فصل 4

کلاس ها : مقدمات

* نمای کلی از کلاس ها
* برنامه ها و کلاس ها : مثالی سریع
* تعریف کلاس
* عضو های کلاس
* ایجاد متغییر ها و نمونه هایی از کلاس
* اختصاص حافظه برای داده ها
* عضو های نمونه
* تصریح کننده دسترسی
* عضو های قابل دسترس از داخل کلاس
* عضو های قابل دسترس از خارج کلاس
* جمع بندی

نمای کلی از کلاس ها

در فصل قبل دیدید که C# شش نوع تعریف شده توسط کاربر را فراهم کرده است. مهم ترین آنها، که ابتدا آن را پوشش می دهیم، کلاس است. از آنجایی که مبحث کلاس وسیع است، در طول چند فصل آینده هم درباره آن بحث خواهد شد.

کلاس یک ساختمان داده ای فعال است

روز های قبل از تحلیل و طراحی شی گرایی برنامه نویسان، به برنامه فقط به عنوان دنباله ای از دستورالعمل ها فکر می کردند. تمرکز در آن زمان در ساختار و بهینه سازی آن دستورالعمل ها بود. با ظهور الگو شی گرایی، تمرکز از بهینه سازی دستورالعمل ها به سازماندهی داده های و توابع یک برنامه به داخل مجموعه ای محصور از آیتم ها و توابع منطقی به نام کلاس تغییر کرد.

کلاس یک ساختمان داده ای است که می تواند داده ها را ذخیره و کد ها را اجرا کند. که شامل داده های عضو و توابع عضو است:

* داده های عضو[[1]](#footnote-1)، داده های مرتبط با کلاس یا نمونه ای از کلاس را ذخیره می کند. به طور کلی ویژگی های شی را از دنیای واقعی در کلاس نشان می دهد.
* توابع عضو[[2]](#footnote-2)، کد ها را اجرا می کند. به طور کلی عملکرد ها و اقدامات شی را از دنیای واقعی، در کلاس نمونه سازی می کند.

یک کلاس C# می تواند هر تعدادی از داده ها و توابع را شامل شود. عضو ها می توانند هر ترکیبی از نه عضو باشد. جدول 1-4 این عضوها را نشان می دهد. در این فصل فیلد ها و متد ها بررسی خواهند شد.

جدول 1-4. نوع ها از عضو های کلاس

|  |  |  |
| --- | --- | --- |
| داده های عضوی که داده ها را ذخیره می کند | توابع عضوی که کد ها را اجرا می کند | |
| 🗸 فیلد ها – Fields | 🗸 متد ها – Methods | ❑ عملگر ها – Operator |
| ❑ ثابت ها – Constants | ❑ خاصیت ها – Properties | ❑ ایندکسر ها – Indexers |
|  | ❑ سازنده ها – Constructors | ❑ رویداد ها – Events |
|  | ❑ مخرب ها – Destructors |  |

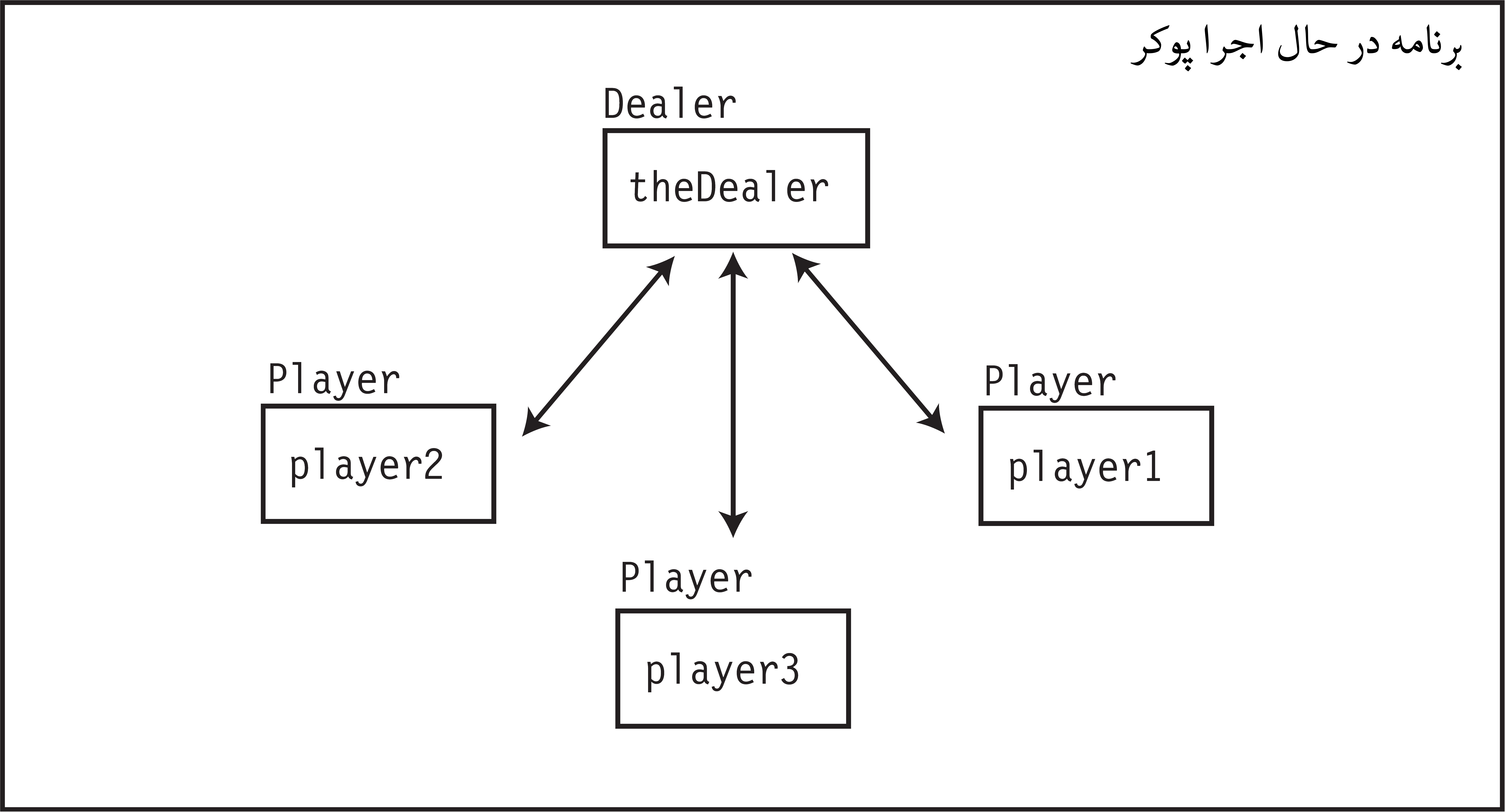
|  |
| --- |
| * توجهکلاس ها مجموعه ای محصور شده از آیتم ها و توابع مرتبط به هم هستند که شی ها را در دنیای واقعی یا در دنیای ادراکی نشان می دهد**.** |

برنامه ها و کلاس ها : مثالی سریع

یک برنامه درحال اجرای، مجموعه ای از نوع ها هستند که با هم در ارتباط هستند و اکثر این نوع ها از کلاس نمونه سازی شده اند. به عنوان مثال، فرض کنید برنامه ای شبیه بازی پوکر دارید. وقتی که آن در حال اجرا است، ممکن است نمونه ای از کلاس Dealer که کار اجرا بازی را انجام می دهد و چندین نمونه از کلاس Player که بازیکنان این بازی را نشان می دهد، داشته باشد.

