فصل 15

رابط ها

* رابط چیست؟
* تعریف رابط
* رابط یک نوع ارجاعی است
* استفاده عملگر as با رابط ها
* پیاده سازی رابط های چند گانه
* پیاده سازی رابط ها با عضو های تکراری
* مرجع برای رابط های چند گانه
* عضو به ارث رسیده به عنوان پیاده سازی
* پیاده سازی صریح عضو های رابط
* ارث بری واسط ها از واسط ها
* مثالی از پیاده سازی کلاس ها مختلف

رابط چیست؟

رابط[[1]](#footnote-1) یک نوع ارجاعی است که مجموعه ای از توابع عضو را مشخص می کند، اما آنها را پیاده سازی نمی کند. پیاده سازی رابط به کلاس ها و ساختار ها واگذار شده است. از این توضیحات به نظر می رسد که ساختاری کاملاً انتزاعی است، پس ابتدا اجازه دهید به شما مشکلی که یک رابط به حل آن کمک می کند را نشان دهیم.

به عنوان مثال، به کد زیر نگاه کنید. اگر شما به متد Main در کلاس Program نگاه کنید، خواهید دید که آن شی ای از کلاس CA را ایجاد و مقداردهی کرده است و شی را برای متد PrintInfo ارسال می کند. متد PrintInfo انتظار شی ای از نوع CA را دارد که اطلاعات موجود در شی کلاس را چاپ کند.

class CA

{

public string Name;

public int Age;

}

class CB

{

public string First;

public string Last;

public double PersonsAge;

}

class Program

{

static void PrintInfo( CA item ) {

Console.WriteLine( "Name: {0}, Age {1}", item.Name, item.Age );

}

static void Main() {

CA a = new CA() { Name = "John Doe", Age = 35 };

PrintInfo( a );

}

}

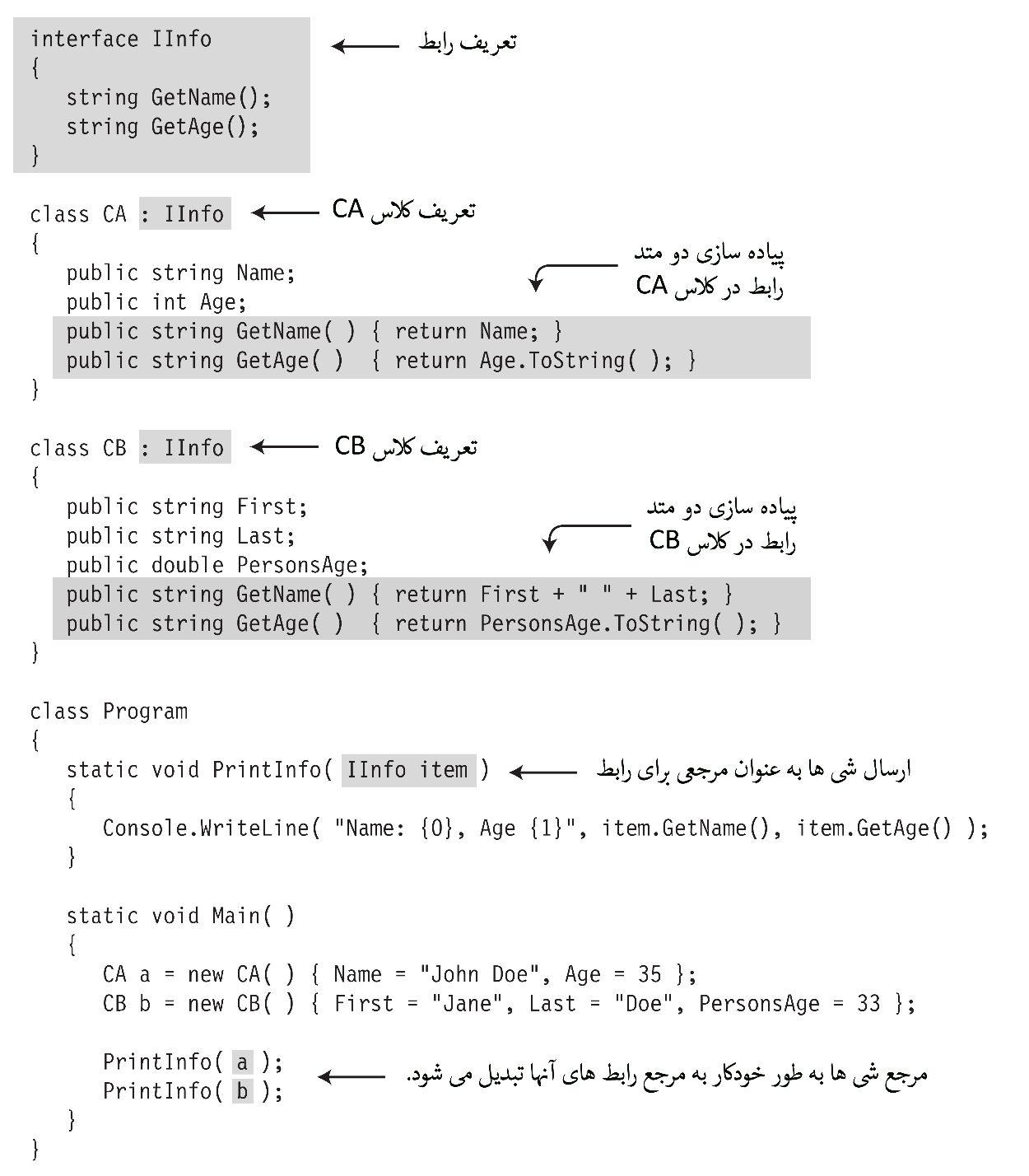
متد PrintInfo تا زمانی که شی از نوع CA را به آن ارسال کنید کار می کند، اما اگر شی از نوع CB (که در کد بالا نشان داده شده است) ارسال شود آن کار نخواهد کرد. با این حال، فرض کنید که الگوریتم متد PrintInfo بسیار مفید بوده است که می خواهید توانایی در خواست بسیاری از شی های کلاس های مختلف را برای آن داشته باشد.

چندین دلیل وجود دارد که این کد موجود کار نخواهد کرد. اول از همه، پارامتر های رسمی متد PrintInfo مشخص می کند که پارامتر های واقعی باید شی ای از نوع CA باشد، بنابراین ارسال شی ای از نوع CB یا هر شی ای دیگری با خطای کامپایلر رو به رو می شود. اما حتی اگر ما بتوانیم این مشکل را دور بزنیم و شی ای از نوع CB را ارسال کنیم، ما هنوز هم یک مشکل داریم، زیرا ساختار CB با ساختار CA متفاوت است. نام و نوع فیلد های آن نسبت به CA تفاوت دارد، و متد PrintInfo هیچ چیزی در مورد این فیلد ها نمی داند.

آیا می توانیم کلاس ها را به گونه ای ایجاد کنیم که آنها بتوانند با موفقیت به متد PrintInfo ارسال شوند و متد PrintInfo بتواند توانایی پردازش آنها را صرف نظر از ساختار کلاس داشته باشد؟ با رابط ها این کار امکان پذیر است.

