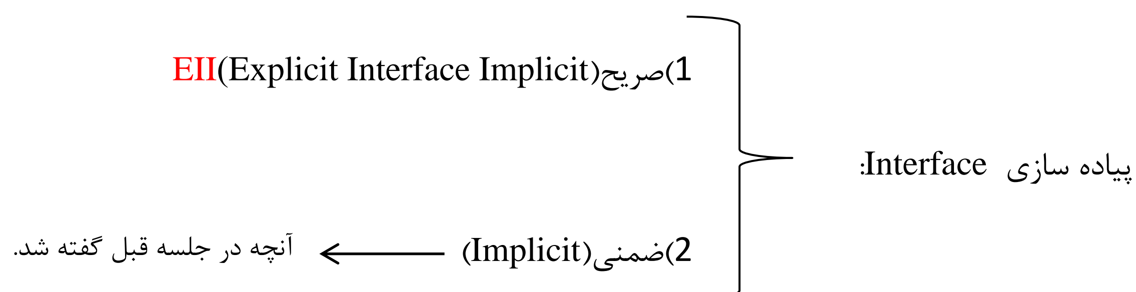


9- با استفاده از Interface کدها قابلیت بهتری در نگهداری،انعطاف و استفاده مجدد پیدا می کند.

مثال:

```
Interface iwrite
{
Void Print();
}
Class Demo:Iwrite
{
Public Static void Main()
```

```
{
Demo ref=new Demo();
Ref.Print();
Console.Readkey();
}
Public void Print()
{
Console.writeline("Interface");
}
}
```



در روش پیاده سازی صریح پیاده سازی قبل از تعریف اسم متد،اسم Interface مربوطه آورده می شود.در ضمن نحوه دستیابی متد مشخص نمی گردد.

مثال:

```
Interface Iwrite()
{
Void Print();
}
Class Demo:Iwrite
```

اسم Interface مربوطه آورده می شود.

```
{
Void Iwrite.Print()
{
Console.writeline("Test");
}
Public Static void Main()
{
Demo d=new Demo();
d.Print();
console.Readkey();
}
}
```

روش پیاده سازی EII

((Iwrite)d).Prnt(); تبدیل نوع انجام می گردد.

چون متد Print بصورت صریح پیاده سازی شده این نوع فراخوانی خطا می دهد.

نکته: اگر ما دو تا Interface داشته باشیم و در آنها متدهای همنام وجود داشته باشد.با استفاده از پیاده سازی صریح می توان از این متدها استفاده کرد.

نکته: برای فراخوانی متدی که به صورت صریح پیاده سازی شده است باید بصورت زیر عمل کنیم:

```
((Iwrite)d).Print();
```

نکته: مزایا و سودمندی متدهای EII:

1- اگر اسم های متد در interfaceهای مختلف با یکدیگر یکسان باشد با استفاده از پیاده EII این مشکل رفع می گردد.

مثال: اگر به مثال قبل تکه کدهای زیر را اضافه کنیم:

```
Interface Iprint()
{
Void print();
}
```

```
Class Demo:Iwrite,Iprint
{
Void Iwrite.print()  ⟶  EII روش پیاده سازی
{
Console.writeline("Test");
}
```

با استفاده از این روش برنامه دچار خطا نشده و ما از متد Print موجود در Interface،Iwrite استفاده می کنیم.

2- متدهای EII به شکل بهتری یک Interface را که چگونگی استفاده از یک Object را تعریف می کند ،را از یک کلاس که چگونگی ایجاد و پیاده سازی Object را تعریف می کند جداسازی کرد.

نکته:در interface فقط اسم متد را تعرف می کنیم.و در کلاس متدها را پیاده سازی می کنیم.و با استفاده از روش صریح این جداسازی واضح تر می شود.

3- متدهای EII نه کاملا Public هستند و نه کاملا Private ،به همین دلیل سطح دستیابی آنها را معین نمی کنیم.

4- آیا می توان در متدهایی که بصورت صریح تعریف گردیده از چندریختی استفاده کرد یا خیر؟

خیر،چون اگر کلمه کلیدی virtual را قبل از نام Interface قرار دهیم،خطا می دهد.

```
Void virtual Iwrite.print();  ⟶  خطا می دهد.
```

5- اگر کلاس مورد نظر ما از یک کلاس یا Interface ارث ببرد.کلاس ما به چه صورت باید تعریف شود؟

```
Class Demo:Iwrite,Base   ✗   ⟶  Base,Iwrite ✓
{
Void Iwrite.print()
{
```

```
Console.writeline("Test");
}
}
```

در نتیجه اگر کلاسی همزمان از یک Interface و کلاس ارث برد.ترتیب و اولویت در هنگام ارث بری مهم است.اول باید اسم کلاس بیاید و بعد اسم Interface بیاید.

6- متدهای EII نمی توانند virtual باشند بنابراین نمی توانند در کلاسهای مشتق override شوند.

کلاسهای Abstract(انتزاعی):

در کلاس پایه که شامل متدها و فیلدهایی است که در بین تمامی کلاسهایی که از آن مشتق می شود مشترک است.به عبارت دیگر کلاس پایه یک مفهوم کلی از یک شئ خاص ارائه نمی دهد.در نتیجه ایجاد یک شئ از این نوع کلاس بی فایده است(مانند Interfaceها).هر زمان که خواسته باشیم که توانایی ایجاد شئ از یک کلاس وجود نداشته باشد آنرا از نوع Abstract تعریف می کنیم.