شی Dealer اطلاعاتی مانند وضعیت فعلی کارت ها و تعداد بازیکنان را ذخیره می کند. وظیفه این شامل مخلوط کردن دسته کارت ها و توزیع کارت ها است.

کلاس Player بسیار متفاوت است. این کلاس اطلاعاتی مانند نام بازیکن و مقدار باقی مانده پول شرط بندی را ذخیره می کند و آن اقداماتی مانند تجزیه و تحلیل دست بازیکن فعلی و قرار دادن شرط را انجام می دهد. شکل 1-4 برنامه درحال اجرای این بازی را نشان می دهد.



شکل 1-4. اشیا یک برنامه در حال اجرا

بدون شک، یک برنامه واقعی علاوه بر کلاس های Dealer و Player از ده ها کلاس دیگر تشکیل می شود. این برنامه شامل کلاس های مانند Card و Deck خواهد شد. هر کلاس چیزی را مدل ساز می کند که جزئی از بازی پوکر است.

|  |
| --- |
| * توجه یک برنامه درحال اجرا مجموعه ای از اشیا هستند که با یکدیگر در حال تعامل اند. |

تعریف کلاس

اگر چه نوع های int و double و char در زبان C# تعریف شده هستند، اما کلاس هایی مانند Dealer و Player همانطور که احتمالا شما هم حدس زده اید در این زبان تعریف نشده اند. اگر بخواهید آنها را در یک برنامه استفاده کنید، باید آنها را خودتان تعریف کنید. این کار را با نوشتن کلاس انجام می گیرد.

تعریف کلاس، ویژگی ها و عضو های کلاس جدید را تعریف می کند. این نمونه ای از کلاس را ایجاد نمی کند بلکه قالبی کلی از این کلاس را به وجود می آورد. که توسط آن نمونه هایی از این کلاس ایجاد خواهد شد. نحوه تعریف کلاس به شرح زیر است:

* نام کلاس
* عضو های کلاس
* ویژگی های کلاس

در زیر نمونه ای از نحوه تعریف کلاس وجود دارد. آکولاد ها حاوی دستورات عضو است که بدنه کلاس را تشکیل می دهد. اعضای کلاس می توانند در هر جایی از بدنه کلاس تعریف شوند.

|  |
| --- |
| کلمه کلیدی نام کلاس  ↓ ↓  class MyExcellentClass  {  *دستورات عضو*  } |

به عنوان مثال، کد زیر خلاصه ای از دو کلاس تعریف شده را نشان می دهد.

|  |
| --- |
| class Dealer // تعریف کلاس  {  ...  }  class Player // تعریف کلاس  {  ...  } |

|  |
| --- |
| * توجه از آنجایی که class declaration برای تعریف کردن یک کلاس جدید استفاده می شود ، اما شما اغلب می بینید که برنامه نویسان و کتاب ها بجای آن از class definition استفاده می کنند. |

عضو های کلاس

فیلد ها و متد ها مهم ترین نوع ها از عضو های کلاس هستند. فیلد ها، داده های عضو و متد ها، توابع عضو هستند.

فیلد ها

فیلد متغیری است که به کلاس تعلق دارد.

* این می تواند از نوع های از پیش تعریف شده و یا از نوع های تعریف شده توسط کاربر باشد.
* مانند تمام متغییر ها، فیلد ها داده را ذخیره می کند و دارای ویژگی های زیر است:
  + می توان بر روی آنها نوشت
  + می توان از روی آنها خواند

نحوه تعریف یک فیلد به شرح زیر است:

|  |
| --- |
| نوع  ↓  *Type Identifier*;  ↑  نام فیلد |

به عنوان مثال، کلاس زیر شامل فیلد MyField است که می تواند مقداری از نوع int را ذخیره کند.

|  |
| --- |
| class MyClass  { نوع  ↓  int MyField;  ↑  } نام فیلد |

|  |
| --- |
| * توجه بر خلاف C و C++ ، در C# هیچ متغییر سراسری (Global) در بیرون از یک نوع (کلاس) وجود ندارد. همه فیلد ها متعلق به یک نوع است و باید در داخل آن نوع تعریف شود. |

مقداردهی اولیه صریح و ضمنی فیلد ها

از آنجایی که فیلد نوعی متغییر است، نحوه مقداردهی یک فیلد همانند مقداردهی متغییر ها است که در فصل قبل نشان داده شد.

* مقداردهی فیلد بخشی از تعریف فیلد است و شامل یک علامت مساوی به همراه عبارتی است که با آن مقداردهی می شود.
* مقداردهی اولیه باید در زمان کامپایل شدن تعیین شود باشد.

|  |
| --- |
| class MyClass  {  int F1 = 17;  } ↑  مقداردهی فیلد |

* اگر مقداردهی اولیه استفاده نشود، کامپایلر مقدار پیش فرضی را به فیلد اختصاص می دهد، که این مقدار پیش فرض توسط نوع فیلد تعیین می شود. جدول 1-3 ( در فصل 3) مقادیر پیش فرض را برای نوع های ساده نشان می دهد. به طور خلاصه، مقدار پیش فرض برای هر یک از نوع ها 0 ( صفر) و برای نوع های بولی false است. مقدار پیش فرض برای نوع های ارجاعی، null است.

