# فصل سوم

## مقدمه‌ای بر کلاس‌ها و شی‌ها

### اهداف

- کلاس‌ها، شی‌ها، توابع عضو و داده چیستند.
- نحوه تعریف کلاس و استفاده از آن در ایجاد کلاس.
- نحوه تعریف توابع عضو در کلاس برای پیاده‌سازی رفتار کلاس.
- نحوه اعلان اعضای داده در کلاس برای پیاده‌سازی صفات کلاس.
- فراخوانی تابع عضو یک شی برای اینکه تابع عضو وظیفه خود را به انجام رساند.
- تفاوت موجود مابین اعضای داده یک کلاس و متغیرهای محلی یک تابع.
- نحوه استفاده از سازنده برای اطمینان از اینکه داده یک شی به هنگام ایجاد شی مقداردهی اولیه شده است.
- نحوه طراحی یک کلاس برای مجزا شدن واسط آن از بخش پیاده‌سازی و افزودن قابلیت استفاده مجدد به آن.

www.yourshop.4kia.ir

**رئوس مطالب**


**۳-۱ مقدمه**
**۳-۲ کلاس‌ها، شی‌ها، توابع عضو و داده**
**۳-۳ نگاهی بر مثال‌های این فصل**
**۳-٤ تعریف کلاس با یک تابع عضو**
**۳-٥ تعریف تابع عضو با پارامتر**
**۳-۶ اعضای داده، توابع set و get**
**۳-۷ مقداردهی اولیه شی‌ها با سازنده‌ها**
**۳-۸ قرار دادن کلاس در یک فایل مجزا برای استفاده مجدد**
**۳-۹ جداسازی واسط از پیاده‌سازی**
**۳-۱۰ اعتبارسنجی داده با توابع set**
**۳-۱۱ مبحث آموزشی مهندسی نرم‌افزار: شناسایی کلاس‌های موجود در مستند نیازهای ATM**


## ۳-۱ مقدمه

در فصل دوم، چند برنامه ساده ایجاد کردیم که می‌توانستند پیغام‌های را به کاربر نشان داده، اطلاعاتی از وی دریافت نمایند، محاسباتی انجام داده و تصمیم‌گیری کنند. در این فصل، شروع به نوشتن برنامه‌هایی می‌کنیم که مفاهیم پایه برنامه‌نویسی شی‌گرای معرفی شده در بخش ۱۷-۱ را بکار می‌گیرند. یکی از ویژگی‌های مشترک در هر برنامه فصل دوم این است که تمام عبارات که کاری انجام می‌دهند در تابع **main** جای داده شده‌اند. بطور کلی، برنامه‌هایی که در این کتاب ایجاد می‌کنید، متشکل از تابع **main** و یک یا چند کلاس که هر یک حاوی اعضای داده و توابع عضو هستند، خواهند بود. اگر شما عضوی از تیم توسعه (طراحی) در یک مجموعه حرفه‌ای هستید، احتمال دارد بر روی سیستم‌های نرم‌افزاری که حاوی صدها، یا هزاران کلاس است، کار کنید. در این فصل، هدف ما توسعه یک چهارچوب کاری ساده خوش‌فرم به لحاظ مهندسی نرم‌افزار به منظور سازماندهی برنامه‌های شی‌گرا در C++ است.

ابتدا توجه خود را به سمت کلاس‌های موجود در دنیای واقعی متمرکز می‌کنیم. سپس به آرامی به سمت دنباله‌ای از هفت برنامه کامل می‌رویم که به توصیف نحوه ایجاد و استفاده از کلاس‌های متعلق به خودمان می‌پردازند. این مثال‌ها با مبحث آموزشی هدفمند که بر توسعه کلاس *grade-book* تمرکز دارد، شروع می‌شود و مربی می‌تواند از آن برای نگهداری امتیازات دانشجو استفاده کند. این مبحث آموزشی در چند فصل آتی بهبود خواهد یافت و در فصل هفتم بحد اعلی خود خواهد رسید.

www.yourshop.4kia.ir

## ۲-۳ کلاس‌ها، شی‌ها، توابع عضو و داده

اجازه دهید بحث را با یک مقایسه ساده که در افزایش درک شما از مطالب بخش ۱۷-۱ موثر است آغاز کنیم. فرض کنید که می‌خواهید با اتومبیلی رانندگی کرده و با فشردن پدال گاز آن را سریعتر به حرکت در آورید. چه اتفاقی قبل از اینکه بتوانید اینکار را انجام دهید، باید رخ دهد؟ بسیار خوب، قبل از اینکه بتوانید با اتومبیلی رانندگی کنید، باید کسی آن را طراحی و ساخته باشد. معمولاً ساخت اتومبیل، با ترسیم یا نقشه‌کشی مهندسی شروع شود. همانند طراحی صورت گرفته برای خانه. این ترسیمات شامل طراحی پدال گاز است که راننده با استفاده از آن سبب می‌شود تا اتومبیل سریعتر حرکت کند. تا حدی، پدال سبب «پنهان» شدن پیچیدگی مکانیزمی می‌شود که اتومبیل را سریعتر بحرکت در می‌آورد، همانطوری که پدال ترمز سبب «پنهان» شدن مکانیزمی می‌شود که از سرعت اتومبیل کم می‌کند، فرمان اتومبیل سبب «پنهان» شدن مکانیزمی می‌شود که اتومبیل را هدایت می‌کند و موارد دیگر. با انجام چنین کارهایی، افراد عادی می‌توانند به آسانی اتومبیل را هدایت کرده و براحتی از پدال گاز، ترمز و فرمان، مکانیزم تعویض دنده و سایر «واسط‌های» کاربرپسند و ساده استفاده کنند تا پیچیدگی مکانیزم‌های داخلی اتومبیل برای راننده مشخص نباشد.

متاسفانه، نمی‌توانید با نقشه‌های ترسیمی یک اتومبیل رانندگی کنید، قبل از اینکه با اتومبیلی رانندگی کنید باید، آن اتومبیل از روی نقشه‌های ترسیمی ساخته شود. یک اتومبیل کاملاً ساخته شده دارای پدال گاز واقعی برای به حرکت درآوردن سریع اتومبیل است. اما این هم کافی نیست، اتومبیل بخودی خود شتاب نمی‌گیرد، از اینرو لازم است راننده بر روی پدال گاز فشار آورد تا به اتومبیل دستور حرکت سریع‌تر را صادر کند.

حال اجازه دهید تا از مثال اتومبیل مطرح شده برای معرفی مفاهیم کلیدی برنامه‌نویسی شی‌گرا در این بخش استفاده کنیم. انجام یک وظیفه در یک برنامه مستلزم یک تابع (همانند **main** که فصل دوم توضیح داده شده) است. تابع، توصیف‌کننده مکانیزمی است که وظیفه واقعی خود را به انجام می‌رساند. تابع پیچیدگی وظایفی که قرار است انجام دهد از دید کاربر خود پنهان می‌سازد، همانند پدال گاز در اتومبیل که پیچیدگی مکانیزم شتاب‌گیری را از دید راننده پنهان می‌کند. در C++، کار را با ایجاد واحدی بنام کلاس که خانه تابع محسوب می‌شود، آغاز می‌کنیم، همانند نقشه ترسیمی اتومبیل که طرح پدال گاز نیز در آن قرار دارد. از بخش ۱۷-۱ بخاطر دارید که به تابع متعلق به یک کلاس، تابع عضو گفته می‌شود. در یک کلاس می‌توان یک یا چند تابع عضو داشت که برای انجام وظایف کلاس طراحی شده‌اند. برای مثال، یک کلاس می‌تواند نشان‌دهنده یک حساب بانکی و حاوی یک تابع عضو برای میزان سپرده در

www.yourshop.4kia.ir

حساب، تابع دیگری برای میزان برداشت پول از حساب و تابع سومی هم برای نمایش میزان پول موجود در حساب باشد.

همانطوری که نمی‌توانید با نقشه ترسیمی اتومبیل رانندگی کنید، نمی‌توانید کلاسی را مشتق کنید. همانطوری که باید قبل از اینکه بتوانید با اتومبیلی رانندگی کنید، شخص از روی نقشه ترسیمی مبادرت به ساخت اتومبیل کرده باشد، شما هم باید قبل از اینکه برنامه بتواند وظایف توصیفی در کلاس را بدست آورد، باید یک شی از کلاس را ایجاد کرده باشید. این یکی از دلایل شناخته شدن C++ بعنوان یک زبان برنامه‌نویسی شی‌گرا است. همچنین توجه نمائید که همانطوری که می‌توان از روی نقشه ترسیمی اتومبیل‌های متعددی ساخت، می‌توان از روی یک کلاس، شیی‌های متعددی ایجاد کرد.