در واقع کلاس Abstract بصورت قراردادی است که تمامی کلاسهای مشتق شده از آن ملزم به رعایت آن می باشد.

یک کلاس Abstract مانند یک Interface است ولی با دیدی وسیع چون برخلاف Interface می توان پیاده سازی متدها را داشته باشیم و اجازه بدهیم که کلاس های مشتق آنرا تغییر دهند.

مزایای کلاسهای Abstract:

1- در واقع یکی از مزیت های آن این است که فراهم نمودن یک کلاس پایه برای کلاسهایی که از آن مشتق شده که می توان متدها را در آن پیاده سازی کنیم یا خیر.
2- می توان سطح دسترسی را برای متدها و فیلدها تعیین کرد.

تفاوتهای یک Abstract class با یک Interface:

1- یک کلاس معمولی فقط می تواند از یک Abstract class ارث برد در حالیکه همان کلاس معمولی می تواند از چندین Interface ارث برد.
2- عناصر موجود در یک Abstract class مانند یک کلاس معمولی می توانند دارای سطح دسترسی باشند ولی یک Interface فاقد چنین امکانی است.

3- در یک Interface فقط می توان به معرفی متدها پرداخت در حالیکه در یک Abstract class علاوه بر تعریف امکان پیاده سازی نیز وجود دارد.
4- Abstract class یک کلاس معمولی می باشد.ولی یک Interface یک کلاس نمی باشد.
5- Abstract class مانند یک کلاس معمولی شامل فیلدها می باشد ولی Interface فاقد این امکان است.

نکته1: اگر در Abstract class قبل از تعریف متد کلمه کلیدی Abstract را قید کنیم.مانند متدهای Interface عمل کرده و پیاده سازی متدها به برنامه منتقل می شود.ولی در غیر اینصورت پیاده سازی در همان Abstract class انجام می گیرد.

نکته2: اگر در Abstract class متدی را اضافه کنیم.اگر این متد Abstract باشد بصورت خودکار در تمامی زیر کلاسها اعمال می شود .اما اگر در یک Interface متدی اضافه گردد باید در تمامی زیر کلاسهای مشتق شده از آن پیاده سازی را انجام دهیم.

مثال نکته 1و2:

Interface Iwrite()

{

A

C⟶ با اضافه کردن این متد باید پیاده سازی را در تمام زیر کلاسها بصورت دستی انجام دهیم.

}

Abstract class Demo

{

Abstract B ⟶ پیاده سازی به زیر کلاس محول می شود. ⟶ همانند Interface عمل می کند.

C⟶ پیاده سازی در اینجا می شود. ⟶ با اضافه کردن این متد پیاده سازی بصورت خودکار در تمام زیر کلاسها صورت می گیرد.

}

Test:Demo

{

}

```
Javad:Iwrite
{
}
```

نکته:از Abstract class نمی توان شئ ایجاد کرد.

مثال:

```
Public abstract class absclass
{
Protected int myvar;
Public abstract void print();
Public int AddtowNumber(int num1,int num2)
{
Return num1+num2;
}
}
Class Demo:abs class
{
Public override void print();
Console.writeline("Test");
{
Public Static void Main()
{
Demo d=new Demo();
d.print();
d.AddTowNumber(12,14);
```

```
}
}
```

نکته: هرگاه تمامی متدها در یک کلاس Abstract بصورت Abstract نوشته شوند(یعنی پیاده سازی آنها در کلاس مشتق واگذار گردد.)Abstract class رفتاری مشابه یک Interface خواهد داشت.

کلاس های Sealed:

بر خلاف کلاس های Abstract که پایه برای سایر کلاسها بودند و سایر کلاسها از آنها مشتق می شدند،کلاس Sealed نمی تواند به عنوان پایه برای کلاسهای دیگر قرار گیرد.(هیچ کلاسی نمی تواند از آن مشتق گردد.)برای تعریف کلاسهای Sealed از کلمه کلیدی Sealed استفاده می کنیم.چون هیچ کلاسی از کلاسهای sealed مشتق نمی شوند پس متدهای داخل این کلاس نمی توانند virtual تعریف گردند.

مثال:

```
Public Sealed class seaclass
{
Public int myvar;
Public sealed int AddTowNumber(int num1,intnum2)
{
Return num1+num2;
}
}
Class Demo
{
Public void print()
{
Console.writeline("Test");
}
```

```
Public Static void Main()
{
seaclass s=new seaclass();
s.AddTowNumber(12,14);
}
```

نکته:می توان کلمه کلیدی Sealed را تنها برای متدها بکار برد که در آنصورت آن متد قابلیت override شدن ندارد.

نکته:در Interface تنها به معرفی متدها می پرداختیم.متد بصورت virtual به مفهوم اولین پیاده سازی بود.متد بصورت override به مفهوم پیاده سازی بعدی متد است.متد بصورت Sealed به مفهوم آخرین پیاده سازی آن متد است.

کلاس های Static:

اگر کلاس بصورت Static تعریف گردد آنگاه بدون آنکه شیئ ای از آن کلاس ایجاد گردد می توان به اعضای آن کلاس دسترسی پیدا کرد.

نکته:لازم است فیلدها و متدها بصورت Static تعریف گردد.