به عنوان مثال، کد زیر چهار فیلد را تعریف می کند. دو فیلد اول به صورت ضمنی مقداردهی شده اند. دو فیلد بعدی به صورت صریح با مقدار اولیه مقداردهی شده اند.

|  |
| --- |
| class MyClass  {  int F1; // مقدار دهی اولیه با 0 ــ نوع مقداری  string F2; //ــ نوع ارجاعی null مقداردهی اولیه با  int F3 = 25; // مقدار دهی اولیه با 25  string F4 = "abcd"; // "abcd" مقداردهی اولیه با  } |

تعریف فیلد های متعدد

می توان با جدا کردن نام فیلد ها با کاما، در یک دستور چندین فیلد از یک نوع را تعریف کرد. نمی توان نوع های مختلفی را در یک دستور تعریف کرد. به عنوان مثال، می توان چهار فیلد تعریف شده قبل را، در دو دستور با همان نتایج تعریف کرد.

|  |
| --- |
| int F1, F3 = 25;  string F2, F4 = "abcd"; |

متد ها

یک متد نام بلوکی از کد های اجرایی است که می تواند بسیاری از قسمت های مختلف برنامه و حتی برنامه ای دیگر را اجرا کند. ( همچنین متد های ناشناسی هم هستند که نام ندارند، ما آنها را در فصل 13 بررسی خواهیم کرد.)

هنگامی که یک متد فراخوانی می شود، کد های خود را اجرا می کند و برمیگردد به کدی که آنرا فراخوانی کرده است. بعضی از متد ها مقداری را به مکان فراخوان برمی گرداند. متد ها همانند توابع عضو در C++ هستند.

کم ترین دستور برای تعریف یک متد شامل موارد زیر است:

* نوع بازگشتی : نوع مقداری که متد برگشت می دهد را تعیین می کند. اگر متدی مقدار برگشت ندهد نوع بازگشتی با void مشخص می شود.
* نام : نام متد را مشخص می کند.
* لیست پارامتر ها: حداقل از یک جفت پرانتز شامل شده است. اگر پارامتری وجود داشته باشد ( در فصل بعد بررسی خواهند شد) ، بین پرانتز ها قرار می گیرد.
* بدنه متد: شامل یک جفت براکت است که حاوی کد های اجرایی است.

به عنوان مثال، کد زیر یک کلاس با متد ساده ای به نام PrintNums تعریف کرده است. با توجه به نحوه تعریف، شما می توانید موارد زیر را درباره PrintNums بگویید:

* نوع بازگشتی به صورت void تعیین شده است؛ از این رو، هیچ مقداری را برگشت نمی دهد.
* لیست پارامترهای آن خالی است. از این رو هیچ مقداری به متد ارسال نمی شود.
* بدنه متد دو خط دستور را شامل شده است. ابتدا عدد 1 و سپس عدد 2 را چاپ می کند.

|  |
| --- |
| class SimpleClass  {  نوع برگشتی لیست پارامترها  ↓ ↓  void PrintNums( )  {  Console.WriteLine("1");  Console.WriteLine("2");  }  } |

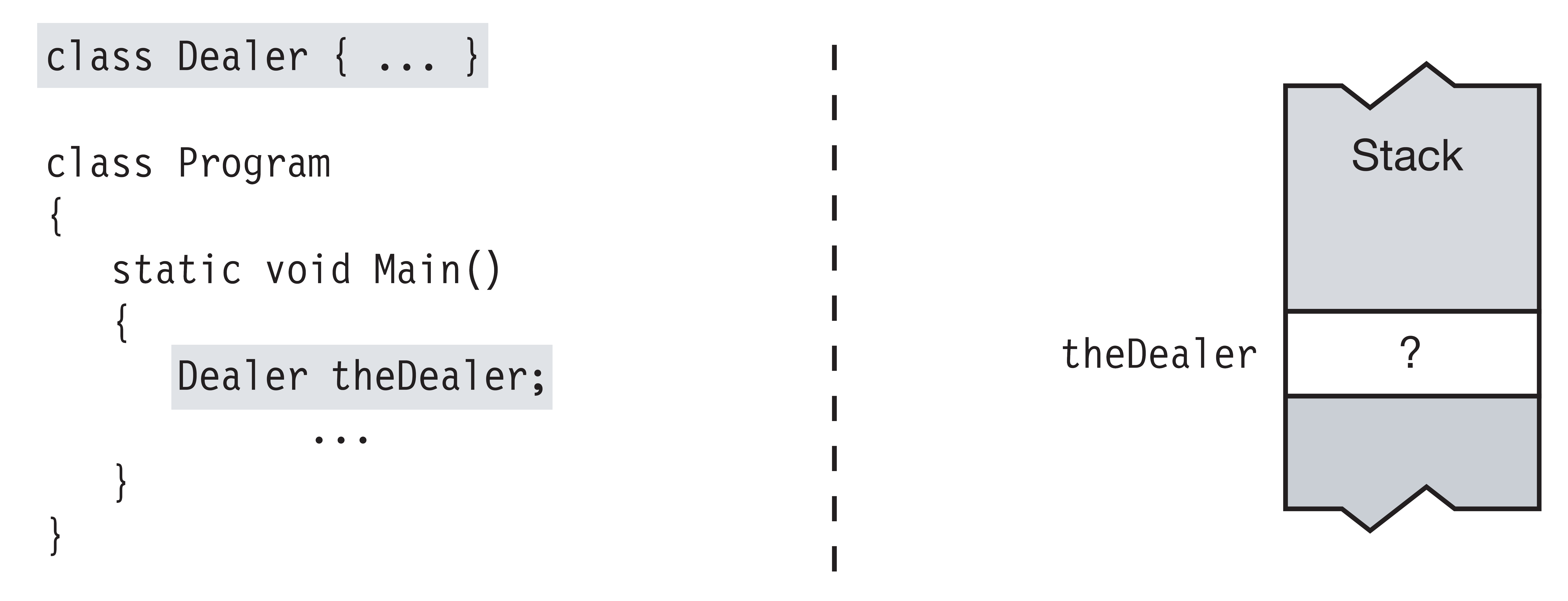
|  |
| --- |
| * توجه برخلاف C و C++، در C# توابع سراسری (Global) بیرون از نوع تعریف شده وجود ندارد. همچنین، برخلاف C و C++ ، نوع بازگشتی پیش فرض برای متد وجود ندارد. همه متد ها باید شامل نوع بازگشتی یا void باشد. |

ایجاد متغییر ها و نمونه هایی از کلاس

تعریف کردن کلاس تنها طرحی از کلاس است. هنگامی که یک کلاس تعریف می شود، می توان نمونه هایی از کلاس را ایجاد کرد.

* از فصل قبل به یاد دارید که کلاس ها نوع های ارجاعی بودند، به این معنی که آنها به هر دو حافظه برای ارجاع به داده ها و برای داده های واقعی نیاز دارند.
* ارجاع به داده ها در متغییری از نوع کلاس ذخیره می شود. بنابراین ، برای ایجاد نمونه ای از کلاس، نیاز است که ابتدا متغییری از نوع کلاس تعریف شود. اگر متغییر مقداردهی نشده باشد، مقدار آن نامشخص است.