زمانیکه رانندگی می‌کنید، با فشردن پدال گاز، پیغامی به اتومبیل ارسال می‌شود که وظیفه‌ای را انجام دهد، که این وظیفه افزودن سرعت اتومبیل است. به همین ترتیب، پیغام‌هایی را به یک شی ارسال می‌کنیم، هر پیغام بعنوان فراخوان یک تابع عضو شناخته می‌شود و به تابع عضو از شیی اعلان می‌کند که وظیفه خود را انجام دهد. اینکار غالباً بعنوان تقاضای سرویس از یک شی شناخته می‌شود.

تا بدین‌جا، از مقایسه اتومبیل برای توضیح کلاس‌ها، شی‌ها و توابع عضو استفاده کردیم. علاوه بر قابلیت‌های اتومبیل، هر اتومبیلی دارای چندین صفت است، صفاتی همانند رنگ، تعداد درها، ظرفیت باک، سرعت جاری و مسافت طی شده. همانند قابلیت‌های اتومبیل، این صفات هم بعنوان بخشی از طراحی اتومبیل در نقشه ترسیمی دیده می‌شوند. همانطوری که رانندگی می‌کنید، این صفات همیشه در ارتباط با اتومبیل هستند. هر اتومبیلی، حافظ صفات خود است. برای مثال، هر اتومبیلی از میزان سوخت موجود در باک خود مطلع است، اما از میزان سوخت موجود در باک سایر اتومبیل‌ها مطلع نیست. به همین ترتیب، یک شی دارای صفاتی است که به همراه شی بوده و در برنامه بکار گرفته می‌شوند.

این صفات به عنوان بخشی از کلاس شی تصریح می‌شوند. برای مثال، یک شی حساب بانکی دارای صفت موجودی است که نشاندهنده مقدار پول موجود در حساب می‌باشد. هر شی حساب بانکی از میزان موجودی در حساب خود مطلع است، اما از موجودی سایر حساب‌ها در بانک اطلاعی ندارد. صفات توسط اعضای داده کلاس مشخص می‌شوند.

## ۳-۳ نگاهی بر مثال‌های این فصل

www.yourshop.4kia.ir

در مابقی این فصل مبادرت به معرفی هفت مثال ساده می‌کنیم که به توصیف مفاهیم معرفی شده در ارتباط با قیاس اتومبیل می‌پردازند. خلاصه‌ای از این مثال‌ها در زیر آورده شده است. ابتدا مبادرت به ایجاد کلاس **GradeBook** می‌کنیم تا قادر به توصیف این مفاهیم باشیم:

۱- اولین مثال نشاندهنده کلاس **GradeBook** با یک تابع عضو است که فقط یک پیغام خوش‌آمدگویی را در زمان فراخوانی به نمایش در می‌آورد. سپس شما را با نحوه ایجاد یک شی از این کلاس و فراخوانی تابع عضو که پیغام خوش‌آمدگویی را ظاهر می‌سازد، آشنا خواهیم کرد.

۲- دومین مثال، اصلاح شده مثال اول است. در این مثال به تابع عضو اجازه داده می‌شود تا نام دوره را به عنوان یک آرگومان قبول کند. سپس، تابع عضو مبادرت به نمایش نام دوره بعنوان بخشی از پیغام خوش‌آمدگویی می‌کند.

۳- سومین مثال نحوه ذخیره‌سازی نام دوره در یک شی **GradeBook** را نشان می‌دهد. برای این نسخه از کلاس، شما را با نحوه استفاده از توابع عضو به منظور تنظیم نام دوره در شی و بدست آوردن نام دوره از شی آشنا می‌کنیم.

۴- چهارمین مثال به توصیف نحوه مقداردهی اولیه داده در شی **GradeBook** به هنگام ایجاد شی می‌پردازد. مقداردهی اولیه توسط یک تابع عضو بنام سازنده کلاس صورت می‌گیرد. همچنین این مثال نشان می‌دهد که هر شی **GradeBook** از نام دورهٔ خود نگهداری می‌کند.

۵- مثال پنجم اصلاح شده مثال چهارم است که به توصیف نحوه قرار دادن کلاس **GradeBook** در یک فایل مجزا به منظوراستفاده مجدد از نرم‌افزار، می‌پردازد.

۶- مثال ششم، تغییریافته مثال پنجم است که حاوی یک اصل مهندسی نرم‌افزار در جداسازی بخش واسط کلاس از بخش پیاده‌سازی آن است. با انجام اینکار فرآیند اصلاح و تغییر آسانتر می‌شود بدون اینکه در سرویس‌گیرنده‌های کلاس تاثیرگذار باشد.

۷- آخرین مثال قابلیت کلاس **GradeBook** را با معرفی اعتبارسنجی داده افزایش خواهد داد. اعتبارسنجی ما را از فرمت صحیح داده در یک شی مطمئن می‌سازد. برای مثال یک شی **Date** نیازمند یک مقدار در محدودهٔ 1 الی 12 است. در این مثال **GradeBook**، تابع عضوی که مبادرت به تنظیم نام دورهٔ برای یک شی **GradeBook** می‌کند، سبب می‌شود تا نام دوره 25 کاراکتر یا کمتر باشد. اگر طول

www.yourshop.4kia.ir

نام دوره بیش از 25 کاراکتر باشد، تابع عضو فقط از 25 کاراکتر اول استفاده کرده و یک پیغام خطا به نمایش در می‌آورد.

توجه کنید که مثال‌های **GradeBook** در این فصل فرآیند واقعی ذخیره‌سازی را انجام نمی‌دهند. در فصل چهارم، فرآیند پردازش امتیازات را با **GradeBook** و در فصل هفتم مبادرت به ذخیره‌سازی اطلاعات در یک شی **GradeBook** خواهیم کرد.

## ۳-٤ تعریف کلاس با یک تابع عضو

کار را با مثالی آغاز می‌کنیم (شکل ۱-۳) که حاوی کلاس **GradeBook** است و می‌توان از آن برای نگهداری امتیازات دانشجویان استفاده کرد. تابع **main** در خطوط 25-20 مبادرت به ایجاد یک شی **GradeBook** کرده است. این برنامه نسخه اول بوده و نسخه کامل آن در فصل هفتم ایجاد خواهد شد. تابع **main** از این شی و تابع عضو آن برای نمایش پیغام بر روی صفحه نمایش (پیغام خوش‌آمدگویی) استفاده می‌کند.

```
1  // Fig. 3.1: fig03_01.cpp
2  // Define class GradeBook with a member function displayMessage;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // function that displays a welcome message to the GradeBook user
13    void displayMessage()
14    {
15       cout << "Welcome to the Grade Book!" << endl;
16    } // end function displayMessage
17 }; // end class GradeBook
18
19 // function main begins program execution
20 int main()
21 {
22    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
23    myGradeBook.displayMessage(); // call object's displayMessage function
24    return 0; // indicate successful termination
25 } // end main
```

```
Welcome to the Grade Book!
```

**شکل ۱-۳ | تعریف کلاس GradeBook با یک تابع عضو، ایجاد یک شی GradeBook و فراخوانی تابع عضو آن.**

ابتدا به توصیف نحوه تعریف یک کلاس و تابع عضو می‌پردازیم. سپس به توضیح نحوه ایجاد یک شی و نحوه فراخوانی تابع عضو یک شی می‌پردازیم. چند مثال اول حاوی تابع **main** و کلاس **GradeBook** است که از آن در همان فایل استفاده می‌کند. در انتهای این فصل، به معرفی روش‌های

www.yourshop.4kia.ir

خبره در ساختارمند کردن برنامه‌ها به منظور افزایش کارایی و انجام بهتر مهندسی نرم‌افزار خواهیم پرداخت.

***کلاس GradeBook***

قبل از اینکه تابع **main** (خطوط 25-20) بتواند شی از کلاس **GradeBook** ایجاد کند، بایستی به کامپایلر توابع عضو و اعضای داده متعلق به کلاس را اعلان کنیم. این کار بنام تعریف کلاس شناخته می‌شود. تعریف کلاس **GradeBook** (خطوط 17-9) حاوی تابع عضوی بنام **displayMessage** است (خطوط 16-13) که پیغامی بر روی صفحه نمایش چاپ می‌کند (خط 15). بخاطر دارید که یک کلاس همانند یک نقشه ترسیمی است، از اینرو نیاز داریم که یک شی از کلاس **GradeBook** ایجاد کنیم (خط 22)، و تابع عضو **displayMessage** آن را فراخوانی نمایم (خط 23) تا خط 15 را به اجرا درآورده و پیغام خوش‌آمدگویی را به نمایش درآورد. بزودی به توضیح دقیق‌تر خطوط 23-22 خواهیم پرداخت.