مثال:

```
Public Static class staclass
{
Public static int myvar;
Public static int AddTowNumber(int num1,intnum2)
{
Return num1+num2;
}
}
```

```
Class Demo
{
Public void print()
{
Console.writeline("Teast");
}
Public Static void Main()
{
staclass s=new staclass();
staclass.myvar=12;
staclass.AddTowNumber(12,15);
}
}
```

خطا می دهد.

نکته:می توان یک متد یا یک فیلد را در یک کلاس معمولی بصورت Static تعریف کرد.آنگاه می توان تنها به آن متد یا فیلد بدون ایجاد شیئ دسترسی داشت.

نکته:در صورتیکه مقدار یک فیلد Static تغییر کند.مقدار آن در کل برنامه تغییر می کند.و کاربرد آن در انتقال اطلاعات از یک فرم به فرم دیگر است.

معرفی خاصیت در C# (Property):شبه فیلد

کلاس
فیلد ← صفات
متد ← رفتار

نکته:خاصیت ه مم می توانند مشابه فیلدها و متدها باشند و هم می توانند مشابه نباشند.یعنی چیزی بین این دو.

از خاصیت مشابه فیلدها استفاده می شود.در واقع خاصیت یک رابط بین یک فیلد منطقی و یک متد فیزیکی است.

یک خاصیت از دو متد set و get تشکیل می شود.

Set ⟶ به منظور دستیابی

Get ⟶ به منظور مقداردهی

مثال:

Public class Position

{

Public int x,y; ⟶ دو فیلد می توانن د بصورت همزمان تعریف شوند ولی دو خاصیت نمی توانند.

Public int X

{

Get{return x;} ⟶ متد دستیابی

Set{x=value;} ⟶ متد مقدار دهی

}

Public int Y ⟶ یک خاصیت

Get{return y;}

Set{y=value;}

}

}

همانند تعریف یک فیلد که نوع آنرا مشخص می کنیم نوع خاصیت نیز باید مشخص شود.

Public class program

{

Public Static void Main()

{

Positin p=new Position();

خطا می دهد چون فیلد x بصورت Private تعریف شده.

```
p.x=12;
p.X=12;
console.writeline(p.Y)
}
```

عملگر انتساب

فیلد x را با استفاده از خاصیت X مقداردهی می کند.

در این حالت متد Set مربوط به خاصیت X فراخوانی می شود.

با استفاده از خاصیت Y،مقدار در فیلد خصوصی y را چاپ می کند.

در این حالت متد get مربوط به خاصیت Y فراخوانی می شود.

X ⟶ X بزرگ

x ⟶ x کوچک

Y ⟶ Y بزرگ

y ⟶ y کوچک

هنگامی که از یک خاصیت در یک برنامه استفاده می کنیم در یکی از دو حالت زیر است:

الف)حالت خواندن:در این وضعیت بصورت خودکار کامپیالر کد مربوطه را به صورت فراخوانی متد get تبدیل می کند.

ب)حالت نوشتن:که می توان مقدار فیلد مربوطه را تغییر داد.کامپایلر بصورت خودکار متد set را فراخوانی می کند.

یک خاصیت را به سه روش می توان تعریف کرد:

1- فقط خواندنی:تنها متد get تعریف می گردد.که تنها قابلیت خواندن خاصیت را داریم.
2- فقط نوشتنی:تنها متد set تعریف می گردد.که برای نوشتن در خاصیت است و هرگونه تلاش برای خواندن با خطا مواجه می شود.
3- خواندنی/ نوشتنی:هم متد set و get تعریف می شود.

مثال:

هم فراخوانی و هم مقدار دهی را با هم داریم.

P.X+=10⟶

نکته:یک خاصیت در واقع کانالی برای دسترسی به فیلدهای Private ایجاد می کند.

نکته:خروجی برنامه قبل صفر(0)می باشد.چون از نوع int است و مقداردهی نگردیده است.(همانند مقدار پیش فرض).

P.X+=10⟶ P.X=P.X+10

محدودیت های خاصیت:

1- نمی توان چندین خاصیت را در یک دستور تعریف کرد(در حالیکه می توانستیم چندین فیلد را در یک دستور تعریف کرد.)

مثال:

Public int x,y;

Public int X,Y{get{…},set{…}} → خطا می دهد.

2- نمی توان خاصیت را بصورت const یا Read only تعریف کرد.

مثال:

Const int X {set{…},get{…}}; → خطا می دهد.

3- متدهای set و get را فقط می توان درون یک خاصیت تعریف کنیم.

اگر در متد set و get شامل دستورات مشابه(مشترکی)بودند نمی توان دستورات مشترک را از قبل از تعریف set و get نوشت.

مثال:

```
Public int X
{
Console.writeline("property");
Get{…};
Set{…};
}
```

خطا می دهد.

4- نمی توان خاصیتی را تعریف کرد که نوع آن void باشد.

مثال:

```
Public void X{set{…}get{…}};
```

5- نمی توان خاصیتی را بدون پارامتر تعریف کرد.

مثال:

```
Public       X{set{…}get{…}};
```

نوع باید تعیین شود.

6- نمی توان خاصیتی را با چندین پارامتر تعریف کرد.

مثال:

```
Public int,string X{set{…}get{…}};
```

دو نوع یا بیشتر نمی شود.