شکل 2-4 چگونگی تعریف متغییر را نشان می دهد. سمت چپ تصویر و بالای کد، کلاس Dealer تعریف شده است. در زیر آن کلاس Program تعریف شده است که شامل متد Main است. در Main متغییر theDealer از نوع Dealer تعریف شده است. از آنجایی که این متغییر مقداردهی نشده است ، مقدار آن نامشخص است، همانطور که در سمت راست تصویر هم نشان داده شده است.



شکل 2-4. اختصاص حافظه برای متغییر کلاس

اختصاص حافظه برای داده ها

هنگامی که متغییری از نوع کلاس تعریف می شود، حافظه تخصیص یافته برای نگه داری مرجع است و برای نگه داری داده های واقعی نیست. برای تخصیص حافطه به داده های واقعی باید از عملگر new کرد.

* عملگر new برای هر نمونه ای با هر نوع مشخص شده ای مقداردهی اولیه و تخصیص حافظه را انجام می دهد. این تخصیص حافظه بسته به نوع آن می تواند از stack یا از heap باشد.
* از عملگر new برای دستورهایی که یک شی را ایجاد می کند استفاده می شود، که شامل موارد زیر است:
  + کلمه کلیدی new
  + نام نوعی که برای تخصیص حافظه از آن نمونه سازی می شود.
  + جفت پرانتزها ایی ، که ممکن است شامل پارامتر هم باشد. در مورد پارامتر ها بعداً صحبت خواهد شد.

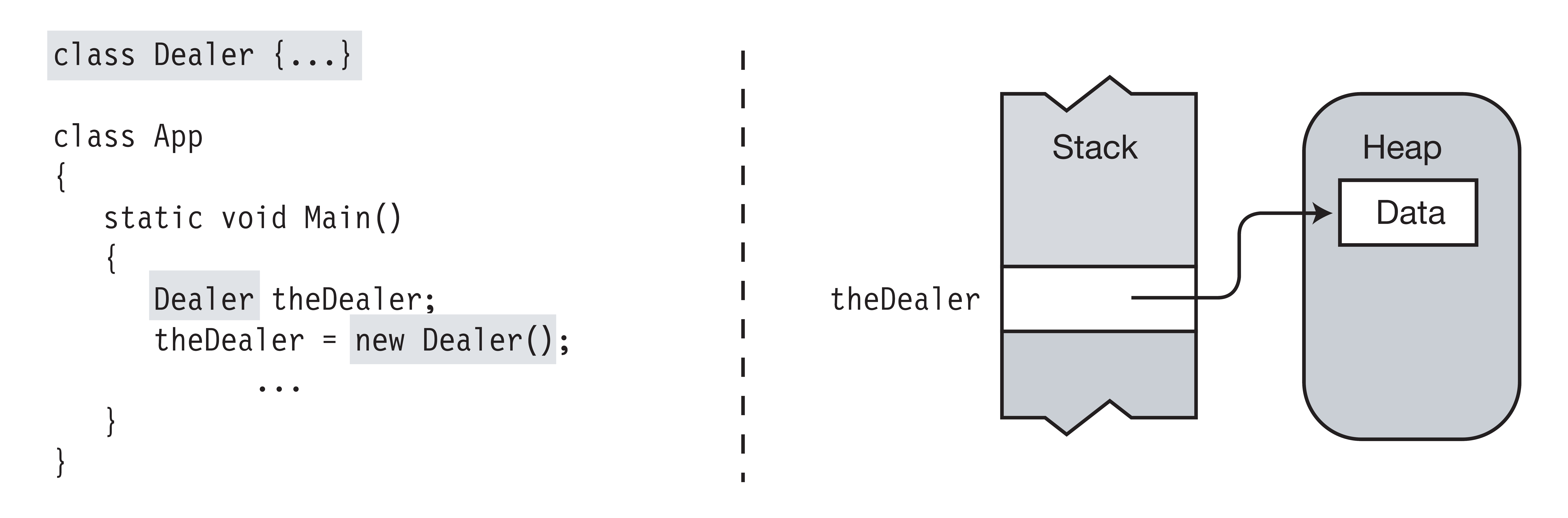
|  |
| --- |
| کلمه کلیدی پرانتز ها الزامی است  ↓  ↓  new *TypeName* ( )  ↑  نوع |

* اگر حافظه تخصیص داده شده برای یک نوع ارجاعی باشد، دستور ایجاد کننده شی، مرجع شی نمونه سازی شده ( که در heap به آن حافظه تخصیص یافته و مقداردهی شده است) را برمی گرداند.

این دقیقاً همان چیزی است که شما نیاز دارید تا بتوانید داده های یک کلاس نمونه سازی شده را مقدار دهی کرده و برای آنها حافظه تخصیص دهید. از عملگر new در دستور ایجاد کننده شی استفاده کنید، و متغییر کلاس را با متغییر برگشتی مقداردهی کنید. به عنوان مثال:

|  |
| --- |
| Dealer theDealer; // تعریف متغییر برای مرجع  theDealer = new Dealer(); // تخصیص حافظه برای شی  ↑  دستور ایجاد کننده شی |

کد سمت چپ در شکل 3-4 از عملگر new برای تخصیص حافظه استفاده کرده است و یک نمونه از کلاس Dealer ایجاد و به متغییر اختصاص داده است. ساختار حافظه در سمت راست کد نشان داده شده است.



شکل 3-4. اختصاص حافظه برای داده های از یک متغییر کلاس

ترکیب مراحل

می توان دو مرحله، مقداردهی اولیه متغییر و دستور ایجاد کننده شی را با هم ترکیب کرد.

|  |
| --- |
| تعریف متغییر  ↓  Dealer theDealer = new Dealer(); // تعریف و مقداردهی اولیه  ↑  مقداردهی با یک دستور ایجاد کننده شی |