تعریف کلاس از خط 9 و با کلمه کلیدی ***class*** و بدنبال آن نام کلاس **GradeBook** آغاز شده است. بطور قراردادی، نام کلاس‌های تعریف شده توسط کاربر با حروف بزرگ شروع می‌شوند و در صورت وجود کلمات بعدی، ابتدای آنها هم با حروف بزرگ آغاز می‌گردد. به این روش غالباً روش *camel case* می‌گویند.

بدنه هر کلاسی در بین جفت براکت باز و بسته (**{** و **}**) محدود می‌شود، همانند خطوط 10 و17. تعریف کلاس با یک سیمکولن خاتمه می‌پذیرد (خط 17).

**خطای برنامه‌نویسی**

*فراموش کردن سیمکولن در انتهای تعریف یک کلاس، خطای نحوی بدنبال خواهد داشت.*

بخاطر دارید که تابع **main** همیشه و بصورت اتوماتیک به هنگام اجرای برنامه فراخوانی می‌شود. اکثر توابع بصورت اتوماتیک فراخوانی نمی‌شوند. همانطوری که مشاهده خواهید کرد، بایستی تابع عضو **displayMessage** را بطور صریح فراخوانی کنید تا وظیفه خود را انجام دهد.

خط 14 حاوی برچسب تصریح‌کننده دسترسی **:public** است. کلمه کلیدی **public**، تصریح‌کننده دسترسی نامیده می‌شود. خطوط 16-13 تعریف‌کننده تابع عضو **displayMessage** هستند. این تابع عضو پس از **:public** قرار دارد تا نشان دهد که تابع «بصورت سراسری در دسترس است»، به این معنی که می‌تواند توسط سایر توابع موجود در برنامه و توسط توابع عضو کلاس‌های دیگر فراخوانی گردد. همیشه در مقابل تصریح‌کننده‌های دسترسی یک کولن (:) قرار داده می‌شود. برای مابقی متن، زمانیکه به

www.yourshop.4kia.ir

تصریح‌کننده دسترسی **public** اشاره می‌کنیم، کولن آن را در نظر نمی‌گیریم. در بخش ۶-۳ به معرفی دومین تصریح‌کننده دسترسی بنام ***private*** خواهیم پرداخت.

هر تابعی در برنامه وظیفه‌ای را انجام داده و امکان دارد مقداری را پس از کامل کردن وظیفه خود برگشت دهد. برای مثال، تابعی می‌تواند یک محاسبه انجام داده و سپس نتیجه آن محاسبه را برگشت دهد. زمانیکه تابعی تعریف می‌شود، باید نوع برگشتی آن را مشخص سازید، تا تابع پس از اتمام کار، آن نوع را برگشت دهد. در خط 13، کلمه کلیدی **void** قرار گرفته است (در سمت چپ نام تابع **displayMessage**) و نشاندهندهٔ نوع برگشتی از سوی تابع است. نوع برگشتی **void** بر این نکته دلالت دارد که **displayMessage** وظیفه‌ای را انجام خواهد داد، اما داده‌ای را به فراخوان خود (در این مثال، تابع فراخوان، **main** می‌باشد) پس از اتمام وظیفه خود برگشت نخواهد داد. (در شکل ۵-۳ مثالی را مشاهده می‌کنید که تابع در آن مقداری را برگشت داده است).

نام تابع عضو **displayMessage** بوده و قبل از آن نوع برگشتی قرار دارد. بطور قراردادی، اسامی توابع با یک حرف کوچک شروع شده و تمام کلمات متعاقب آن با حرف بزرگ شروع می‌شوند. پرانتزهای قرار گرفته پس از نام تابع عضو، بر این نکته دلالت دارند که این یک تابع است. یک جفت پرانتز خالی، همانند خط 13، نشان می‌دهد که این تابع عضو، نیازی به داده اضافی برای انجام وظیفه خود ندارد. در بخش ۵-۳ شاهد تابع عضوی خواهید بود که برای انجام وظیفه خود نیازمند داده اضافی است. معمولاً به خط 13، *سرآیند تابع* گفته می‌شود. بدنه هر تابعی با براکت‌های باز و بسته مشخص می‌شود (**{** و **}**)، همانند خطوط 14 و 16.

بدنه تابع حاوی عباراتی است که وظیفه تابع را به انجام می‌رسانند. در این مورد، تابع عضو displayMessage حاوی یک عبارت است (خط 15) که پیغام "Welcome to the Grade Book!" را به نمایش در می‌آورد. پس از اجرای این عبارت، تابع وظیفه خود را به انجام رسانده است.

**خطای برنامه‌نویسی**

*برگشت دادن مقداری از یک تابع که نوع برگشتی آن بصورت **void** اعلان شده است، خطای کامپایلر بدنبال خواهد داشت.*

**خطای برنامه‌نویسی**

*تعریف تابعی در درون تابع دیگر، خطای نحوی است.*

***تست کلاس GradeBook***

www.yourshop.4kia.ir

در این مرحله می‌خواهیم که از کلاس **GradeBook** در برنامه استفاده کنیم. همانطوری که در فصل دوم آموختید، تابع **main** با اجرای هر برنامه‌ای شروع بکار می‌کند. خطوط 25-20 از برنامه شکل ۱-۳ حاوی تابع **main** هستند، که اجرای برنامه را تحت کنترل دارد.

در این برنامه، مایل هستیم تابع عضو **displayMessage** از کلاس **GradeBook** برای نمایش پیغام خوش‌آمدگویی اجرا گردد (فراخوانی شود). اصولاً تا زمانیکه شیی از یک کلاس ایجاد نکرده باشید، نمی‌توانید تابع عضو کلاس را فراخوانی نمائید (در بخش ۷-۱۰ با تابع عضو **static**، آشنا خواهید شد که در این مورد یک استثناء می‌باشد.) خط 22 یک شی از کلاس **GradeBook** بنام **myGradeBook** ایجاد می‌کند. دقت کنید که نوع متغیر **GradeBook** است، کلاس تعریف شده در خطوط 17-9. زمانیکه متغیرهایی از نوع **int** اعلان می‌کنیم، همانند کاری که در فصل دوم انجام دادیم، کامپایلر از مفهوم **int** مطلع است و می‌داند که آن یک نوع بنیادین می‌باشد. با این وجود، زمانیکه مبادرت به نوشتن خط 22 می‌کنیم، کامپایلر بصورت اتوماتیک از مفهوم نوع **GradeBook** اطلاعی ندارد و آن را نمی‌شناسد، این نوع یک نوع تعریف شده توسط کاربر است. از اینرو، باید به کامپایلر، با قرار دادن تعریف کلاس همانند کاری که در خطوط 17-9 انجام داده‌ایم، **GradeBook** را معرفی کنیم. اگر این خطوط را حذف کنیم، کامپایلر پیغام خطایی همانند "*'GradeBook' : undeclared identifier*" را در GNU C++ صادر می‌کند. هر کلاس جدیدی که ایجاد می‌کنید، تبدیل به یک نوع جدید می‌شود که می‌تواند برای ایجاد شی‌ها بکار گرفته شود. برنامه‌نویسان می‌توانند در صورت نیاز مبادرت به تعریف کلاس‌های با نوع جدید نمایند، و این یکی از دلایل شناخته شدن C++ بعنوان یک زبان بسط‌پذیر است.

خط 23 مبادرت به فراخوانی تابع عضو **displayMessage** (تعریف شده در خطوط 16-13) با استفاده از متغیر **myGradeBook** و بدنبال آن عملگر نقطه (.)، سپس نام تابع **displayMessage** و یک جفت پرانتز خالی می‌کند. این فراخوانی موجب می‌شود که تابع **displayMessage** برای انجام وظیفه خود فراخوانی شود. در ابتدای خط 23، عبارت "**myGradeBook.**" بر این نکته دلالت دارد که **main** بایستی از شی **GradeBook** که در خط 22 ایجاد شده است، استفاده نماید. پرانتزهای خالی در خط 13 نشان می‌دهند که تابع عضو **displayMessage** نیازی به داده اضافی برای انجام وظیفه خود ندارد. (در بخش ۵-۳، با نحوه ارسال داده به تابع آشنا خواهید شد) زمانیکه **displayMessage** وظیفه خود را انجام داد، تابع **main** به اجرای خود از خط 24 ادامه خواهد داد، که نشان می‌دهد **main** وظیفه خود را به بدرستی انجام داده است. با رسیدن به انتهای **main**، برنامه خاتمه می‌پذیرد.