7- نمی توان یک خاصیت را از طریق متد set مقداردهی اولیه کرد.در حالیکه برای یک فیلد چنین امکاناتی وجود دارد.

مثال:

```
Int x=10;
```

تعریف خاصیت در interface:

```
Interface Ilocation
{
Int X{set;get};
Int Y{set;get};
}
```

می توان برای interface خاصیت تعریف کرد.

در این حالت تنها کلمات کلیدی setوget را نوشته و به جای بدنه آنها از (;) استفاده می کنیم.

مثال:

```
Public class position:Ilocation
{
Private int x,y;
Public int X
{
Get{return x;};
Set{x=value;};
}
Public int Y
{
Get{return y;}
Set{y=value;}
}
```

```
}
Public class program
{
Public Static void Main()
{
Position p=new Position();
p.x=12; ⟶     خطا می دهد.
p.X=12;
console.writeline(p.Y);
}
}
```

نکته:در کلاسی که از interface ارث می برد می توان خاصیت را بصورت virtual (مجازی)تعریف کرد که سایر کلاس های مشتق شده از این کلاس اجازه override داشته باشند.

نکته:همچنین می توان با استفاده از پیاده سازی صریح یک خاصیت آنرا بصورت غیر public و غیر virtual تعریف کرد.

مثال:

```
Int Ilocation.X
{
Get{return x;};
Set{x=value;};
}
Int Ilocation.Y
{
Get{return y;};
```

```
Set{y=value;};
}
}
```

مثال:

1- پیاده سازی برنامه با استفاده از property (مثال در کلاس).
2- با در نظر گرفتن محیط عملیاتی دانشگاه،کلاس های موجود در آنرا نوشته همراه با سلسله مراتب کلاس ها(وراثت،پلی مورفیسم،کلاس abstract،کلاس Sealed،کلاس Satic،interface پیاده سازی کنید.)

کلیه کلاس های موجود (فیلدها و متدها)را در این محیط عملیاتی شناسایی کنید و نسبت آنها را با هم مشخص کنید.پس از آن،نوع هر کلاس و همچنین interface در صورت نیاز و پلی مورفیسم برای پیاده سازی بصورت مختلف برای متدها ارائه دهید.(تمامی این تحلیل ها بصورت تایپ شده تحویل گردد.)

قسمت نهایی:

```
Class program
{
Stime t1=new stim(12,14);
Stime t2;
T2=t1;
}
}
```

نکته: می توان در interface خاصیت ها را به صورت فقط خواندني یا فقط نوشتني هم تعریف کرد حتی می توان توابع set و get را در بین interface های مختلف توزیع کرد.

خاصیت های static:

مثال:

```
Using system;
Public Class position
{
Public static int i ;        // بدون ایجاد شی می توان به آن دسترسی پیدا کرد
Public int  j;               // بدون ایجاد شی نمی توان به آن دسترسی پیدا کرد
Public int Add()             // فیلد نمونه ای
{
Return  i+j;
}
}
Public Class program
{
Public static void main()
{
Position  p1=new   position();
P1.j=10;
Position.i=10;               // فقط با نام کلاس فراخوانی می گردد.یعنی مربوط به یک کلاس است
Position  p2=new  position ();
P2.j=20;                     // برای یک شی است .با شی می توان به آن دسترسی داشت.
Console . writeline(p1.Add());
Console. Writeline(p2.Add());
```

در یک کلاس می توان دو فیلد داشت:

1) فیلدهای نمونه ای (فیلدی برای هر شی)

2) فیلد static (فیلدی برای یک نوع)

نکته: اگر مقدار فیلد static تغییر کند برای تمام شی ها تغییر می کند و اعمال می شود .

نکته:فیلد static مانند سود بانکی می باشد و فیلد معمولی مانند نام و نام خانوادگی و... می باشد

.

نکته: برای فیلدهای static باید خاصیت static نوشت.

مثال:

```
Using system;
Public Class position
{
Private  static int x ;      // فیلد از نوع مقداری یا شمارشی
                             // فیلد از نوع static
Public  static  int  x;      // خاصیت static
Public int Add()
```

```
{
Get{return x;}
Set{x=value;}
}
}
Public Class program
{
Public static void main()
{
Position.x=10;
Console . writeline(position.x);
Position  p=new  position ();
}
}
```

از نوع ارجاعی

نکته: زمانی که یک شی را از یک کلاس ایجاد کنیم آن شی از نوع ارجاعی است ولی زمانی که از نوع int و... تعریف کرده و نوع مقداری و شمارشی است.

نوع های داده ای در #C:

1-نوع داده ای مقداری (value type):

مانند در این نوع داده ای برای ایجاد یک نمونه نیازی به استفاده از کلمه کلیدی new نمی باشد

2-نوع داده ای ارجاعی (Refrence type):

در این حالت با استفاده از کلمه کلیدی new باید یک نمونه ایجاد کرد و سپس از آن استفاده نمود .