کد موجود در شکل 1-15 مشکل را با یک رابط حل کرده است. نیازی به درک جزئیات نیست، اما به طور کلی به شرح زیر است:

* ابتدا، آن یک رابط با نام IInfo تعریف کرده است که شامل دو متد GetName و GetAge است که هر کدام از آنها یک رشته را برمی گرداند.
* کلاس CA و CB هر کدام رابط IInfo را در لیست کلاس پایه خود اضافه می کند و سپس دو متد مورد نیاز رابط را پیاده سازی می کند.
* سپس متد Main نمونه ای کلاس های CA و CB ایجاد می کند و آنها را به متد PrintInfo ارسال می کند.
* ار آنجایی که نمونه های کلاس با رابط پیاده سازی شده اند، متد PrintInfo می تواند متد ها را فراخوانی کند، و هر نمونه از کلاس متد خودش را همانطور که در کلاس خود تعریف شده است اجرا می کند.



شکل 1-15 با استفاده از رابط متد PrintInfo ایجاد شده است تا برای هر تعداد کلاس قابل استفاده باشد

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Name: John Doe, Age 35  Name: Jane Doe, Age 33 |

مثالی برای استفاده از رابط IComparable

حال که بعضی از مشکلات حل شده توسط رابط ها را دیده ایم، به نمونه دوم نگاه کنید و وارد جزئیات بیشتری شوید. با کد زیر شروع می کنیم که باعث می شود یک آرایه نامرتب از اعداد صحیح را به صورت صعودی مرتب کند. مواردی که کد انجام می دهد به شرح زیر است:

* خط اول آرایه ای از پنج عدد صحیح را بدون هیچ ترتیبی خاصی ایجاد می کند.
* خط دوم با استفاده از متد استاتیک Sort در کلاس Array عناصر آرایه را مرتب می کند.
* حلقه foreach آنها را چاپ می کند، نشان می دهد که اعداد صحیح در حال حاضر به صورت صعودی مرتب شده اند.

var myInt = new [] { 20, 4, 16, 9, 2 }; // ایجاد آرایه

Array.Sort(myInt); // مرتب کردن عناصر

foreach (var i in myInt) // چاپ کردن آنها

Console.Write("{0} ", i);

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| 2 4 9 16 20 |

روشن است که متد Sort کلاس Array برای آرایه های نوع int به درستی کار می کند، اما همانطور که در زیر نشان داده شده است، اگر از کلاس های خود استفاده کنید چه اتفاقی خواهد افتاد؟

class MyClass // تعریف کلاس

{

public int TheValue;

}

...

MyClass[] mc = new MyClass[5]; // ایجاد آرایه ای از پنج عنصر

... // ایجاد و مقدار دهی عناصر

Array.Sort(mc); // سعی برای مرتب کردن – خطا تولید می کند.

هنگامی که شما سعی می کنید این کد را اجرا کنید، با جای مرتب سازی عناصر باعث تولید یک استثنا می شود. متد Sort کلاس Array برای اشیا کلاس MyClass به این دلیل کار نمی کند، که نمی دادند چگونه باید اشیا کلاس تعریف شد توسط کاربر را با یک دیگر مقایسه کند و چگونه آنها را مرتب کند. متد Sort کلاس Array به رابط IComparable بستگی دارد، که در BCL تعریف شده است. IComparable متدی با نام CompareTo دارد.

کد زیر تعریف از رابط IComparable را نشان می دهد. توجه کنید که بدنه رابط شامل اعلان متد CompareTo می باشد که مشخص می کند پارامتری از نوع object دریافت می کند. اگر چه متد دارای نام، پارامتر و نوع بازگشتی است اما هیچ پیاده سازی وجود ندارد. در عوض قسمت پیاده سازی با نقطه ویرگول نشان داده شده است.

کلمه کلیدی نام رابط

↓ ↓

public interface IComparable

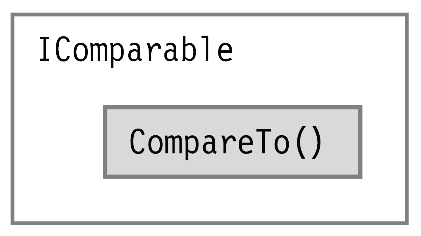
{

int CompareTo( object obj );

} ↑

نقطه ویرگول بجای پیاده سازی بدنه متد

شکل 2-15 رابط IComparable را نشان می دهد. متد CompareTo با رنگ خاکستری نشان داده شده است.



شکل 2-15 نمایش رابط IComparable

اگر چه در تعریف رابط برای متد CompareTo پیاده سازی انجام نمی شود، اما مستندات .NET از رابط IComparable توضیح می دهد که متد باید چه کاری انجام دهد، در صورتی که شما کلاس یا ساختاری را ایجاد می کنید که این رابط را پیاده سازی می کند. این گزارش می گوید که در زمان فراخوانی متد CompareTo باید یکی از سه مقادیر زیر برگردانده شود:

* یک مقدار منفی، اگر شی فعلی کمتر از شی پارامتر باشد.
* یک مقدار مثبت، اگر شی فعلی بیشتر از شی پارامتر باشد.
* صفر، اگر هر دو شی برابر باشند.

الگوریتم استفاده شده توسط Sort به این بستگی دارد که آیا می تواند عناصرش را با متد CompareTo مقایسه کند. نوع int برای رابط IComprable پیاده سازی شده است، اما برای کلاس MyClass انجام نشده است، بنابراین وقتی مرتب سازی تلاش می کند که از متد CompareTo که در کلاس MyClass وجود ندارد استفاده کند یک خطا تولید می کند.

با پیاده سازی کلاس برای رابط IComparable می توانید متد Sort را برای اشیا کلاس MyClass ایجاد کنید. برای پیاده سازی یک رابط برای کلاس یا ساختار باید دو مورد انجام دهید:

* باید نام رابط در لیست کلاس پایه آن لیست شود.
* باید برای هر یک از عضو های رابط در کلاس پیاده سازی انجام شود.

به عنوان مثال، کد زیر کلاس MyClass را برای رابط IComparable به روز رسانی می کند. به موارد زیر توجه کنید:

* نام رابط در لیست کلاس پایه ذکر شده است.
* کلاس متد CompareTo را پیاده سازی می کند، که نوع پارامتر و نوع بازگشتی متد با متد رابط مطابقت می کند.
* متد CompreTo مطابق با اسناد ارائه شده برای رابط پیاده سازی شده است، به این معنا که با مقایسه شی موجود و شی ارسال شده مقدار منفی 1، مثبت 1 یا 0 را بر می گرداند.

نام رابط در لیست کلاس پایه

↓

class MyClass : IComparable

{

public int TheValue;

public int CompareTo(object obj) // پیاده سازی متد رابط

{

MyClass mc = (MyClass)obj;

if (this.TheValue < mc.TheValue) return -1;

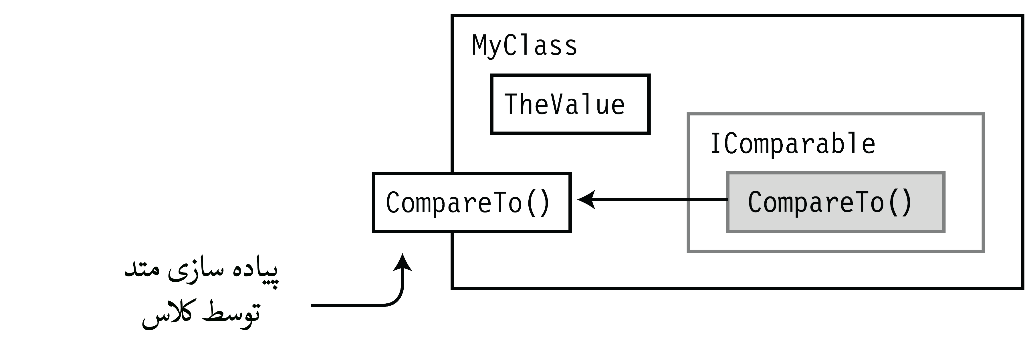
if (this.TheValue > mc.TheValue) return 1;

return 0;

}

}

شکل 3-15 کلاس به روز شده را به تصویر کشیده است. فلش از متد سایه دار رابط به متد کلاس نشان می دهد که متد رابط دارای کد بدنه نیست، اما در کلاس پیاده سازی شده است.