***دیاگرام کلاس UML برای کلاس GradeBook***

www.yourshop.4kia.ir

از بخش ۱۷-۱ بخاطر دارید که UML یک زبان گرافیکی بکار رفته توسط برنامه‌نویس برای نمایش سیستم‌های شی‌گرا به یک روش استاندارد است. در UML، هر کلاس در یک دیاگرام کلاس و بصورت یک مستطیل با سه قسمت (بخش) مدل‌سازی می‌شود. شکل ۲-۳ نشاندهنده یک دیاگرام کلاس UML برای کلاس **GradeBook** معرفی شده در برنامه ۱-۳ است. بخش فوقانی حاوی نام کلاس است، که در وسط قرار گرفته و بصورت توپر نوشته می‌شود. بخش میانی، حاوی صفات کلاس است که مرتبط با اعضای داده در C++ است. در شکل ۲-۳ بخش میانی خالی است، چرا که این نسخه از کلاس **GradeBook** در برنامه ۱-۳ دارای صفات نیست. (در بخش ۶-۳ نسخه‌ای از کلاس **GradeBook** عرضه شده که دارای یک صفت است.) بخش تحتانی حاوی عملیات کلاس است، که متناظر با توابع عضو در C++ می‌باشد.

**شکل ۲-۳ | دیاگرام کلاس UML نشان می‌دهد که کلاس GradeBook دارای یک عملیات سراسری بنام displayMessage است.**

UML مبادرت به مدلسازی عملیات‌ها با لیست کردن نام عملیات و بدنبال آن مجموعه پرانتزها می‌کند. کلاس **GradeBook** فقط دارای یک تابع عضو بنام **displayMessage** است، از اینرو بخش تحتانی در شکل ۲-۳ فقط یک عملیات با این نام را لیست کرده است. تابع عضو **displayMessage** برای انجام وظیفه خود نیازی به اطلاعات اضافی ندارد، از اینرو پرانتزهای قرار گرفته پس از **displayMessage** در این دیاگرام کلاس خالی هستند، همانند سرآیند تابع عضو قرار گرفته در خط 13 برنامه۱-۳. علامت جمع (+) که قبل از نام عملیات آورده شده، نشان می‌دهد که **displayMessage** یک عملیات سراسری در UML است (یک تابع عضو **public** در C++). به دفعات از دیاگرام‌های کلاس UML برای خلاصه کردن صفات و عملیات کلاس‌ها استفاده خواهیم کرد.

## ۳-۵ تعریف تابع عضو با پارامتر

در مثال قیاس اتومبیل در بخش ۲-۳ خاطرنشان کردیم که فشردن پدال گاز سبب ارسال پیغامی به اتومبیل می‌شود تا وظیفه‌ای را به انجام برساند، که در این مورد افزودن سرعت اتومبیل است. اما اتومبیل باید چقدر سرعت بگیرد؟ همانطوری که می‌دانید، با فشردن هر چه بیشتر پدال، سرعت اتومبیل افزایش پیدا می‌کند. بنابر این پیغام ارسالی به اتومبیل هم حاوی وظیفه بوده و همچنین حاوی اطلاعات دیگری است که به اتومبیل در انجام وظیفه کمک می‌کند. این اطلاعات اضافی بنام پارامتر شناخته می‌شوند، مقدار پارامتر به اتومبیل کمک می‌کند تا میزان افزایش سرعت را تعیین کند. به همین ترتیب، یک تابع عضو می‌تواند نیازمند یک یا چندین پارامتر بعنوان داده اضافی برای انجام وظیفه خود باشد. فراخوان تابع مقادیری بنام آرگومان، برای هر پارامتر تابع تدارک می‌بیند. برای مثال، در سپرده‌گذاری در حساب

www.yourshop.4kia.ir

بانکی، فرض کنید تابع عضو **deposit** از کلاس **Account** پارامتری که نشاندهنده مقدار سپرده است، مشخص کرده باشد. زمانیکه تابع عضو **deposit** فراخوانی شود، مقدار آرگومان که نشاندهنده مقدار سپرده است به پارامتر تابع عضو کپی می‌شود. سپس تابع عضو این مقدار سپرده را به میزان موجودی اضافه می‌کند.

***تعریف و تست کلاس GradeBook***

مثال بعدی (برنامه شکل ۳-۳) تعریف مجددی از کلاس **GradeBook** (خطوط 23-14) با تابع عضو **displayMessage** است (خطوط 22-18) که نام دوره را بعنوان بخشی از رشته خوش‌آمدگویی چاپ می‌کند. تابع عضو جدید **displayMessage** مستلزم یک پارامتر است (**courseName** در خط 18) که نشاندهنده نام دوره است.

قبل از اینکه به بررسی ویژگی‌های جدید کلاس **GradeBook** بپردازیم، اجازه دهید به نحوه استفاده از کلاس جدید در **main** نگاهی داشته باشیم (خطوط 40-26). در خط 28 یک متغیر از نوع رشته (***string***) بنام **nameOfCourse** ایجاد شده است که از آن برای ذخیره کردن نام دوره وارد شده توسط کاربر استفاده خواهیم کرد. متغیر از نوع **string** نشاندهنده رشته‌ای از کاراکترها همانند "CS101 Introduction to C++ Programming" است. نوع **string** در واقع شی از کلاس **string** کتابخانه استاندارد C++ است. این کلاس در فایل سرآیند <**string**> تعریف شده و نام **string** همانند **cout**، متعلق به فضای‌نامی **std** است. برای اینکه خط 28 قادر به کامپایل شدن باشد، خط 9 حاوی سرآیند <**string**> است. دقت کنید که اعلان **using** در خط 10 به ما اجازه می‌دهد تا به آسانی در خط 28 از **string** بجای **std::string** استفاده کنیم. برای این لحظه، می‌توانید در مورد متغیرهای **string** همانند متغیرهای از نوع دیگر همانند **int** فکر کنید. در بخش ۱۰-۳ با قابلیت‌های **string** بیشتر آشنا خواهید شد.

خط 29 یک شی از کلاس **GradeBook** بنام **myGradeBook** ایجاد می‌کند. خط 32 به کاربر اعلان می‌کند که نام دوره را وارد سازد. خط 33 مبادرت به خواندن نام دوره از ورودی کاربر کرده و آن را با استفاده از تابع کتابخانه‌ای **getline** به متغیر **nameOfCourse** تخصیص می‌دهد. قبل از اینکه به توضیح این خط از کد بپردازیم، اجازه دهید تا توضیح دهیم چرا نمی‌توانیم برای بدست آوردن نام دوره فقط بنویسیم

```
cin>>nameOfCourse;
```

www.yourshop.4kia.ir

در اجرای نمونه این برنامه، ما از نام دوره "CS101 Introduction to C++ Programming" استفاده کرده‌ایم که حاوی چندین کلمه است. زمانیکه **cin** با عملگر استخراج به کار گرفته می‌شود، مبادرت به خواندن کاراکترها تا رسیدن به اولین کاراکتر *white-space* می‌کند. از اینرو فقط "CS101" توسط این عبارت خوانده می‌شود و برای خواندن مابقی نام دوره مجبور به عملیات ورودی اضافی خواهیم بود.