مثال:

```
Using system ;
Public class program
{
Public static  void main()
{
Int  i, j;
i=10;
j=i;
position  p1=new   position ();
position  p2=new   position ();
p1.x=10;
p2=p1;
```

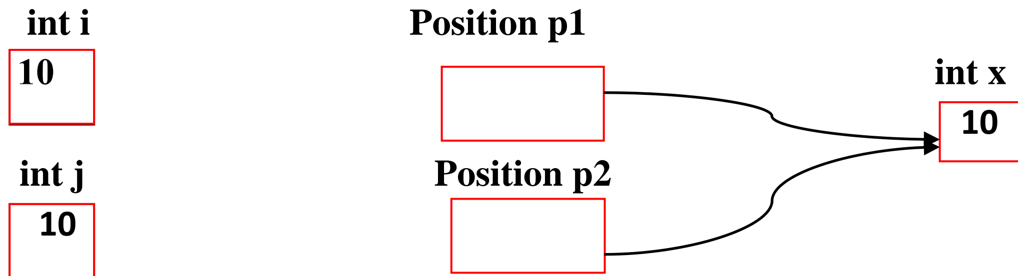
از نوع مقداری (Int i, j)

از نوع ارجاعی (p1, p2)

```
}
}
```

نکته: کلاس **position** برنامه فوق همانند برنامه قبل پیاده سازی می گردد.

تفاوت بین داده ای مقداری و نوع داده ای ارجاعی :



نکته:کلاس نوع ارجاعی است ولی در نوع داده ای مقدار در همان حافظه قرار می گیرد.

آشنایی با ساختار (struct):

ساختار نوع مقداری است.و بسیار شبیه کلاسها می باشد.می توانند برای خود شامل فیلد،متد و سازنده باشند.

نحوه تعریف یک struct:

مثال:

کلمه کلیدی

```
Struct   Time
{
Private    int  hours ,minutes,seconds;
Public  Time(int h, int m, int s)
{
Hours=h;
minutes=m;
seconds-s;
}
}
```

متد سازنده هم نام با کلاس است

```
Using system;
Public  class  program
{
Public  static   void main()
{
Time  T1;
Time  T2=new  Time(17,25,30);
}
}
```

انواع داده در #C:

Bool →	true/false
Byte →	داده 8 بیتی بدون علامت
Short →	16 بیتی عدد صحیح
int →	32 بیتی عدد صحیح
unit →	32 بیتی بدون علامت
long →	64 بیتی عدد صحیح
ulong →	64 بیتی بدون علامت
float →	32 بیتی اعشاری
double →	64 بیتی اعشاری
char →	کاراکتر 16 بیتی
string →	رشته 16 بیتی

سه دستور یکسان برای تعریف متغیر از نوع int:

1- int i=7;
2- System.int 32 i=new system.int 32 (7);
3- System.int 32 i=7;
4- Int max =int.max value;

بزرگترین مقدار Intرا شامل می شود

عملگرها در C#:

نحوه تعریف کاراکتر و مقداردهی آن :

Char C='B'; → با استفاده از تک کوتیشن

% → باقی مانده

مثال:

Int x=60 ;

Int y= x%29; → خروجی 2 می باشد

+ → جمع

– → تفریق

* → ضرب

/ → تقسیم

++ → افزایشی

-- → کاهشی

مثال:

Int x=5;

Console.writeline(x++); → پسوندی

```
Console.writeline(++x);
```
→ پیشوندی

عملگرهای شرطی و رابطه ای:

== → مساوی

!= → غیر مساوی

> → کوچکتر

=> → کوچکتر مساوی

< → بزرگتر

=< → بزرگتر مساوی

مکمل: علامت مکمل بیتی → ~

مثال:

```
Int  x=127;
~X=  -128;
```

عملگرهای منطقی:

معرفی ساختار کنترلی:

1-دستور if :

```
If  ( عبارت بولی )
{ جملات اول }
Else
{ جملات دوم }
```

2-while:

While(عبارت بولی)

{

دستورات

}

مثال:

```
Int  i=0;
While(I != 10)
{
Console.writeline( i );
i++;
}
```

3-دستور for:

For(مقداردهی اولیه; عبارت بولی ; بروز رسانی متغیر)

{

دستورات

}

مثال:

```
For (int  i=0; i<10 ;i++)
{
Console.writeline( i );
}
```

تمرین:

1-برنامه ای نوشته که فاکتوریل عدد ورودی را حساب کند .

2-برنامه ای نوشته که حاصل عبارت زیر را محاسبه کند (ورودی x را از کاربر دریافت کند.)

$$E^x=1+x/11 + x^2/21 + \ldots\ldots\ldots\ldots$$

آرایه در #C:

نحوه تعریف آرایه:

`Int [ ] x;` ⟶ آرایه ای از نوع intبه نام x بدون اندازه در نظر گرفته می شود

تعیین اندازه آرایه:

`Int [ ] x=new int [ 10 ];` ⟶ آرایه ای با نامx از نوع int با طول 10 در نظر گرفته می شود

`X[ 0 ]=1;` ⟶ مقدار دهی به خانه های آرایه

آرایه ای از رشته ها:

`String [] str=new string [2]{"abc","1234"}` ⟶ مقداردهی در حین تعریف آرایه

`Str[0]={"abc","1234"};` ⟶ مقداردهی آرایه

آرایه های چند بعدی:

`Int[ , ] x=new int [4,7];`

نکته: در زبان C حتما باید اندازه آرایه در حین تعریف آرایه مشخص گردد ولی در #C می توان اندازه آرایه را در حین برنامه مشخص کرد.