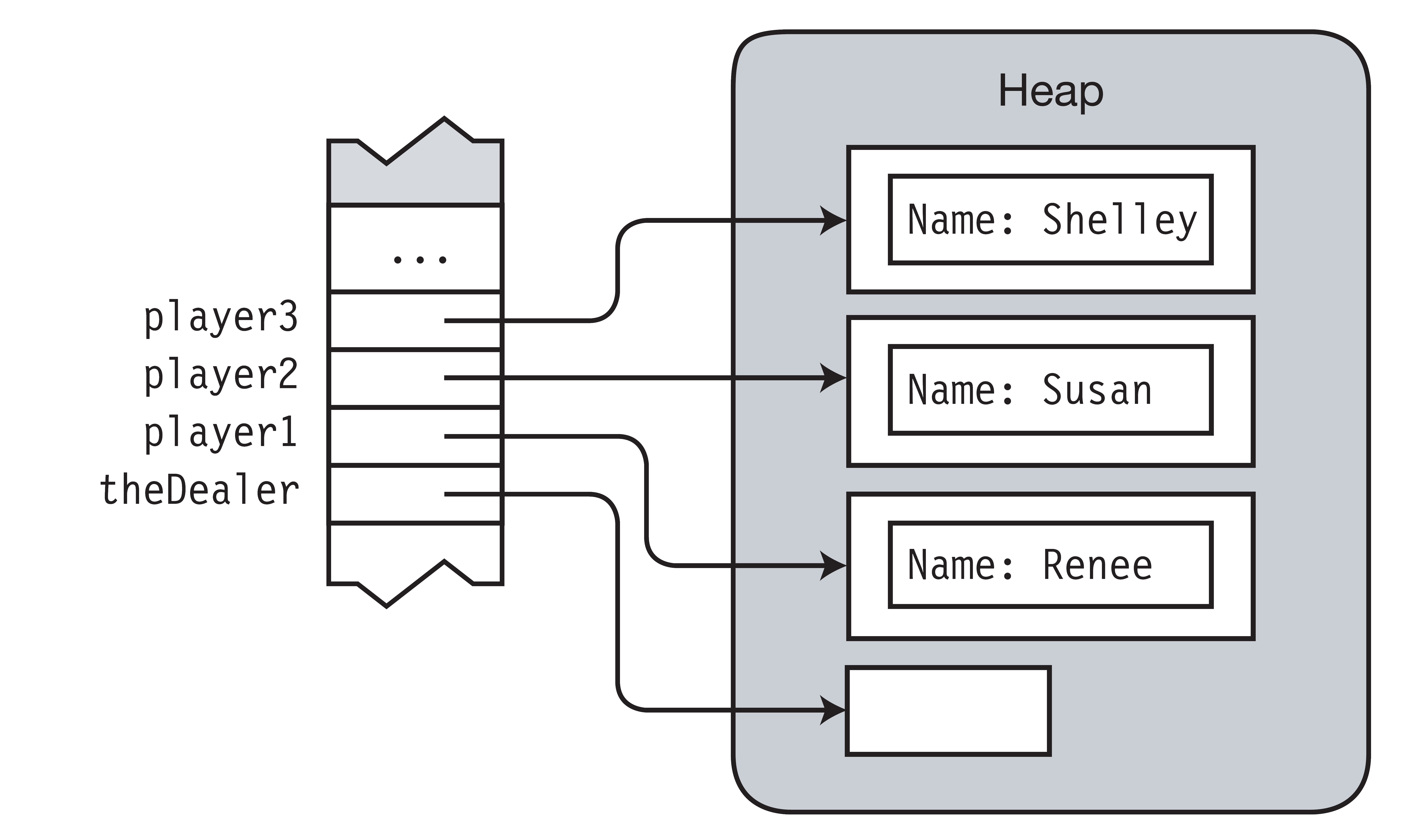
عضو های نمونه

تعریف کردن کلاس در عمل به عنوان طرحی است که توسط آن می توان نمونه هایی بسیاری از کلاس را ایجاد کرد.

* عضو های نمونه: هر نمونه ای از کلاس یک نهاد مجزا است که مجموعه ای از داده های عضو است و از نمونه های دیگر کلاس متمایز است. از آن جایی که آنها با نمونه ای از کلاس ارتباط دارند عضو های نمونه نامیده می شوند.
* عضو های استاتیک: عضو های پیش فرض هستند که قبل از اینکه نمونه ای کلاس ایجاد شود با کلاس در ارتباط هستند. ما آن را در فصل 6 بررسی خواهیم کرد.

به عنوان مثالی از عضو های نمونه، کد زیر بازی پوکر را با سه نمونه از کلاس Player نشان می دهد. شکل 4-4 نشان می دهد که هر نمونه دارای یک مقدار متفاوتی برای نام فیلد است.

|  |
| --- |
| class Dealer { ... } // تعریف کلاس  class Player { // تعریف کلاس  string Name; // فیلد  ...  }  class Program {  static void Main()  {  Dealer theDealer = new Dealer();  Player player1 = new Player();  Player player2 = new Player();  Player player3 = new Player();  ...  }  } |



شکل 4-4. عضو های نمونه دارای مقادیر مجزا بین اشیا کلاس هستند.

تصریح کننده دسترسی

در یک کلاس ، توابع عضو به راحتی می توانند به هر عضوی از کلاس دسترسی داشته باشند.

تصریح کننده دسترسی یک بخش اختیاری در تعریف عضو ها است که مشخص می کند چه بخش هایی دیگر از برنامه به عضو دسترسی داشته باشند. تصریح کننده دسترسی قبل از نوع عضو قرار می گیرد. نحوه استفاده برای متد ها و فیلد ها به شرح زیر است:

|  |
| --- |
| فیلد ها  *تصریح کننده دسترسی نوع شناسه*  متد ها  *تصریح کننده دسترسی نوع برگشتی نام متد* ()  {  ...  } |

پنج دسته از تصریح کننده های دسترسی به شرح زیر هستند. در این فصل دو مورد از آنها را توضیح خواهیم داد و بقیه موارد را در فصل 7 توضیح می دهیم.

* private
* public
* protected
* internal
* protected internal

تصریح کننده های private و public

عضو های private فقط از داخل کلاس که در آن تعریف شده است قابل دسترسی هستند و کلاس های دیگر نمی توانند به آنها دسترسی داشته باشند یا آنها را ببینند.

* تصریح کننده private به صورت پیش فرض است. بنابراین اگر یک عضو بدون تصریح کننده دسترسی تعریف شده باشد، آن عضو به صورت private است.
* همچنین شما می توانید به صراحت از تصریح کننده دسترسی private به عنوان عضو خصوصی استفاده کنید. هیچ تفاوتی بین تعریف عضو private به صورت ضمنی ( عضوی که بدون تصریح کننده دسترسی تعریف شده باشد) و به صورت صریح ( عضوی که از تصریح کننده دسترسی private استفاده کند) وجود ندارد.