در این مثال، مایل هستیم تا کاربر نام کامل دوره را تایپ کرده و کلید *Enter* را برای تحویل آن به برنامه فشار دهد، و کل نام دوره در متغیر رشته‌ای **nameOfCourse** ذخیره گردد فراخوانی تابع **getline(**cin, nameOfCourse**)** در خط 33، سبب می‌شود که کاراکترها (از جمله کاراکترهای فاصله که کلمات را از هم در ورودی جدا می‌کنند) از شی ورودی استاندارد جریان **cin** (صفحه کلید) تا رسیدن به کاراکتر خط جدید خوانده شده و در متغیر رشته‌ای **nameOfCourse** جای داده شده و کاراکتر خط جدید حذف می‌گردد. توجه نمائید زمانیکه کلید *Enter* فشرده می‌شود (در زمان تایپ ورودی)، یک خط جدید (*newline*) به جریان ورودی افزوده می‌شود. همچنین توجه کنید که فایل سرآیند <**string**> باید قرار داشته باشد تا بتوان از تابع **getline** استفاده کرد. نام این تابع متعلق به فضای نامی **std** است.

```
1  // Fig. 3.3: fig03_03.cpp
2  // Define class GradeBook with a member function that takes a parameter;
3  // Create a GradeBook object and call its displayMessage function.
4  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <string> // program uses C++ standard string class
10 using std::string;
11 using std::getline;
12
13 // GradeBook class definition
14 class GradeBook
15 {
16 public:
17    // function that displays a welcome message to the GradeBook user
18    void displayMessage( string courseName )
19    {
20       cout << "Welcome to the grade book for\n" << courseName << "!"
21          << endl;
22    } // end function displayMessage
23 }; // end class GradeBook
24
25 // function main begins program execution
26 int main()
27 {
28    string nameOfCourse; // string of characters to store the course name
29    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
30
31    // prompt for and input course name
32    cout << "Please enter the course name:" << endl;
```

www.yourshop.4kia.ir

```
33    getline( cin, nameOfCourse ); // read a course name with blanks
34    cout << endl; // output a blank line
35
36    // call myGradeBook's displayMessage function
37    // and pass nameOfCourse as an argument
38    myGradeBook.displayMessage( nameOfCourse );
39    return 0; // indicate successful termination
40 } // end main
```

```
Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

**شکل ۳-۳ | تعریف کلاس GradeBook با تابع عضوی که پارامتر دریافت می‌کند.**

خط 38 تابع عضو **displayMessage** شی **myGradeBook** را فراخوانی می‌کند. **متغیر nameOfCource** موجود در درون پرانتزها آرگومانی است که به تابع **displayMessage** ارسال می‌شود، از اینروست که تابع می‌تواند وظیفه خود را به انجام رساند. مقدار متغیر **nameOfCourse** در **main** تبدیل به مقدار پارامتر **courseName** تابع عضو **displayMessage** در خط 18 می‌شود. زمانیکه این برنامه اجرا شود، توجه نمائید که تابع عضو **displayMessage** به همراه پیغام خوش‌آمدگویی، نام دورهٔ تایپ شده توسط کاربر را به نمایش درمی‌آورد.

### *آرگومان‌ها و پارامترهای بیشتر*

برای تصریح اینکه تابعی نی‌ازمند داده برای انجام وظایف خویش است، اطلاعات اضافی را در لیست پارامتری تابع قرار می‌دهیم، لیستی که در درون پرانتزها قرار گرفته پس از نام تابع جای دارد. لیست پارامترها می‌تواند به هر تعداد پارامتر داشته باشد یا اصلاً پارامتری نداشته باشد (همانند پرانتزهای خالی در خط 13 برنامه شکل ۱-۳) تا نشان دهد تابع نیاز به هیچ پارامتری ندارد. لیست پارامتری تابع عضو **displayMessage** به نحوی اعلان شده که تابع نیازمند یک پارامتر است (شکل ۳-۳، خط 18). هر پارامتر باید دارای نوع و یک شناسه باشد. در این مورد، نوع **string** بوده و شناسه **courseName** است و نشان می‌دهد که تابع عضو **displayMessage** مستلزم یک رشته برای انجام وظیفه خویش است. بدنه تابع عضو از پارامتر **courseName** برای دسترسی به مقدار ارسالی به تابع در فراخوانی تابع (خط 38 در **main**) استفاده می‌کند. خطوط 21-20 مقدار پارامتر **courseName** را بعنوان بخشی از پیغام خوش‌آمدگویی به نمایش در می‌آورد. توجه نمائید که نام متغیر پارامتری (خط 18) می‌تواند همنام متغیر آرگومان (خط 38) یا نام متفاوتی داشته باشد. در فصل ششم با دلایل اینکارها آشنا خواهید شد.

یک تابع می‌تواند دارای چندین پارامتر باشد، بشرطی که هر پارامتر از دیگری با یک ویرگول مجزا شده باشد (در مثال شکل‌های ۴-۶ و ۵-۶ شاهد آنها خواهید بود). تعداد و ترتیب آرگومان‌ها در یک تابع فراخوانی شده بایستی با تعداد و ترتیب لیست پارامترها در سرآیند تابع عضو فراخوانی شده مطابقت داشته

www.yourshop.4kia.ir

باشد. همچنین، نوع آرگومان‌ها در تابع فراخوانی شده باید با نوع پارامترهای متناظر در سرآیند تابع مطابقت داشته باشد. (همانطوری که جلوتر می‌رویم، شاهد خواهید بود که نوع یک آرگومان می‌تواند همانند نوع پارامتر متناظر خود نباشد.) در مثال ما، یک آرگومان رشته‌ای در تابع فراخوانی شده وجود دارد (**nameOfCourse**) که دقیقاً با پارامتری رشته‌ای در تعریف تابع عضو مطابقت دارد (**CourseName**).

**خطای برنامه‌نویسی**

*قراردادن یک سیمکولن پس از پرانتز سمت راست در لیست پارامتری تعریف تابع، خطای نحوی است.*

**خطای برنامه‌نویسی**

*تعریف یک پارامتر تابع همانند یک متغیر محلی در تابع، خطای کامپایل بدنبال خواهد داشت.*

**برنامه‌نویسی ایده‌ال**

*از ابهام اجتناب کنید، از اسامی یکسان برای آرگومان‌های ارسالی به تابع و پارامترهای متناظر در تعریف تابع، استفاده نکنید.*

**برنامه‌نویسی ایده‌ال**

*برای توابع و پارامترها اسامی با معنی انتخاب کنید تا خوانایی برنامه افزایش یافته و نیاز به افزودن توضیحات را کم نماید.*

### *به روز کردن دیاگرام کلاس UML برای کلاس GradeBook*

دیاگرام کلاس UML در شکل ۴-۳ مبادرت به مدلسازی کلاس **GradeBook** از برنامه ۳-۳ کرده است. همانند کلاس **GradeBook** تعریف شده در برنامه ۱-۳، این کلاس **GradeBook** حاوی یک تابع عضو سراسری **displayMessage** است. با این وجود، این نسخه از **displayMessage** دارای یک پارامتر می‌باشد. UML پارامتر را با لیست کردن نام پارامتر، بدنبال آن یک کولن و سپس نوع پارامتر در درون پرانتزهای قرار گرفته پس از نام عملیات مدل‌سازی می‌کند. UML دارای نوع داده‌های متعلق بخود شبیه C++ است. UML یک زبان مستقل بوده و از آن در کنار انواع زبان‌های برنامه‌نویسی استفاده می‌شود، از اینرو اصطلاحات تخصصی آن دقیقاً مطابق با C++ نیست. برای مثال، نوع رشته (**String**) در UML متناظر با نوع **string** در C++ است. تابع عضو **displayMessage** از کلاس **GradeBook** (شکل ۳-۳ خطوط 22-18) دارای یک پارامتر رشته‌ای بنام **CourseName** است، از اینرو در شکل ۴-۳ بصورت **courseName : String** در میان پرانتزها و پس از نام **diaplayName**، قرار گرفته است. دقت کنید که این نسخه از کلاس **GradeBook** هنوز دارای اعضای داده نیست.

www.yourshop.4kia.ir

**شکل ٤-٣ | دیاگرام کلاس UML نشان می‌دهد که کلاس GradeBook دارای عملیات displayMessage با یک پارامتر از نوع String در UML است.**

## ٦-٣ اعضای داده، توابع set و get

در فصل دوم، تمام متغیرهای برنامه را در تابع **main** اعلان کردیم. متغیرهای اعلان شده در بدنه تعریف یک تابع، بنام متغیرهای محلی شناخته می‌شوند و می‌توانند فقط از خطی که در آن تابع اعلان شده‌اند تا رسیدن به براکت بستن سمت راست (}) در تعریف تابع بکار گرفته شوند. قبل از اینکه بتوان از یک متغیر محلی استفاده کرد، باید ابتدا آن را اعلان کرده باشید. زمانیکه تابع به کار خود خاتمه می‌دهد، مقادیر متغیرهای محلی آن از بین می‌روند (در فصل ششم با متغیرهای محلی **static** آشنا خواهید شد که از این قاعده مستثنی هستند). از بخش ۲-۳ بخاطر دارید که شیی که دارای صفت است، آن را با خود همراه دارد و در برنامه از آن استفاده می‌شود. چنین صفاتی در کل طول عمر یک شی وجود خواهند داشت.

معمولاً کلاسی که دارای یک یا چندین تابع عضو است، مبادرت به دستکاری کردن صفات متعلق به یک شی مشخص از کلاسی می‌کند. صفات به عنوان متغیرهای در تعریف کلاس عرضه می‌شوند. چنین متغیرهای، بنام اعضای داده شناخته می‌شوند و در درون تعریف کلاس اعلان می‌گردند اما در خارج از تعریف بدنه تابع عضو کلاس جای می‌گیرند. هر شی از کلاس مسئول نگهداری کپی از صفات متعلق به خود در حافظه است. مثال مطرح شده در این بخش به توصیف کلاس **GradeBook** می‌پردازد که حاوی یک عضو داده **CourseName** است تا نشان‌دهنده نام یک دوره خاص از شی **GradeBook** باشد.

### *کلاس GradeBook با یک عضو داده، یک تابع set و get*

در این مثال، کلاس **GradeBook** (برنامه شکل ۵-۳) مبادرت به نگهداری نام دوره بصورت یک عضو داده می‌کند و از اینروست که می‌تواند بکار گرفته شده یا در هر زمان اجرای برنامه آن را اصلاح کرد. کلاس حاوی توابع عضو **setCourseName**، **getCourseName** و **displayMessage** است. تابع عضو **setCourseName** نام دوره را در عضو داده **GradeBook** ذخیره می‌سازد. تابع عضو **getCourseName** نام دوره را از عضو داده بدست می‌آورد. تابع عضو **displayMessage** که هیچ پارامتری ندارد، پیغام خوش‌آمدگویی را که شامل نام دوره است، به نمایش در می‌آورد. با این همه، همانطوری که مشاهده می‌کنید، در حال حاضر تابع، نام دوره را با فراخوانی تابع دیگری در همان کلاس بدست می‌آورد، تابع **getCourseName**.