نکته: سازنده پیش فرض یعنی سازنده بدون ورودی

مثال:

```
Class CTime
{
Public int hours,minutes,seconds;
Public CTime ()
{
Console.writeline("Test");
}
}
Struct  STime
{
Public  int  hours,minutes,seconds;
Public  STime()
{
Hours=12;
Minutes=12;
}
Class  program
{
Static   void  main ()
{
```

نوع متغیرها intاست

سازنده پیش فرض

خطا می دهد

چون تمام فیلد ها باید مقداردهی گردد

```
CTime  T1;
STime  T2;
T1.hours=12;   ---->  خطا می دهد        CTime   T1 = new  CTime();
T2.hours=12;   ---->  نحوه دیگر تعریف    STime  T2 = new  STime();
}
}
```

نکته: با تعریف T2 به نحوه ی زیر یعنی ;T2.hours =12 اگر دستور چاپ بدهیم با خطا مواجه می شویم چون هیچ مقداری ندارد ولی با تعریف به صورت کامل با خطا مواجه نمی شویم.

تفاوت های بین ساختار و کلاس :

1-نمی توان برای ساختار سازنده ی پیش فرض تعریف کرد.(سازنده ی پیش فرض : سازنده ی بدون پارامتر)

به این دلیل که کامپایلر به صورت خودکار یک سازنده ی پیش فرض برای ساختار در نظر می گیرد ،در کلاس کامپایلر زمانی برای یک کلاس سازنده ی پیش فرض در نظر می گیرد که هیچ سازنده ای تعریف نشده باشد .اما در یک struct اگر در یک یا چند سازنده تعریف شده باشد باز هم برای آن سازنده پیش فرض در نظر گرفته می شود .

مثال:

```
Class  CTime
{
Public  int  hours,minutes,seconds;
Public CTime (int h,int m)  ---->  در کلاس می توان تمام فیلدها را مقدار دهی نکرد
{
hours=h;
minutes=m;  ---->  فیلد second مقدار دهی نگردیده است
```

```
}
}
Class  program
{
CTime    T1 = new   CTime(12,14);
Console.writeline(T1.seconds);
STime   T2 = new    STime(12,14);
T2.seconds=23;
}
Struct  STime
{
Public  int  hours,minutes,seconds;
Public  STime(int  h,int  m)
{
Hours=h;
Minutes=m;
Seconds=0;
}
}
```

خروجی صفر است چون از نوع عدد صحیح است

تمام فیلد ها باید مقداردهی گردد

جهت رفع خطا

2-در سازنده یک کلاس اگر فیلدی مقدار دهی اولیه نشود کامپایلر بر اساس نوع آن فیلد مقدار دهی اولیه می کند .ولی در سازنده ساختار باید صریحا تمامی فیلدها مقداردهی اولیه شوند .در غیر اینصورت با خطای زمان کامپایل مواجه می شویم.

مثال:

```
Class CTime
{
```

```
Public  int  hours=0;
Public  int  minutes,seconds;
Public CTime (int h,int m)
{
hours=h;
minutes=m;
}
Struct  STime
{
Public  int  hours=0;   →   خطا می دهد
Public  int  minutes,seconds;
}
```

3-کلاس می تواند از کلاس دیگری ارث برد ولی ساختارها چنین قابلیتی ندارند .یعنی Struct به صورت ضمنی Sealed هستند ونمی توان از آنها به ارث برد .

4-در کلاس ها می توان فیلدها را در زمان تعریف مقداردهی اولیه کرد ولی در ساختارها امکان چنین کاری وجود ندارد .(مانند تکه کد فوق(مثال قبل))

مثال:

```
Class CTime
{
Public  int  hours;
Public  int  minutes,seconds;
Public CTime (int h,int m)
{
Hours=h;
```

```
Minutes=m;
}
Struct  STime
{
Public  int  hours;
Public int  minutes , seconds;
}
```

چون مقداردهی نگردیده

```
Class  program
{
CTime    T1 = new   CTime();
Console.writeline(T1.seconds);
```

خروجی 0 است

```
STime  T2;
Console.writeline(T2.seconds);
```

خطا می دهد

```
}
```

5- هنگامی که یک متغیر از نوع struct تعریف می کنید فیلدهای داخل آن به صورت پیش فرض مقداردهی نمی شوند و مقدار آنها تا زمانی که به صورت صریحا معین نشود قابل خواندن نیست .حال اگر یک فیلد از ساختارمقداردهی گردید تنها می توان آن فیلد را خواند و سایر فیلدها را باید جداگانه مقداردهی کرد.

ساده ترین راه برای مقداردهی اولیه فیلدهای یک ساختار آن است که ساختار را با سازنده پیش فرض آن فراخوانی کنیم .وچون ساختار همیشه شامل سازنده پیش فرض هستند فیلدها را بر اساس نوع آنها مقدار دهی می کند .