شکل 3-15 پیاده سازی رابط IComparable در کلاس MyClass

حال که رابط IComprable با کلاس MyClass پیاده سازی شده است، متد Sort به خوبی کار خواهد کرد. به هر حال، تنها پیاده سازی متد CompareTo کافی نیست ــ آن باید قسمت پیاده سازی رابط را داشته باشد به این معنی که باید نام رابط در لیست کلاس پایه وجود داشته باشد.

در زیر کد به روز رسانی شده کامل را نشان می دهد، که می توان از متد Sort کلاس Array برای اشیا کلاس MyClass استفاده کرد. متد Main ارایه ای از کلاس MyClass ایجاد می کند و عناصر آن را مقداردهی و چاپ می کند. سپس متد Sort فراخوانی می شود و برای اینکه نشان دهد آنها مرتب شده اند دوباره آنها را چاپ می کند.

class MyClass : IComparable // کلاس پیاده سازی رابط

{

public int TheValue;

public int CompareTo(object obj) // پیاده سازی متد

{

MyClass mc = (MyClass)obj;

if (this.TheValue < mc.TheValue) return -1;

if (this.TheValue > mc.TheValue) return 1;

return 0;

}

}

class Program

{

static void PrintOut(string s, MyClass[] mc)

{

Console.Write(s);

foreach (var m in mc)

Console.Write("{0} ", m.TheValue);

Console.WriteLine("");

}

static void Main()

{

var myInt = new [] { 20, 4, 16, 9, 2 };

MyClass[] mcArr = new MyClass[5]; // MyClass ایجاد ارایه ای از کلاس

for (int i = 0; i < 5; i++) // مقداردهی آرایه

{

mcArr[i] = new MyClass();

mcArr[i].TheValue = myInt[i];

}

PrintOut("Initial Order: ", mcArr); // چاپ کردن ارایه اولیه

Array.Sort(mcArr); // Sort the array.

PrintOut("Sorted Order: ", mcArr); // چاپ کردن آرایه مرتب شده

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Initial Order: 20 4 16 9 2  Sorted Order: 2 4 9 16 20 |

تعریف رابط

در بخش قبل از رابطی استفاده شد که از قبلا در BCL تعریف شده است. در این بخش، خواهید دید که یک رابط چگونه تعریف می شود. مواردی که مهم درباره تعریف یک رابط بدانید به شرح زیر است:

* تعریف یک رابط نمی تواند شامل موارد زیر باشد:
  + داده ها عضو
  + عضو های استاتیک
* تعریف یک رابط تنها می تواند شامل اعلان توابع عضو غیر استاتیک زیر باشد:
  + متد ها
  + خصوصیت ها
  + رویداد ها
  + ایندکسر ها
* اعلان توابع عضو نمی تواند شامل قسمت کد پیاده سازی باشد. در عوض بجای بدنه توابع عضو از نقطه ویرگول استفاده می شود.
* طبق قرار داد، نام رابط ها با حروف بزرگ I ( به عنوان مثال، ISaveable) آغاز خواهند شد.
* همانند کلاس ها و ساختار ها، همانطور که در کلاس های جزئی و نوع های جزئی که در بخش 6 شرح داده شده است، اعلان رابط ها می توانند به صورت اعلان رابط جزئی تقسیم شوند.

کد زیر مثالی از تعریف رابط با دو تابع عضو را نشان می دهد:

کلمه کلیدی نام رابط

↓ ↓

interface IMyInterface1 نقطه ویرگول بجای بدنه متد

{ ↓

int DoStuff ( int nVar1, long lVar2 );

double DoOtherStuff( string s, long x );

} ↑

نقطه ویرگول بجای بدنه متد

یک تفاوت مهم بین دسترسی به رابط و دسترسی به عضو های رابط وجود دارد:

* تعریف رابط می تواند شامل یکی از اصلاحگر های دسترسی public، protected، internal، یا private باشد.
* عضو های رابط به طور ضمنی public هستند، و هیچ اصلاحگر دسترسی مجاز نیست که استفاده شود ( حتی public).

اصلاحگر های دسترسی مجاز هست در رابط ها استفاده شوند

↓

public interface IMyInterface2

{

private int Method1( int nVar1, long lVar2 ); // خطا

} ↑

اصلاحگر های دسترسی مجاز نیستند در عضو های رابط استفاده شوند.

پیاده سازی واسط

تنها کلاس ها و ساختار ها می توانند یک رابط را پیاده سازی کنند. همانطورکه در مثال Sort نشان داده شد، برای پیاده سازی یک رابط، کلاس یا ساختار باید:

* لیست کلاس پایه باید شامل نام رابط باشد.
* هر یک از عضو های رابط پیاده سازی شود.

به عنوان مثال، کلاس زیر تعریف جدیدی از کلاس MyClass را نشان می دهد، که رابط IMyInterface1 که در بخش قبل تعریف شده است را پیاده سازی می کند. توجه کنید که نام رابط در لیست کلاس پایه بعد از دو نقطه ذکر شده است و همچین کلاس پیاده سازی عضو های رابط را فراهم کرده است.

دو نقطه نام رابط

↓ ↓

class MyClass: IMyInterface1

{

int DoStuff ( int nVar1, long lVar2 )

{ ... } // پیاده سازی

double DoOtherStuff( string s, long x )

{ ... } // پیاده سازی

}

مواردی که مهم است درباره پیاده سازی رابط ها بدانید به شرح زیر است:

* اگر یک کلاس رابط را پیاده سازی کند، آن باید تمام عضو های رابط را پیاده سازی کند.
* اگر یک کلاس از یک کلاس پایه مشتق شده باشد و نیز همچنین بخواهد رابط ها را پیاده سازی کند، نام کلاس پایه باید در لیست کلاس پایه قبل از رابط ها ذکر شود. ( به یاد داشته باشید که تنها یک کلاس پایه در لیست کلاس پایه می تواند وجود داشته باشد، بنابراین هر نوع دیگری که ذکر می شود باید رابط باشد.)

کلاس پایه باید در ابتدا ذکر شود نام رابط ها

↓ ↓

class Derived : MyBaseClass, IIfc1, IEnumerable, IComparable

{

...