```
// Fig. 3.5: fig03_05.cpp
// Define class GradeBook that contains a courseName data member
// and member functions to set and get its value;
// Create and manipulate a GradeBook object.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
```

www.yourshop.4kia.ir

```
9
10 #include <string> // program uses C++ standard string class
11 using std::string;
12 using std::getline;
13
14 // GradeBook class definition
15 class GradeBook
16 {
17 public:
18    // function that sets the course name
19    void setCourseName( string name )
20    {
21       courseName = name; // store the course name in the object
22    } // end function setCourseName
23
24    // function that gets the course name
25    string getCourseName()
26    {
27       return courseName; // return the object's courseName
28    } // end function getCourseName
29
30    // function that displays a welcome message
31    void displayMessage()
32    {
33       // this statement calls getCourseName to get the
34       // name of the course this GradeBook represents
35      cout << "Welcome to the grade book for\n" << getCourseName() << "!"
36          << endl;
37    } // end function displayMessage
38 private:
39    string courseName; // course name for this GradeBook
40 }; // end class GradeBook
41
42 // function main begins program execution
43 int main()
44 {
45    string nameOfCourse; // string of characters to store the course name
46    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
47
48    // display initial value of courseName
49    cout << "Initial course name is: " << myGradeBook.getCourseName()
50       << endl;
51
52    // prompt for, input and set course name
53    cout << "\nPlease enter the course name:" << endl;
54    getline( cin, nameOfCourse ); // read a course name with blanks
55    myGradeBook.setCourseName( nameOfCourse ); // set the course name
56
57    cout << endl; // outputs a blank line
58    myGradeBook.displayMessage(); // display message with new course name
59    return 0; // indicate successful termination
60 } // end main
```

```
Initial course name is:

Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

www.yourshop.4kia.ir

**شکل ۵-۳ | تعریف و تست کلاس GradeBook با یک عضو داده و توابع set و get.**

**برنامه‌نویسی ایده‌ال**

*یک خط خالی مابین تعریف تابع عضو قرار دهید تا خوانایی برنامه افزایش پیدا کند.*

معمولاً یک مدرس بیش از یک دوره را تدریس می‌کند که هر دوره دارای نام متعلق بخود است. در خط 39، متغیر **CourseName** از نوع رشته اعلان شده است. بدلیل اینکه متغیر اعلان شده در درون تعریف کلاس اعلان شده (خطوط 40-15) اما در خارج از بدنه تعاریف توابع عضو جای دارد (خطوط 22-19، 28-25 و 37-31)، خط 39 اعلانی برای یک عضو داده است. هر نمونه‌ای (شی) از کلاس **GradeBook** حاوی یک کپی از هر عضو داده کلاس خواهد بود. برای مثال، اگر دو شی **GradeBook** وجود داشته باشد، هر شی دارای یک کپی از **CourseName** متعلق بخود خواهد بود، همانطوری که در مثال برنامه شکل ۷-۳ شاهد آن خواهید بود. مزیت ایجاد **CourseName** بصورت یک عضو داده در این است که تمام توابع عضو یک کلاس (در این مورد، **GradBook**) می‌توانند در هر عضو داده موجود که در تعریف کلاس وجود دارد، دستکاری کنند (در این مورد، **CourseName**).

***تصریح‌کننده دسترسی public و private***

اکثر اعلان‌های اعضای داده پس از برچسب تصریح‌کننده دسترسی **private:** ظاهر می‌شوند (خط 38). همانند **public**، کلمه کلیدی **private** یک تصریح‌کننده است. متغیرها یا توابع اعلان شده پس از **private** (و قبل از تصریح‌کننده دسترسی بعدی) فقط در توابع عضو کلاسی که در آن اعلان شده‌اند، در دسترس خواهند بود. بنابر این، عضو داده **CourseName** فقط می‌تواند در توابع عضو **setCourseName**، **getCourseName** و **displayMessage** از کلاس **GradeBook** بکار گرفته شود. بدلیل اینکه عضو داده **CourseName** بصورت **private** اعلان شده است، نمی‌تواند توسط توابع خارج از کلاس (همانند **main**) یا توابع عضو کلاس‌های دیگر در برنامه بکار گرفته شود. مبادرت به دسترسی عضو داده **courseName** در یکی از نقاط این برنامه با عبارتی همانند **myGradeBook.courseName** خطای کامپایل تولید کرده و پیغامی مشابه

```
cannot access private member declared in class 'GradeBook'
```

به نمایش در می‌آید.

**مهندسی نرم‌افزار**

*بعنوان یک قانون، اعضای داده بایستی بصورت **private** و توابع عضو بصورت **public** اعلان شوند. (مشاهده خواهید کرد در صورتیکه توابع عضو خاصی که فقط از طریق توابع عضو دیگر موجود در کلاس در دسترس قرار می‌گیرند، اگر بصورت **private** اعلان شوند، مناسب خواهند بود.)*

www.yourshop.4kia.ir

**خطای برنامه‌نویسی**

*اقدام یک تابع، که عضوی از یک کلاس خاص نیست، به دسترسی به یک عضو **private** آن کلاس، خطای کامپایل بدنبال خواهد داشت.*

دسترسی پیش‌فرض برای اعضای کلاس حالت **private** است، از اینرو تمام اعضای قرار گرفته پس از سرآیند و قبل از اولین تصریح‌کننده دسترسی بصورت **private** خواهند بود. امکان دارد تصریح‌کننده‌های دسترسی **public** و **private** تکرار شوند، اما اینکار ضرورتی ندارد و می‌تواند سبب سردرگمی شود.

**برنامه‌نویسی ایده‌ال**

*علیرغم این واقعیت که تصریح‌کننده‌های دسترسی **public** و **private** می‌توانند تکرار شده و جابجا نوشته شوند، اما سعی کنید تمام لیست عضوهای **public** یک کلاس را در یک گروه و تمام عضوهای **private** را در گروه دیگری قرار دهید. در این حالت توجه سرویس‌گیرنده بر روی واسط **public** کلاس متمرکز می‌شود، بجای اینکه توجه آن به پیاده‌سازی کلاس معطوف شود.*

**برنامه‌نویسی ایده‌ال**

*اگر می‌خواهید لیست اعضای **private** را در ابتدای تعریف کلاس قرار دهید، بصورت صریح از کلمه **private** استفاده کنید علیرغم اینکه **private** انتخاب پیش‌فرض است. در اینحالت وضوح برنامه افزایش می‌یابد.*

اعلان اعضای داده با تصریح‌کننده دسترسی **private**، بعنوان پنهان‌سازی داده شناخته می‌شود. زمانیکه برنامه مبادرت به ایجاد (نمونه‌سازی) از یک شی کلاس **GradeBook** می‌کند، عضو داده **courseName** در شی کپسوله (پنهان) شده و فقط می‌تواند توسط توابع عضو آن کلاس در دسترس قرار گیرد. در کلاس **GradeBook**، توابع عضو **setCourseName** و **getCourseName** مبادرت به دستکاری مستقیم عضو داده **courseName** می‌کنند (اگر لازم باشد **diaplayMessage** هم می‌تواند این کار را انجام دهد).