تعریف T2 به ســــــه روش:

مثال: **نحوه ← (1)**

```
Class  program
{
CTime   T1 = new   CTime();
```

```
Console.writeline(T1.seconds);  →  0
STime T2;
Console.writeline(T2.seconds);  →  خطا
}
```

نحوه → (2)

```
Class  program
{
CTime   T1 = new   CTime();
Console.writeline(T1.seconds);
STime T2;
T2=12;  →          تا زمانی که مقدار دهی نگردد خطا می دهد
Console.writeline(T2.seconds);  →     12
}
```

نحوه→(3)

```
Class  program
{
CTime   T1 = new   CTime();
Console.writeline(T1.seconds);  →  0
STime T2 = new  STime();  →  تعریف به این روش مانند T1عمل می کند
Console.writeline(T2.seconds);  →  0
}
```

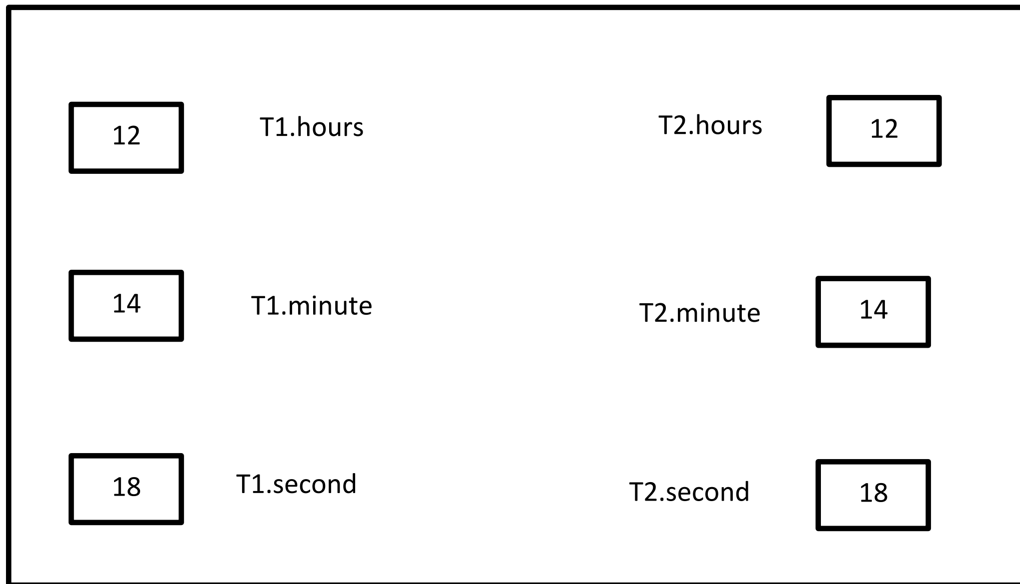
نکته: در روش فوق متغیر second را بر اساس int گرفته و خروجی آن را بر همین اساس چاپ می کند.

مثال:

```
Class CTime
```

```
{
Public  int  hours;
Public  int  minutes,seconds;
Public CTime (int h,int m)
{
Hours=h;
Minutes=m;
}
}
Struct  STime
{
Public  int  hours;
Public int  minutes , seconds;
Public  STime (int h , int m ,int s)
{
Hours=h;
Minutes=m;
Seconds=s;
}
}
```

حافظه:



نکته: تنها زمانی می توان یک متغیر از نوع struct را با متغیر دیگری از struct مقداردهی کنیم.که متغیر سمت راست مقداردهی اولیه شده باشد.

```
Int i=7
System.Int32 i=7;
System.Int32 i= new system.Int32(7)
```

یک ساختار است.

نوع int و ...... در حقیقت نوعی struct است.

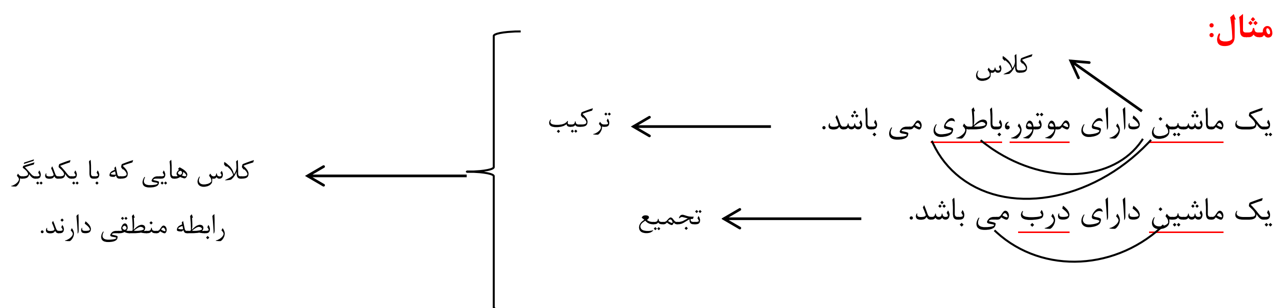
مثال: در مورد int نام دیگری برای ساختار Int32 در (namespace)فضای نام system است.

نکته: نوع string یک کلاس می باشد.

کلاس های تو در تو:(Nested class)

کلاس داخلی،کلاسی است که در داخل کلاس بزرگتر نوشته می شود.

مثال:



مزایای کلاس های تو در تو:

1- گروه بندی منطقی کلاس ها:

گاهی ممکن است یک کلاسی داشته باشیم که تنها توسط یک کلاس دیگر مورد استفاده قرار گیرد.پس بصورت منطقی این دو کلاس در یک گروه قرار می گیرند.پس بهتر است کلاس را در داخل کلاس استفاده کننده تعریف کنیم.

2- افزایش کیفیت(رعایت بهتر Encapsulation):

گاهی یک کلاس نیاز دارد به فیلد های خصوصی در کلاس دیگر دسترسی پیدا کند.