به عنوان مثال، در کد زیر هر دو نوع تعریف شده به صورت private هستند.

|  |
| --- |
| int MyInt1; // به صورت ضمنی privat تعریف  private int MyInt2; // به صورت صریح private تعریف  ↑  تصریح کننده دسترسی |

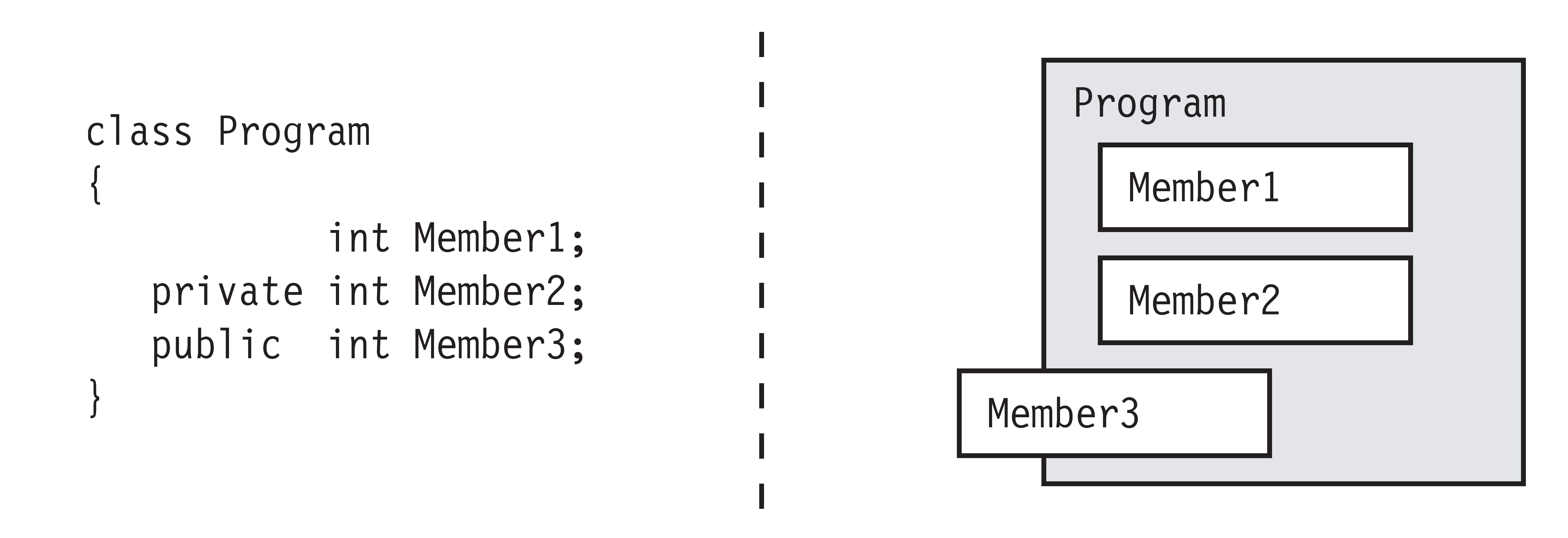
عضو های public برای تمام اشیا برنامه قابل دسترس هستند. شما باید از تصریح کننده دسترسی public برای دسترسی عمومی استفاده کنید.

|  |
| --- |
| تصریح کننده دسترسی  ↓  public int MyInt; |

شرح تصریح کننده های private و public

همانطور که در شکل 5-4 نشان داده شده است، کلاس به عنوان جعبه ای نشان داده شده است.

* اعضای کلاس به عنوان جعبه های کوچکتری در داخل جعبه کلاس نشان داده شده اند.
* عضو های خصوصی به طور کامل در داخل جعبه کلاس محصور شده اند.
* عضو های عمومی تا حدی در خارج از جعبه کلاس قرار دارند.

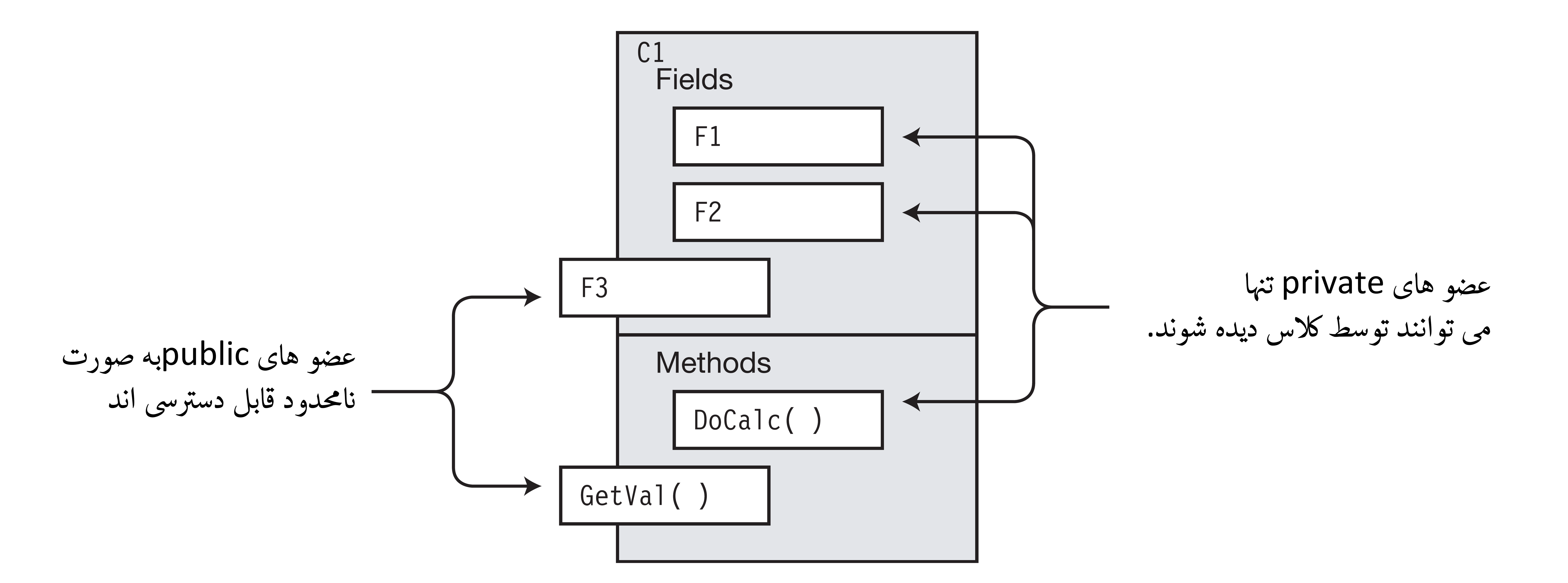


شکل 5-4. نشان دهنده کلاس ها و اعضای آن

مثالی از تصریح کننده دسترسی

در کد زیر کلاسC1 هر دو تصریح کننده عمومی و خصوصی را هم برای فیلد ها و هم برای متد ها تعریف کرده است. شکل 6-4 نمایی از اعضای کلاس C1 را نشان می دهد.