}

مثالی با یک رابط ساده

کد زیر رابط IIfc1 را تعریف می کند، که تنها شامل متد PrintOut است. کلاس MyClass رابط IIfc1 را با لیست کردن در کلاس پایه و تهیه متدی با نام PrintOut (دارای امضا و نوع بازگشتی متد رابط) پیاده سازی می کند. متد Main شی ای از کلاس را ایجاد و متد شی را فراخوانی می کند.

interface IIfc1 Semicolon in place of body // تعریف رابط

{ ↓

void PrintOut(string s);

}

رابط

↓

class MyClass : IIfc1 // تعریف کلاس

{

public void PrintOut(string s) // Implementation

{

Console.WriteLine("Calling through: {0}", s);

}

}

class Program

{

static void Main()

{

MyClass mc = new MyClass(); // Create instance.

mc.PrintOut("object"); // Call method.

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Calling through: object |

رابط یک نوع ارجاعی است

رابط فراتر از یک لیست عضو ها برای کلاس یا ساختار است. این یک نوع مرجع است.

می توان به طور مستقیم به رابط دسترسی پیدا کرد. با این حال، می توان با عمل کست (عملگر مرجع گیری) مرجع شی کلاس مرجع ای برای رابط گرفت. هنگامی که مرجع رابط را در اختیار دارید، می توانید با استفاده از نماد نقطه عضو های رابط را فراخوانی کنید.

به عنوان مثال، کد زیر یک نمونه از گرفتن مرجع رابط شی کلاس را نشان می دهد.

* در اولین دستور، متغییر mc مرجعی برای شی کلاس است که رابط IIfc1 را پیاده سازی کرده است. در دستور کست که مرجع ای برای رابط می گیرد و آن را به متغییر ifc اختصاص می دهد.
* دستور دوم از مرجع رابط برای فراخوانی متد پیاده سازی شده استفاده می کند.

رابط مرجع گیری برای رابط

↓ ↓

IIfc1 ifc = (IIfc1) mc; //دریافت مرجع برای رابط

↑ ↑

مرجع رابط مرجع شی کلاس

ifc.PrintOut ("interface"); // استفاده از مرجع برای فراخوانی عضو

↑

استفاده از نماد نقطه برای فراخوانی از طریق مرجع رابط

به عنوان مثال، کد زیر یک رابط و یک کلاس را تعریف می کند که آن را پیاده سازی می کند. کد در متد Main شی ای از نوع کلاس ایجاد می شود و متد پیاده سازی شده را از طریق شی کلاس فراخوانی می کند. همچنین متغیری از نوع رابط ایجاد می شود و با عمل مرجع گیری از شی کلاس، مرجع ای برای رابط به متغییر تخصیص داده می شود، و متد پیاده سازی شده از طریق مرجع رابط فراخوانی می شود. شکل 4-15 کلاس و مرجع رابط را به تصویر کشیده است.

interface IIfc1

{

void PrintOut(string s);

}

class MyClass: IIfc1

{

public void PrintOut(string s)

{

Console.WriteLine("Calling through: {0}", s);

}

}

class Program

{

static void Main()

{

MyClass mc = new MyClass(); // ایجاد شی کلاس

mc.PrintOut("object"); // فراخوانی متد پیاده سازی شده شی کلاس

IIfc1 ifc = (IIfc1)mc; // مرجع گیری از مرجع شی کلاس برای رابط

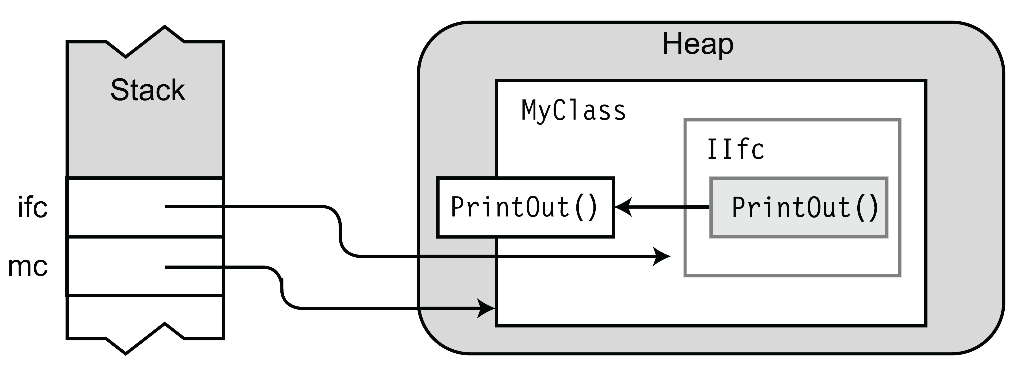
ifc.PrintOut("interface"); // فراخوانی متد رابط

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Calling through: object  Calling through: interface |



شکل 4-15 مرجع ای برای شی کلاس و مرجع ای برای رابط

استفاده از عملگر as با رابط ها

در بخش قبل توجه شدید که می توان با عملگر کست مرجع رابط را از شی دریافت کرد. حتی ایده بهتر استفاده از عملگر as است. عملگر as در فصل 16 پوشش داده شده است، از آنجایی که برای استفاده با رابط ها مناسب است، در اینجا با آن اشاره می کنیم.

اگر اقدام به مرجع گیری از مرجع شی کلاس برای مرجع رابطی کنید، که کلاس آن را پیاده سازی نکرده است، عملیات مرجع گیری یک استثنا تولید می کند. می توان از این مشکل را با استفاده از عملگر as اجتناب کرد. این کار به شرح زیر است:

* اگر کلاس رابط را پیاده سازی کند، عبارت ارجاع به رابط را برگرداند.
* اگر کلاس رابط را پیاده سازی نکند، دستور بجای تولید استثنا null را بر می گرداند. (استثنا ها خطا های غیر منتظره در کد هستند. استثنا ها در فصل 22 بررسی خواهند شد ــ اما شما می خواهید از استثنا ها جلوگیری کنید زیرا آنها به طور قابل توجه ای سرعت برنامه را کاهش می دهند و دریک وضعیت نامناسب برنامه را ترک می کند.)

کد زیر استفاده از عملگر as را نشان می دهد. خط اول از عملگر as برای بدست آوردن مرجع رابط از شی کلاس استفاده می کند. در نتیجه دستور مقدار متغییر b را یا با null و یا با مرجع رابط ILiveBirth تعیین می کند.

خط دوم مقدار متغییر b را بررسی می کند، اگر null نباشد، دستور فراخوانی متد رابط اجرا می شود.

مرجع شی کلاس نام رابط

↓ ↓

ILiveBirth b = a as ILiveBirth; //(ILiveBirth)a مانند عملگر کست :

↑ ↑

مرجع رابط عملگر

if (b != null)

Console.WriteLine("Baby is called: {0}", b.BabyCalled());

پیاده سازی رابط های چند گانه

در مثال هایی که تاکنون نشان داده شده است کلاس ها تنها یک رابط را پیاده سازی می کردند.

* کلاس یا ساختار می تواند هر تعداد رابط را پیاده سازی کند.
* تمام رابط های پیاده سازی شده باید در لیست کلاس پایه توسط کاما از یک دیگر جدا شوند.