**مهندسی نرم‌افزار**

*در فصل دهم خواهید آموخت که توابع و کلاس‌های اعلان شده توسط یک کلاس حالت دوست **(friend)** پیدا کرده و می‌توانند به اعضای **private** کلاس دسترسی پیدا کنند.*

**اجتناب از خطا**

*با ایجاد اعضای داده کلاس بصورت **private** و توابع عضو کلاس بصورت **public**، کار خطایابی آسانتر می‌شود چرا که دستکاری‌کننده‌های داده محلی هستند.*

***توابع عضو setCourseName و getCourseName***

تابع عضو **setCourseName** تعریف شده در خطوط 22-19 به هنگام اتمام وظیفه خود هیچ مقداری برگشت نمی‌دهد و از اینرو نوع برگشتی آن **void** است. تابع عضو یک پارامتر دریافت می‌کند، **name**، که نشان‌دهنده نام دوره‌ای است که به آن بعنوان یک آرگومان ارسال خواهد شد (خط 55 از **main**).

www.yourshop.4kia.ir

خط 21 مبادرت به تخصیص **name** به عضو داده **courseName** می‌کند. در این مثال، **setCourseName** مبادرت به اعتبارسنجی نام دوره نمی‌کند (بررسی فرمت نام دوره با یک الگوی خاص). برای نمونه فرض کنید که دانشگاهی می‌تواند از اسامی دوره تا 25 کاراکتر یا کمتر چاپ بگیرد. در این مورد، مایل هستیم کلاس **GradeBook** ما را مطمئن سازد که عضو داده **courseName** هرگز بیش از 25 کاراکتر نداشته باشد. در بخش ۱۰-۳ با تکنیک‌های اولیه اعتبارسنجی آشنا خواهید شد.

تابع عضو **getCourseName** تعریف شده در خطوط 28-25 یک شی خاص از **GradeBook** بنام **courseName** را برگشت می‌دهد. تابع عضو دارای یک لیست پارامتری خالی است و از اینرو برای انجام وظیفه خود نیازی به اطلاعات اضافی ندارد. تابع مشخص می‌سازد که یک رشته برگشت خواهد داد. زمانیکه تابعی که نوع برگشتی آن بجز **void** است فراخوانی می‌شود و وظیفه خود را انجام می‌دهد، تابع نتیجه‌ای را به فراخوان خود برگشت می‌دهد. برای مثال، هنگامی که به سراغ یک ATM می‌روید و تقاضای نمایش میزان موجودی خود را می‌کنید، انتظار دارید ATM مقداری که نشاندهنده میزان موجودی است به شما برگشت دهد. به همین ترتیب، هنگامی که عبارتی تابع عضو **getCourseName** از یک شی **GradeBook** را فراخوانی می‌کند، عبارت انتظار دریافت نام دوره را دارد (در این مورد، یک رشته است که توسط نوع برگشتی تابع مشخص گردیده است). اگر تابعی بنام **square** داشته باشید که مربع آرگومان خود را برگشت می‌دهد، عبارت

```
int results = square(2);
```

مقدار **4** را از تابع **square** برگشت داده و متغیر **result** را با **4** مقداردهی اولیه می‌کند. اگر تابعی بنام **maximum** داشته باشید که بزرگترین عدد را از بین سه آرگومان خود برگشت می‌دهد، عبارت

```
int biggest = maximum(27,114,41);
```

عدد **114** را از تابع **maximum** برگشت داده و متغیر **biggest** با عدد **114** مقداردهی اولیه می‌شود.

**خطای برنامه‌نویسی**

*فراموش کردن مقدار برگشتی از تابعی که مقدار برگشتی برای آن در نظر گرفته شده است، خطای کامپایل بدنبال خواهد داشت.*

توجه کنید که عبارات قرار گرفته در خطوط 21 و 27 هر یک از متغیر **courseName** (خط 39) استفاده کرده‌اند، بدون اینکه این متغیر در توابع عضو اعلان شده باشد می‌توانیم از **courseName** در توابع عضو کلاس **GradeBook** استفاده کنیم چرا که **courseName** یک عضو داده از کلاس است. همچنین

www.yourshop.4kia.ir

توجه نمائید که ترتیب تعریف توابع عضو تعیین‌کننده ترتیب فراخوانی در زمان اجرا نیستند از اینرو تابع عضو **getCourseName** می‌تواند قبل از تابع عضو **setCourseName** تعریف شود.

***تابع عضو displayMessage***

تابع عضو **displayMessage** در خطوط 37-31 هیچ نوع داده‌ای را پس از کامل کردن وظیفه خود برگشت نمی‌دهد، از اینروست که نوع برگشتی آن **void** تعریف شده است. تابع، پارامتری دریافت نمی‌کند، بنابر این لیست پارامتری آن خالی است. خطوط 36-35 پیغام خوش‌آمدگویی را که حاوی مقدار عضو داده **courseName** است در خروجی چاپ می‌کنند. خط 35 مبادرت به فراخوانی تابع **getCourseName** برای بدست آوردن مقداری از **courseName** می‌کند. توجه نمائید که تابع عضو **displayMessage** هم قادر به دسترسی مستقیم به عضو داده **courseName** است. در مورد اینکه چرا از فراخوانی تابع عضو **getCourseName** برای بدست آوردن مقدار **courseName** استفاد کرده‌ایم، توضیح خواهیم داد.

***تست کلاس GradeBook***

تابع **main** (خطوط 60-43) یک شی از کلاس **GradeBook** ایجاد کرده و از توابع عضو آن استفاده می‌کند. در خط 46 یک شی **GradeBook** بنام **myGradeBook** ایجاد شده است. خطوط 50-49 نام دوره اولیه را با فراخوانی تابع عضو **getCourseName** به نمایش در می‌آورند. توجه کنید که اولین خط خروجی نشاندهنده نام دوره نمی‌باشد، چرا که عضو داده **courseName** در ابتدای کار تهی است. بطور پیش‌فرض، مقدار اولیه یک رشته، رشته تهی است، رشته‌ای که حاوی هیچ کاراکتری نمی‌باشد. زمانیکه یک رشته تهی به نمایش درآید، چیزی بر روی صفحه نمایش ظاهر نمیشود.

خط 53 از کاربر می‌خواهد که نام دوره را وارد سازد. متغیر محلی **nameOfCourse** اعلان شده در خط 45، با نام دوره وارد شده توسط کاربر مقداردهی می‌شود که از فراخوانی تابع **getline** بدست می‌آید (خط 54). خط 55 تابع عضو **setCourseName** را فراخوانی کرده و **nameOfCourse** را بعنوان آرگومان تابع بکار می‌گیرد. به هنگام فراخوانی تابع، مقدار آرگومان به پارامتر **name** (خط 19) از تابع عضو **setCourseName** کپی می‌گردد (خطوط 22-19). سپس مقدار پارامتر به عضو داده **courseName** تخصیص می‌یابد (خط 21). خط 57 یک خط خالی در خروجی قرار داده، سپس خط 58 تابع عضو **displayMessage** را برای نمایش پیغام خوش‌آمدگویی به همراه نام دوره فراخوانی می‌کند.

***مهندسی نرم‌افزار با توابع set و get***

اعضای داده **private** یک کلاس فقط قادر به دستکاری شدن از طریق توابع عضو آن کلاس هستند (و «دوستان» آن کلاس که در فصل دهم با آنها آشنا خواهید شد). بنابر این سرویس‌گیرنده یک شی،

www.yourshop.4kia.ir

(یعنی هر کلاس یا تابعی که مبادرت به فراخوانی توابع عضو از خارج شی می‌کند)، توابع عضو **public** کلاس را برای دریافت سرویس‌های کلاس برای شی‌های خاصی از کلاس فراخوانی می‌کند. به همین دلیل است که عبارات موجود در تابع **main** (برنامه شکل ۵-۳، خطوط 60-43) مبادرت به فراخوانی توابع عضو **setCourseName**، **getCourseName** و **displayMessage** موجود در شی **GradeBook** می‌کند. غالباً توابع عضو **public** کلاس‌ها به سرویس‌گیرنده‌ها اجازه تخصیص مقادیر به (set) یا دریافت مقادیر از (get) اعضای داده **private** را می‌دهند. نیازی نیست که اسامی این توابع عضو حتماً با set یا get شروع شوند، اما این روش نام‌گذاری مرسوم است. در این مثال، تابع عضوی که مبادرت به تنظیم (تخصیص مقادیر) عضو داده **courseName** می‌کند، **setCourseName** نام دارد و تابع عضوی که مقداری از عضو داده **courseName** بدست می‌آورد، تابع **getCourseName** نامیده می‌شود. توجه کنید که گاهاً توابع set، بنام *mutators* شناخته می‌شوند (چرا که مبادرت به تغییر یا اصلاح مقادیر می‌کنند)، و توابع get بنام *accessors* نامیده می‌شوند (چرا که به مقادیر دسترسی پیدا می‌کنند).