|  |
| --- |
| class C1  {  int F1; // تعریف فیلد خصوصی به صورت ضمنی  private int F2; // تعریف فیلد خصوصی به صورت صریح  public int F3; // تعریف فیلد عمومی  void DoCalc() // تعریف متد خصوصی به صورت ضمنی  {  ...  }  public int GetVal() // تعریف متد عمومی  {  …  }  } |



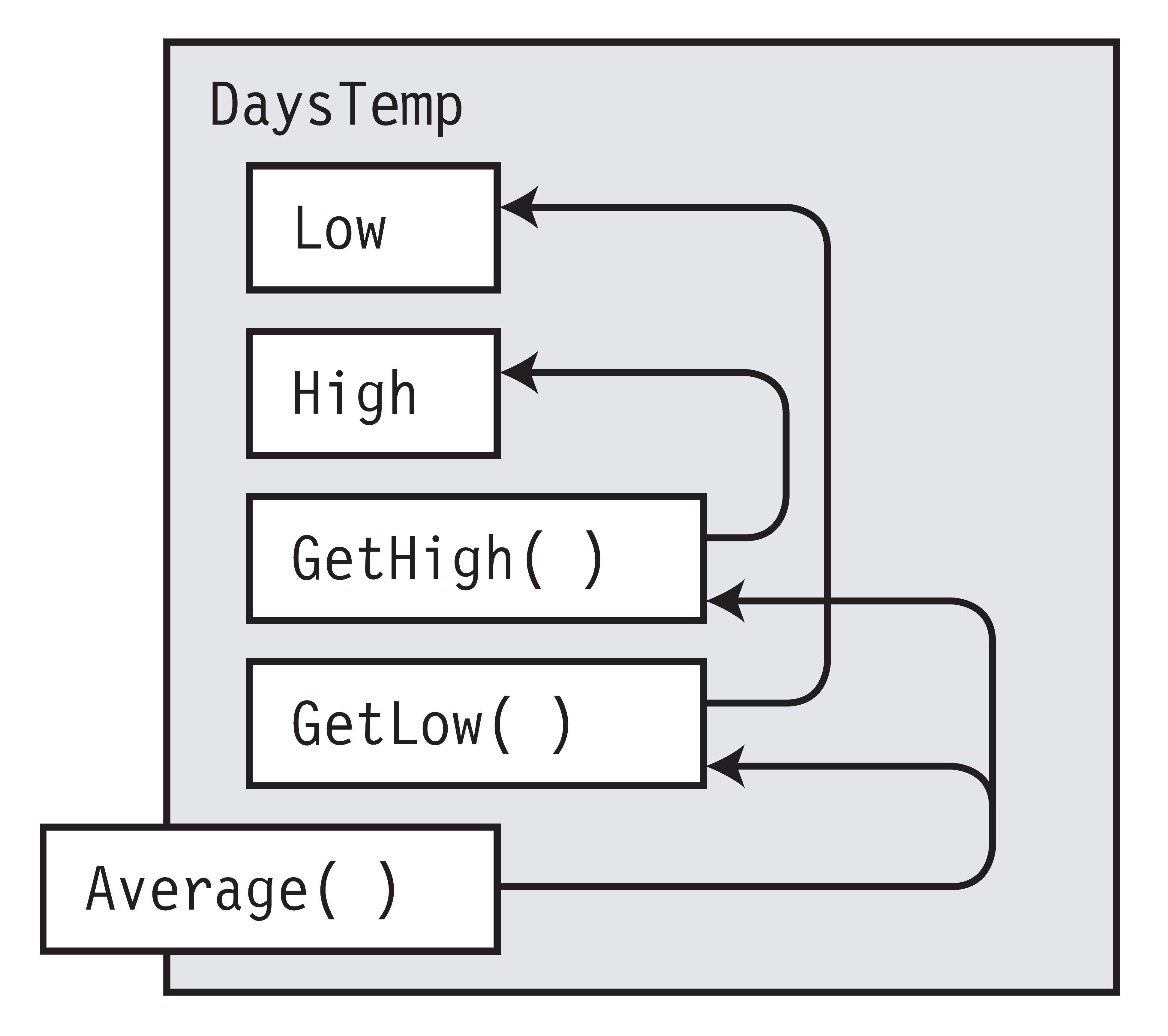
شکل 6-4. عضو های خصوصی و عمومی کلاس

عضو های در دسترس از داخل کلاس

همانطور که گفته شد، اعضای یک کلاس می توانند به سایر اعضای کلاس ( تنها با استفاده از نام آنها) دسترسی داشته باشند.

به عنوان مثال، کلاس زیر نشان دهنده متد هایی از کلاس است که به دیگر متد ها و فیلد ها دسترسی پیدا می کند. حتی اگر فیلد ها و هر دو متد به صورت خصوصی تعریف شده باشند، همه عضو های کلاس می توانند به هر متدی ( یا هر تابع عضوی) از کلاس دسترسی داشته باشند. شکل 7-4 نمایی از کد را نشان می دهد.

|  |
| --- |
| class DaysTemp  {  // فیلد ها  private int High = 75;  private int Low = 45;  // متد ها  private int GetHigh()  {  return High; // دسترسی به فیلد خصوصی  }  private int GetLow()  {  return Low; // دسترسی به فیلد خصوصی  }  public float Average ()  {  return (GetHigh() + GetLow()) / 2; // دسترسی به متدهای خصوصی  } ↑ ↑  } دسترسی به متدهای خصوصی |



شکل 7-4. عضوها درون یک کلاس می توانند آزادانه با یکدیگر دسترسی داشته باشند.

عضو های در دسترس از خارج از کلاس

برای دسترسی به یک عضو عمومی از خارج از کلاس، باید شامل نام متغییر (کلاس) و نام عضو باشد و با نقطه از یکدیگر جدا شده باشند. این، نحوه نماد گذاری نقطه ای[[3]](#footnote-3) نامیده می شود؛ بعداً در جزئیات بیشتری در مورد آن بحث خواهد شد.

به عنوان مثال، خط دوم از کد زیر نمونه ای از دسترسی به یک متد از خارج از کلاس را نشان می دهد.

|  |
| --- |
| DaysTemp myDt = new DaysTemp(); // ایجاد یک شی از کلاس  float fValue = myDt.Average(); // دسترسی به عضو از خارج از کلاس  ↑ ↑  نام متغییر نام عضو |

به عنوان مثال، در کد زیر دو کلاس DaysTemp و Program تعریف شده است.