به عنوان مثال، کد زیر کلاس MyData را نشان می دهد، که رابط های IDataStore و IDataRetrieve را پیاده سازی می کند. شکل 5-15 پیاده سازی چند گانه را در کلاس MyData به تصویر کشیده است.

interface IDataRetrieve { int GetData(); } // تعریف رابط

interface IDataStore { void SetData( int x ); } // تعریف رابط

رابط رابط

↓ ↓

class MyData: IDataRetrieve, IDataStore // تعریف کلاس

{

int Mem1; // تعریف فیلد

public int GetData() { return Mem1; }

public void SetData( int x ) { Mem1 = x; }

}

class Program

{

static void Main() // Main

{

MyData data = new MyData();

data.SetData( 5 );

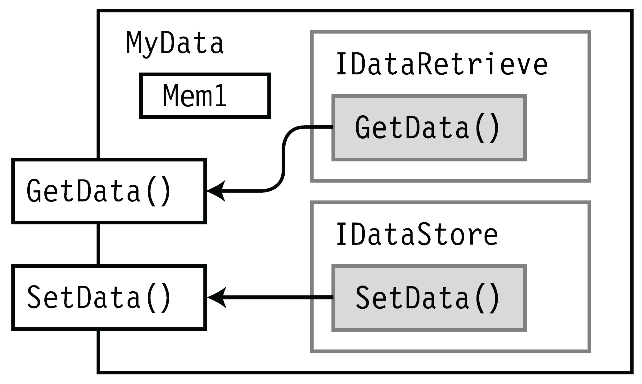
Console.WriteLine("Value = {0}", data.GetData());

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Value = 5 |



شکل 5-15 کلاس پیاده سازی رابط های چند گانه

پیاده سازی رابط ها با عضو ها تکراری

از آنجایی که یک کلاس می تواند هر تعداد رابط را پیاده سازی کند، این امکان وجود دارد که دو یا چندین عضو از رابط ها امضا و نوع بازگشتی یکسانی داشته باشند. بنابراین، کامپایلر به این وضعیت چگونه رسیدگی می کند؟

به عنوان مثال، فرض کنید دو رابط زیر را داشته باشید. هر رابط دارای متد PrintOut با امضا و نوع بازگشتی یکسان هستند. اگر کلاسی برای پیاده سازی هر دو رابط ایجاد کنید. چگونه باید به این متد های تکراری رسیدگی کرد؟

interface IIfc1

{

void PrintOut(string s);

}

interface IIfc2

{

void PrintOut(string t);

}

پاسخ به این سوال این است که اگر کلاس چندین رابط را پیاده سازی می کند که عضو آنها دارای امضا و نوع بازگشتی یکسانی هستند، کلاس می تواند یک عضو پیاده سازی کند که مورد رضایت تمام رابط ها است.

به عنوان مثال، کد زیر کلاس MyClass را نشان می دهد که هر دو رابط IIfc1 و IIfc2 را پیاده سازی می کند.

class MyClass : IIfc1, IIfc2 // پیاده سازی دو رابط

{

public void PrintOut(string s) // یک پیاده سازی برای هر دو رابط

{

Console.WriteLine("Calling through: {0}", s);

}

}

class Program

{

static void Main()

{

MyClass mc = new MyClass();

mc.PrintOut("object");

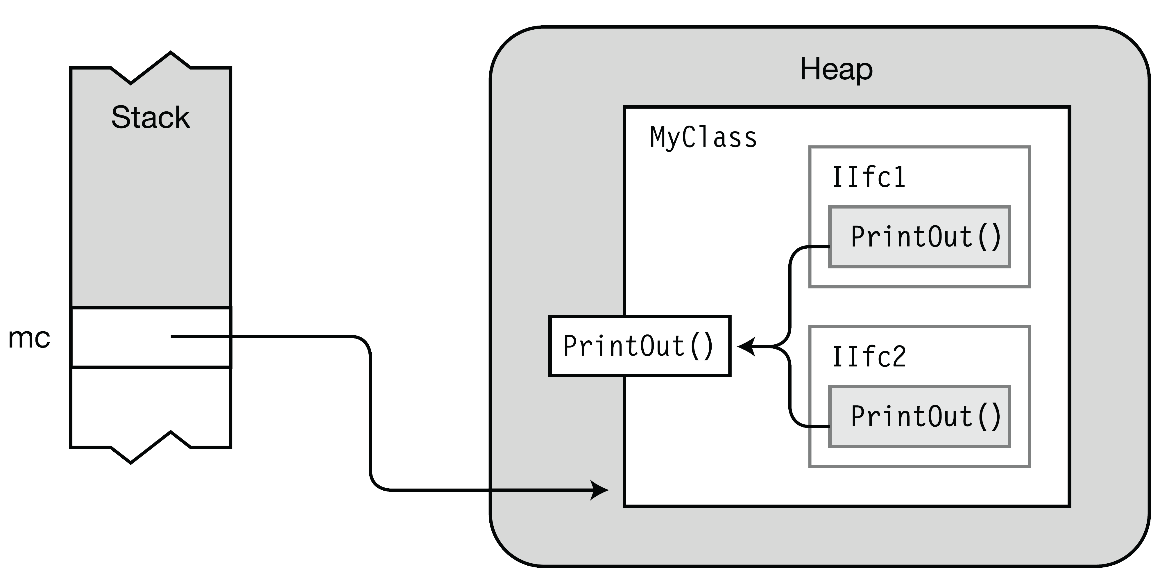
}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Calling through: object |

شکل 6-15 متد هر دو رابط را به تصویر کشیده است، که این متد ها با یک متد در کلاس پیاده سازی شده اند.



شکل 6-15 رابط های چند گانه پیاده سازی شده توسط کلاس

مرجع برای رابط های چند گانه

قبلا مشاهده کردید که رابط ها نوع های ارجاعی هستند و اینکه می توان با عملگر as و یا با کست کردن مرجعی برای رابط گرفت. اگر کلاس رابط های چندگانه را پیاده سازی کند، می توان برای هر رابط مرجع جداگانه ای گرفت.

به عنوان مثال، کلاس زیر دو رابط را پیاده سازی می کند. متد Main با سه روش متد PrintOut را فراخوانی می کند:

* از طریق شی کلاس
* از طریق مرجع رابط IIfc1
* از طریق مرجع رابط IIfc2

شکل 7-15 شی کلاس و مرجع رابط های IIfc1 و IIfc2 را به تصویر کشیده است.

interface IIfc1 // تعریف رابط

{

void PrintOut(string s);

}

interface IIfc2 // تعریف رابط

{

void PrintOut(string s);

}

class MyClass : IIfc1, IIfc2 // تعریف کلاس

{

public void PrintOut(string s)

{

Console.WriteLine("Calling through: {0}", s);

}

}

class Program

{

static void Main()

{

MyClass mc = new MyClass();