مثال:فرض کنیم کلاس B نیاز دارد به فیلد های خصوصی کلاس A دسترسی پیدا کند.می توان کلاس B را در داخل کلاس A تعریف کرده،آنگاه کلاس B به تمامی فیلد های کلاس A دسترسی داشته در حالیکه فیلد های کلاس A همچنان بصورت private هستند.

3- خوانایی بهتر و نگهداری آسانتر کدها:

استفاده از کلاس های داخلی،کدها را به محل استفاده نزدیکتر می کند.

مثال:

```
Public class A
{
Public int v1;
Public class B
{
Public int v2;
}
}
Public class program
{
Public Static void Main(){
A a=new A();
a.v1=100;
A.B b=new A.B();
b.v2=200;
}
}
```

کلمه کلیدی (class)

کلاس داخلی می باشد. (Public class B { Public int v2; })

شئ از کلاس A (a)

شئ از کلاس B (b)

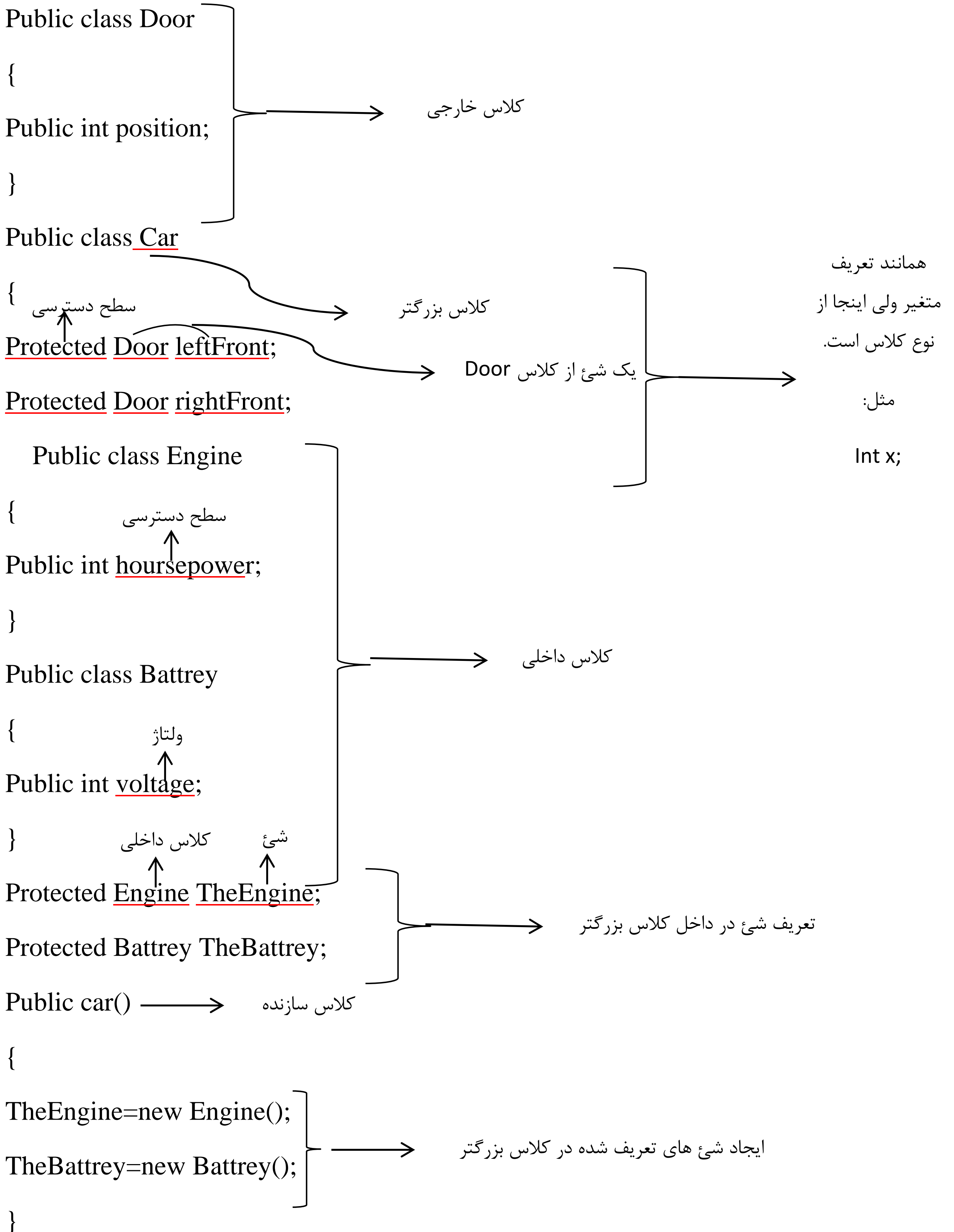
اگر B b=new B(); ✖ خطا می دهد.

نکته:یک کلاس داخلی مثل یک کلاس معمولی تعریف می گردد.

نکته:اشیایی هستند که می توانند بصورت مستقل وجود داشته باشد.و رابطه ی آن باشئ بزرگتر رابطه تجمیع(Aggregation) است.

نکته:اشیایی هستند که نمی توانند بصورت مستقل باشند و وجود آنها منوط به شئ بزرگتر است.رابطه ی این اشیاء با شئ بزرگتر رابطه ترکیب(Composition) است.

مثال:



```
}
Public class fordcar:Car
{
Public fordCar()
TheEngine.hoursepower=100;
}
}
```

پایان