* دو فیلد در DaysTemp به صورت عمومی تعریف شده اند، بنابراین می توان از خارج از کلاس به آنها دسترسی داشت.
* متد Main عضوی از کلاس Program است. آن یک متغییر و شی از کلاس DaysTemp ایجاد کرده است، و فیلد های شی را مقدار دهی کرده است، و سپس مقدار فیلد را می خواند و آن را چاپ می کند.

|  |
| --- |
| class DaysTemp // DaysTemp تعریف کلاس  {  public int High = 75;  public int Low = 45;  }  class Program // Program تعریف کلاس  {  static void Main()  { نام متغییر  ↓  DaysTemp temp = new DaysTemp(); // ایجاد شی  نام متغییر و نام فیلد  ↓  temp.High = 85; // دسترسی به فیلد ها  temp.Low = 60; نام متغییر و نام فیلد  ↓  Console.WriteLine("High: {0}", temp.High ); // خواندن فیلد ها  Console.WriteLine("Low: {0}", temp.Low );  }  } |

این کد خروجی زیر را تولید می کند.

|  |
| --- |
| High: 85  Low: 60 |

**جمع بندی**

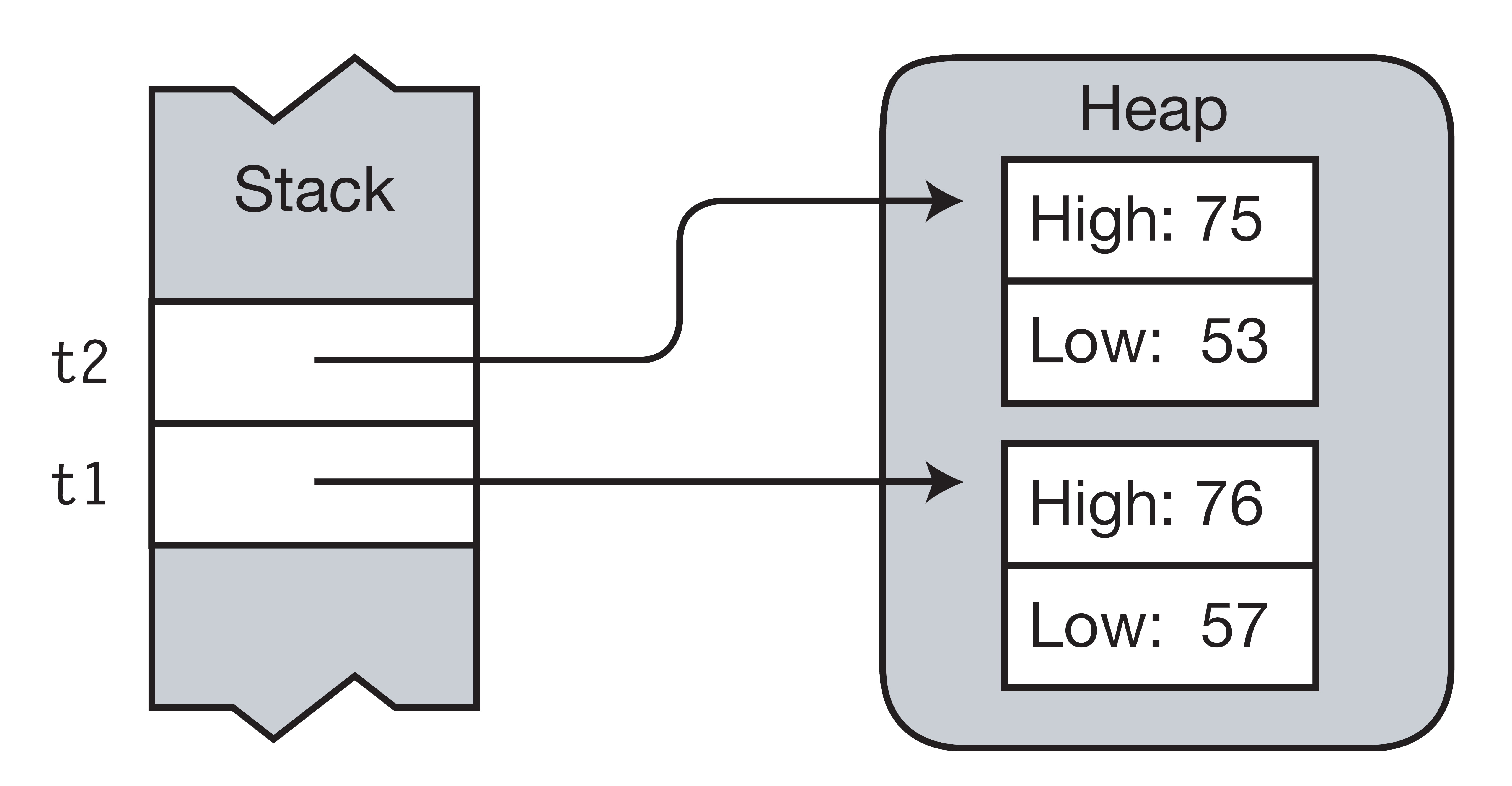
کد زیر دو نمونه را ایجاد می کند و مرجع آنها را در متغییر های t1 و t2 ذخیره می کند. شکل 8-4 طریقه ذخیره سازی t1 و t2 را در حافظه نشان می دهد. کد زیر سه عملی که تاکنون درباره کلاس استفاده شده است را نشان می دهد:

* تعریف کلاس
* ایجاد نمونه ای از کلاس
* دسترسی به اعضای کلاس ( مقدار دهی و فراخوانی فیلد ها )

|  |
| --- |
| class DaysTemp // تعریف کلاس  {  public int High, Low; // تعریف فیلد  public int Average() // تعریف متد  {  return (High + Low) / 2;  }  }  class Program  {  static void Main()  {  // DaysTempایجاد دو نمونه از  DaysTemp t1 = new DaysTemp();  DaysTemp t2 = new DaysTemp();  // مقداردهی فیلد های هر یک از نمونه های کلاس  t1.High = 76; t1.Low = 57;  t2.High = 75; t2.Low = 53;  // فراخوانی فیلد ها و متدها از هر یک از نمونه ها  Console.WriteLine("t1: {0}, {1}, {2}",  t1.High, t1.Low, t1.Average() );  Console.WriteLine("t2: {0}, {1}, {2}",  t2.High, t2.Low, t2.Average() );  ↑ ↑ ↑  } فیلد فیلد متد  } |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| t1: 76, 57, 66  t2: 75, 53, 64 |



شکل 8-4. قالب قرار گیری حافظه از نمونه های t1 و t2

1. Data Members [↑](#footnote-ref-1)
2. Function Members [↑](#footnote-ref-2)
3. dot-syntax notation [↑](#footnote-ref-3)