IIfc1 ifc1 = (IIfc1) mc; // IIfc1دریافت مرجع برای

IIfc2 ifc2 = (IIfc2) mc; // IIfc2دریافت مرجع برای

mc.PrintOut("object"); // فراخوانی از طریق شی کلاس

ifc1.PrintOut("interface 1"); // IIfc1 فراخوانی از طریق رابط

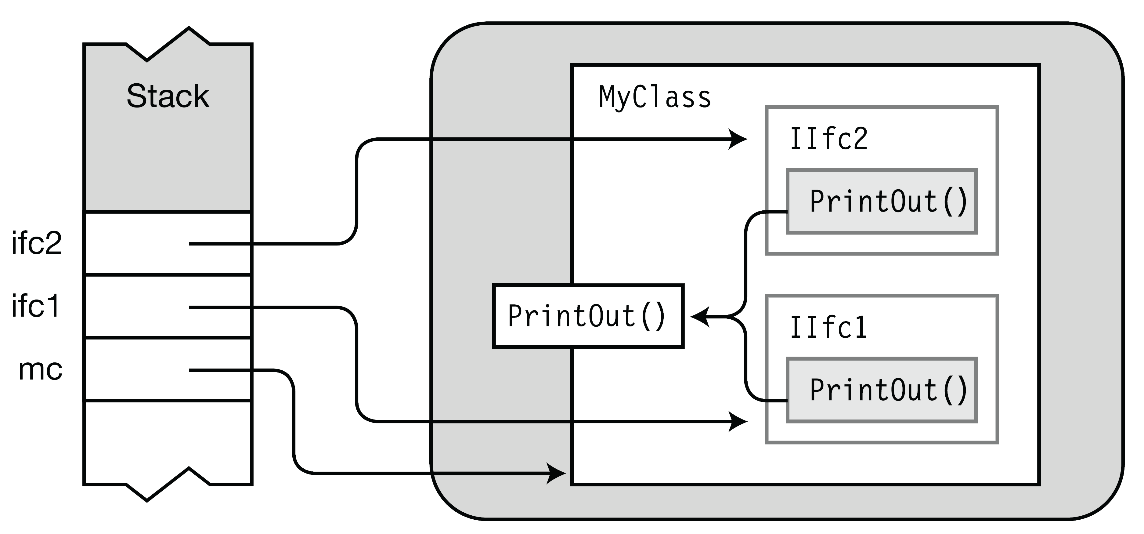
ifc2.PrintOut("interface 2"); // IIfc2فراخوانی از طریق رابط

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Calling through: object  Calling through: interface 1  Calling through: interface 2 |



شکل 7-15 مرجع های جداگانه برای رابط های مختلط در کلاس

عضو به ارث رسیده به عنوان پیاده سازی

کلاسی که رابط را پیاده سازی می کند، می تواند از کدی که از کلاس پایه خود به ارث می برد برای پیاده سازی عضو ها رابط استفاده کند. به عنوان مثال، کد زیر پیاده سازی با استفاده از کلاس پایه را نشان می دهد.

* رابط IIfc1 دارای متد PrintOut است.
* کلاس MyBaseClass شامل متدی با نام PrintOut است که با متد رابط IIfc1 مطابقت دارد.
* کلاس Derived بدنه ای خالی دارد، اما از کلاس MybaseClass مشتق شده است و در لیست کلاس پایه آن رابط IIfc1 وجود دارد.
* حتی اگر بدنه کلاس Derived خالی باشد، کد کلاس پایه می تواند نیاز رابط را برای پیاده سازی متد برطرف کند.

interface IIfc1 { void PrintOut(string s); }

class MyBaseClass // تعریف کلاس پایه

{

public void PrintOut(string s) // تعریف متد

{

Console.WriteLine("Calling through: {0}", s);

}

}

class Derived : MyBaseClass, IIfc1 // تعریف کلاس

{

}

class Program {

static void Main()

{

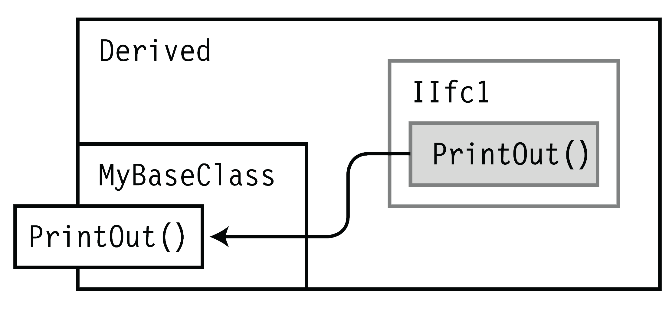
Derived d = new Derived(); // Create class object.

d.PrintOut("object."); // Call method.

}

}

شکل 8-15 کد های قبل را به تصویر کشیده است. توجه کنید که فلش از IIfc1 به کلاس پایه می رود.



شکل 8-15 پیاده سازی در کلاس پایه

پیاده سازی صریح عضو رابط ها

در بخش قبل مشاهده کردید که یک کلاس می تواند تمام عضو های مورد نیاز چند رابط را پیاده سازی کند.

اما چه می شود اگر بخواهید برای هر رابط پیاده سازی جدا گانه ای داشته باشید؟ در این کار با پیاده سازی صریح عضو رابط امکان پذیر است. پیاده سازی صریح عضو رابط دارای مشخصات زیر است:

* مانند تمام پیاده سازی های دیگر، در داخل کلاس یا ساختار قرار می گیرد.
* با استفاده از نام رابط تعریف می شود، که شامل نام رابط و نام عضو است و با نقطه از یک دیگر جدا می شوند.

کد زیر ساختاری برای پیاده سازی صریح عضو رابط نشان می دهد. متد PrintOut هر یک از دو رابط توسط کلاس MyClass پیاده سازی شده است.

class MyClass : IIfc1, IIfc2

{ نام رابط

↓

void IIfc1.PrintOut (string s) // پیاده سازی صریح

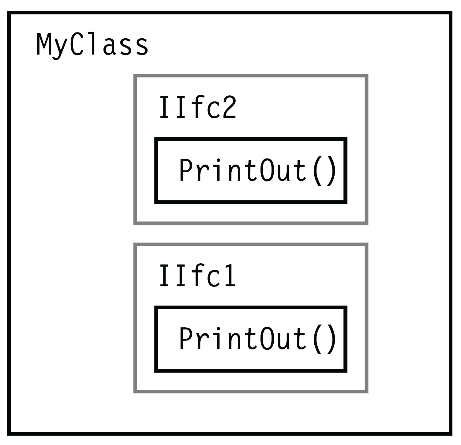
{ ... }

void IIfc2.PrintOut (string s) // پیاده سازی صریح

{ ... }

}

شکل 9-15 کلاس و رابط ها را به تصویر کشیده است. توجه داشته باشید که جعبه ها به صورت خاکستری نیستند و پیاده سازی صریح عضو رابط را نشان می دهند.، زیرا آنها اکنون نشان دهنده کد های واقعی هستند.



شکل 9-15 پیاده سازی صریح عضو رابط

به عنوان مثال، در کد زیر کلاس MyClass عضو های هر دو رابط را به صورت صریح پیاده سازی می کند. توجه داشته باشید که در این مثال پیاده سازی صریح عضو رابط وجود دارد و هیچ پیاده سازی در سطح کلاس وجود ندارد.

interface IIfc1 { void PrintOut(string s); } // تعریف رابط

interface IIfc2 { void PrintOut(string t); } // تعریف رابط

class MyClass : IIfc1, IIfc2

{

نام رابط

↓

void IIfc1.PrintOut(string s) // پیاده سازی صریح عضو رابط

{

Console.WriteLine("IIfc1: {0}", s);

}

نام رابط

↓

void IIfc2.PrintOut(string s) // پیاده سازی صریح عضو رابط

{

Console.WriteLine("IIfc2: {0}", s);

}

}

class Program

{

static void Main()

{

MyClass mc = new MyClass(); // ایجاد شی کلاس

IIfc1 ifc1 = (IIfc1) mc; // IIfc1 دریافت مرجع برای رابط

ifc1.PrintOut("interface 1"); // فراخوانی متد پیاده سازی شده

IIfc2 ifc2 = (IIfc2) mc; // IIfc2 دریافت مرجع برای رابط

ifc2.PrintOut("interface 2"); // فراخوانی متد پیاده سازی شده

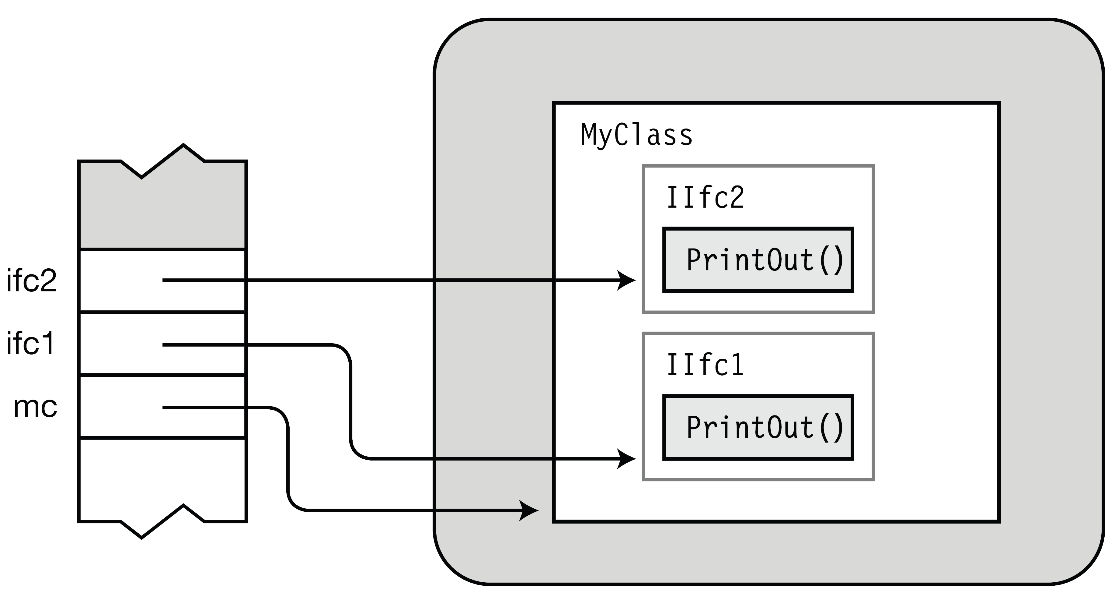
}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| IIfc1: interface 1  IIfc2: interface 2 |

شکل 10-15 کد را به تصویر کشیده است. توجه داشته باشید که متد رابط ها در سطح کلاس پیاده سازی نشده اند، اما کد آنها را شامل می شود.



شکل 10-15 مرجع برای رابط ها با پیاده سازی صریح عضو رابط

هنگامی که پیاده سازی صریح عضو رابط وجود دارد، یک پیاده سازی سطحی کلاس مجاز است اما لازم نیست. کلاس یا ساختار باید متد را پیاده سازی کند. بنابراین می توان سه حالت پیاده سازی زیر را داشت:

* پیاده سازی در سطح کلاس
* پیاده سازی صریح عضو رابط
* هم پیاده سازی در سطح کلاس و هم پیاده سازی صریح عضو رابط

دسترسی به پیاده سازی صریح عضو رابط

پیاده سازی صریح عضو رابط تنها از طریق رابط قابل دسترس است. به این معنا که عضو های کلاس دیگر نمی توانند به طور مستقیم به آنها دسترسی داشته باشند.

به عنوان مثال، کد زیر کلاس MyClass را نشان میدهد که رابط IIfc1 را پیاده سازی می کند. توجه داشته باشید که حتی با وجود اینکه متد Method1 عضوی از کلاس MyClass است، اما می تواند به طور مستقیم به پیاده سازی آن دسترسی داشته باشد.

* دو خط ابتدایی در متد Method1 خطای کامپایلری تولید می کند، زیرا در تلاش است که به طور مستقیم به پیاده سازی دسترسی داشته باشد.
* تنها خط آخر قابل کامپایلر شدن است، زیرا شی جاری (this) به نوع رابط کست داده شده است، و از مرجع رابط برای دسترسی به پیاده سازی صریح عضو رابط استفاده می کند.

class MyClass : IIfc1

{

void IIfc1.PrintOut(string s) // پیاده سازی صریح عضو رابط

{

Console.WriteLine("IIfc1");

}

public void Method1()

{

PrintOut("..."); // خطا در کامپایل کردن

this.PrintOut("..."); // خطا در کامپایل کردن

((IIfc1)this).PrintOut("..."); // فراخوانی متد صحیح است

} ↑

} مرجع گیری برای رابط

این محدودیت دارای یک انشعاب مهم در وراثت است. زیرا سایر عضو های کلاس نمی توانند به طور مستقیم به پیاده سازی صریح عضو رابط دسترسی داشته باشند، عضو های کلاس مشتق شده نمی توانند به طور مستقیم به آنها دسترسی داشته باشند. آنها همیشه از طریق مرجع رابط قابل دسترسی هستند.

ارث بری واسط ها از واسط ها

قبلا دیدید که پیاده سازی های رابط می تواند از کلاس پایه به ارث رسیده شده باشد. اما خود یک رابط می تواند از یک یا چند رابط دیگر ارث بری کند.

* همانطور که در زیر نشان داده شده است، برای مشخص کردن ارث بری یک رابط از رابط های دیگر، نام رابط ها پایه بعد از نام رابط در لیستی که با کاما از یک دیگر جدا شده اند قرار می گیرند. لیست رابط پایه با دو نقطه از رابط جدا می شود.

دو نقطه لیست رابط پایه

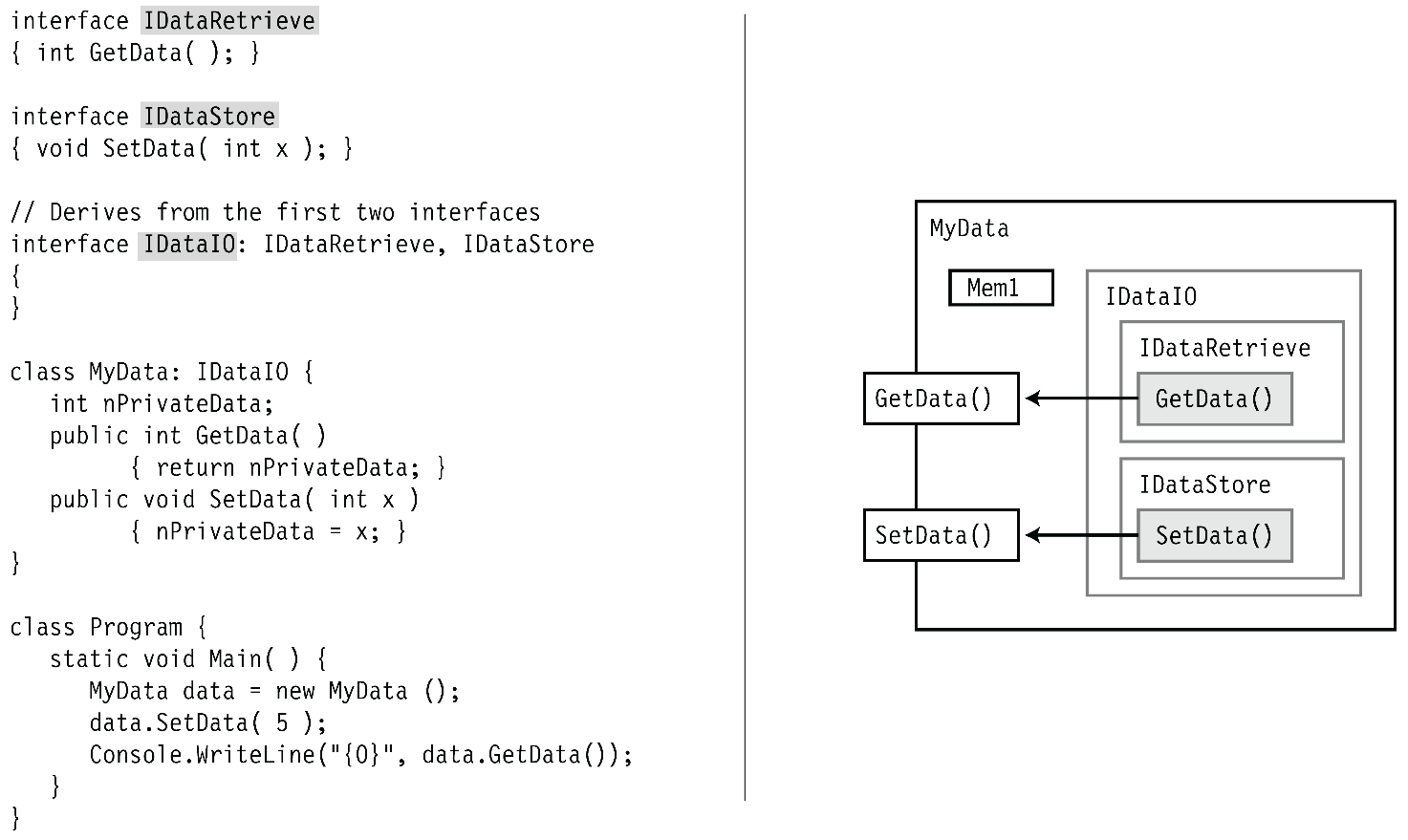
↓ ↓

interface IDataIO : IDataRetrieve, IDataStore

{ ...

* بر خلاف کلاس ها، که تنها یک کلاس می تواند در لیست کلاس پایه خود داشته باشد، رابط ها می توانند هر تعداد رابط را در لیست رابط پایه خود داشته باشند.
  + رابط ها موجود در لیست خودشان می توانند از رابط های دیگر ارث بری کنند.
  + در نتیجه رابط شامل تمام عضو های تعریف شده آن و تمام عضو های رابط های پایه آن است.

کد موجود در شکل 11-15 سه رابط را تعریف می کند. رابط IDataIO از دو مورد اول ارث بری می کند. سمت راست شکل، کد را به تصویر کشیده است.



شکل 11-15 ارث بری چند گانه رابط ها

مثالی از پیاده سازی کلاس های مختلف

کد زیر چندین جنبه از رابط ها را نشان می دهد. این برنامه کلاس Animal را تعریف می کند که به عنوان کلاس پایه ای برای کلاس های دیگر استفاده می شود. همچنین رابط ILiveBirth را تعریف کرده است.

کلاس های Cat، Dog و Bird از کلاس پایه Animal مشتق می شوند. هر دو کلاس Cat و Dog رابط ILiveBirth را پیاده سازی می کنند، اما کلاس Bird این کار را انجام نمی دهد.

در متد Main، ارایه ای کلاس Animal ایجاد می شود و هر یک از عناصر را با یکی از کلاس ها پر می کند. سپس برای هر یک از عناصر ارایه از عملگر as استفاده می کند تا با استفاده از آن مرجعی برای رابط ILiveBirth بگیرد، و متد BabyCalled آن را فراخوانی می کند.

interface ILiveBirth // تعریف رابط

{

string BabyCalled();

}

class Animal // Animal کلاس پایه

{

}

class Cat : Animal, ILiveBirth // Cat تعریف کلاس

{

string ILiveBirth.BabyCalled()

{

return "kitten";

}

}

class Dog : Animal, ILiveBirth // Dog تعریف کلاس

{

string ILiveBirth.BabyCalled()

{

return "puppy";

}

}

class Bird : Animal // Bird تعریف کلاس

{

}

class Program

{

static void Main()

{

Animal[] animalArray = new Animal[3]; // Animal ایجاد ارایه ای از

animalArray[0] = new Cat(); // Cat درج کلاس

animalArray[1] = new Bird(); // Bird درج کلاس

animalArray[2] = new Dog(); // Dog درج کلاس

foreach( Animal a in animalArray ) // چرخه ای برای تمام عناصر آرایه

{

ILiveBirth b = a as ILiveBirth; // پیاده سازی شده...ILiveBirth اگر رابط

if (b != null)

Console.WriteLine("Baby is called: {0}", b.BabyCalled());

}

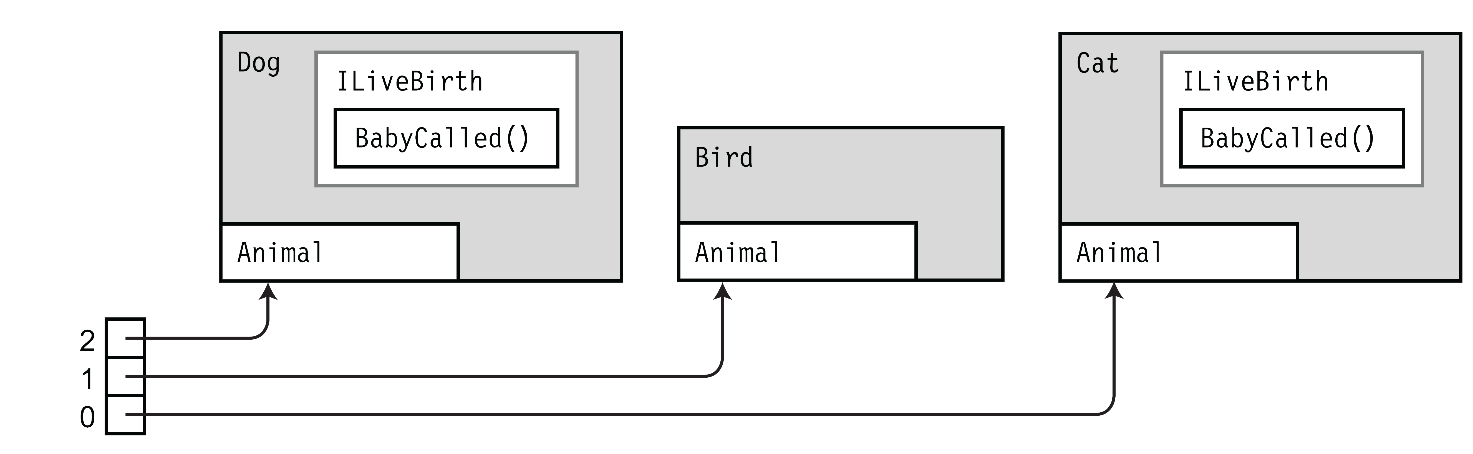
}

}

این کد خروجی زیرا تولید می کند:

|  |
| --- |
| Baby is called: kitten  Baby is called: puppy |

شکل 12-15 آرایه و شی های درون حافظه را به تصویر کشیده است.



شکل 12-15 انواع شی های مختلف از کلاس پایه Animal در آرایه پراکنده شده اند.

1. Interface [↑](#footnote-ref-1)