بخاطر دارید که اعلان اعضای داده با تصریح‌کننده دسترسی **private** موجب پنهان‌سازی داده می‌شود. تدارک دیدن توابع *set* و *get* به سرویس‌گیرنده‌های کلاس اجازه دسترسی به داده‌های پنهان شده را می‌دهند، اما بصورت غیرمستقیم. سرویس‌گیرنده از مبادرت خود به اصلاح یا بدست آوردن داده یک شی اطلاع دارد، اما از اینکه شی چگونه این عملیات را انجام می‌دهد، اطلاعی ندارد. در برخی از موارد امکان دارد کلاسی به یک روش مبادرت به عرضه داده در محیط داخلی نماید، در حالیکه همان داده را به روش مختلفی در اختیار سرویس‌گیرنده‌ها قرار می‌دهد. برای مثال، فرض کنید کلاس **Clock** زمانی از روز را بصورت یک عضو داده **time** و از نوع **int** و **private** عرضه می‌کند که تعداد ثانیه‌های از نیمه‌شب را ذخیره می‌سازد. با این همه، زمانیکه سرویس‌گیرنده‌ای تابع عضو **getTime** از شی **Clock** را فراخوانی می‌کند، زمان برحسب ساعت، دقیقه و ثانیه در یک رشته با فرمت "HH:MM:SS" به وی برگشت داده می‌شود. به همین ترتیب، فرض کنید کلاس **Clock** یک تابع set بنام **setTime** تدارک دیده که یک رشته پارامتری با فرمت "HH:MM:SS" دریافت می‌نماید. با استفاده از قابلیت‌های عرضه شده در فصل هیجدهم، تابع **setTime** قادر به تبدیل این رشته به تعداد ثانیه‌ها است، که تابع آن را در عضو داده **private** ذخیره سازد. همچنین تابع *set* می‌تواند به بررسی اعتبار مقدار دریافتی پرداخته و اعتبار آن را تایید یا لغو کند (مثلاً "12:30:43" معتبر بوده اما "42:85:70" معتبر نیست). توابع *set* و *get* به سرویس‌گیرنده امکان تعامل با یک شی را فراهم می‌آورند، اما داده **private** (خصوصی) شی همچنان بصورت کپسوله (پنهان) و ایمن در خودش نگهداری می‌شود.

www.yourshop.4kia.ir

همچنین توابع *set* و *get* یک کلاس می‌توانند توسط سایر توابع عضو موجود در درون کلاس به منظور دستکاری کردن داده **private** کلاس بکار گرفته شوند، اگر چه این توابع عضو می‌تواند بصورت مستقیم به داده **private** دسترسی پیدا کنند. در برنامه شکل ۵-۳، توابع عضو **getCourseName** و **setCourseName** از نوع توابع عضو **public** هستند، از اینرو در دسترس‌گیرنده‌های کلاس و همچنین خود کلاس قرار دارند. تابع عضو **displayMessage** تابع عضو **getCourseName** را برای بدست آوردن مقدار عضو داده **courseName** برای نمایش آن، فراخوانی می‌کند، با اینکه خود **displayMessage** می‌تواند بصورت مستقیم به **courseName** دسترسی پیدا کند. دسترسی به عضو داده از طریق تابع *get* آن، سبب ایجاد یک کلاس بهتر و پایدارتر می‌شود (کلاسی که نگهداری آن آسان بوده و کمتر از کار می‌افتد). اگر تصمیم به تغییر عضو داده **courseName** بگیریم، ساختار تعریف‌کننده **displayMessage** نیازمند اصلاح نخواهد بود، فقط بدنه توابع *get* و *set* که مستقیماً عضو داده را دستکاری می‌کنند نیاز به تغییر خواهند داشت. برای مثال، فرض کنید که می‌خواهیم نام دوره را در دو عضو داده عرضه کنیم، **courseName** (مثلاً "CS101") و **courseTitle** (مثلاً "Introduction to C++ Programming"). هنوز هم تابع **displayMessage** می‌تواند با یک فراخوانی تابع عضو **getCourseName** نام کامل دوره را بدست آورده و بعنوان بخشی از پیغام خوش‌آمدگویی به نمایش در آورد. در اینحالت، تابع **getCourseName** نیازمند ایجاد و برگشت دادن یک رشته **حاوی** **courseNumber** و بدنبال آن **courseTitle** است. در ادامه تابع عضو **displayMessage** عنوان کامل دوره "CS101 Introduction to C++ Programming" را به نمایش در خواهد آورد، چرا که از تغییرات صورت گرفته بر روی اعضای داده کلاس به دور مانده است. مزایای فراخوانی تابع *set* از یک تابع عضو دیگر کلاس به هنگام بحث اعتبارسنجی در بخش ۱۰-۳ بیشتر آشکار خواهد شد.

**برنامه‌نویسی ایده‌ال**

*سعی کنید همیشه میزان تاثیرات در اعضای داده کلاس را با استفاده از توابع set و get در سطح محلی نگهداری کنید.*

**مهندسی نرم‌افزار**

*نوشتن برنامه‌هایی که درک و نگهداری آنها آسان باشد، مهم است. تغییرات همیشه رخ می‌دهند. بعنوان برنامه‌نویس باید انتظار داشته باشید که کدهای نوشته شده توسط شما زمانی به تغییر نیاز پیدا خواهند کرد.*

**مهندسی نرم‌افزار**

*طراح کلاس نیازی به تدارک دیدن توابع set یا get برای هر ایتم داده* **private** *ندارد، این موارد فقط در صورت نیاز و شرایط مقتضی در نظر گرفته شوند. اگر سرویسی برای کد سرویس‌گیرنده مناسب باشد، باید آن سرویس در واسط* **public** *کلاس وارد گردد.*

www.yourshop.4kia.ir

***دیاگرام UML کلاس GradeBook با یک داده عضو و توابع set و get***

شکل ۶-۳ حاوی دیاگرام کلاس UML به روز شده برای نسخه‌ای از کلاس **GradeBook** بکار رفته در برنامه ۵-۳ است. این دیاگرام مبادرت به مدل‌سازی داده عضو **courseName** بعنوان یک صفت در بخش میانی کلاس کرده است. UML اعضای داده را در لیستی که در آن نام صفت، یک کولن و نوع صفت قرار گرفته عرضه می‌کند. نوع UML صفت **courseName**، نوع **String** است که متناظر با **string** در C++ می‌باشد. عضو داده **courseName** در C++ حالت **private** دارد و از اینرو در دیاگرام کلاس با یک علامت منفی (-) در مقابل نام صفت مشخص شده است. علامت منفی در UML معادل با تصریح‌کننده دسترسی **private** در C++ است. کلاس **GradeBook** حاوی سه تابع عضو **public** است، از اینرو در لیست دیاگرام کلاس این سه عملیات در بخش تحتانی یا سوم جای گرفته‌اند. بخاطر دارید که نماد جمع (+) قبل نام هر عملیات نشان می‌دهد که عملیات در C++ حالت **public** دارد. عملیات **setCourseName** دارای یک پارامتر **String** بنام **name** است. در UML نوع برگشتی از یک عملیات با قرار دادن یک کولن و نوع برگشتی پس از پرانتزهای نام عملیات مشخص می‌شود. تابع عضو **getCourseName** از کلاس **GradeBook** (شکل ۵-۳) دارای نوع **string** برگشتی در C++ است، بنابر این دیاگرام کلاس نوع برگشتی **String** را در UML به نمایش درآورده است. توجه نمائید که عملیات‌های **setCourseName** و **displayMessage** مقداری برگشت نمی‌دهند (نوع برگشتی آنها **void** است)، از اینرو در دیاگرام کلاس UML نوع برگشتی از پرانتزها برای آنها در نظر گرفته نشده است. UML از **void** همانند C++ در زمانیکه تابع مقداری برگشت نمی‌دهد، استفاده نمی‌کند.

**شکل ٦-۳ | دیاگرام کلاس UML برای کلاس GradeBook با یک صفت private برای courseName و عملیات‌های public برای توابع setCourseName، getCourseName و displayMessage**

## ۷-۳ مقداردهی اولیه شی‌ها با سازنده‌ها

همانطوری که از بخش ۶-۳ بخاطر دارید، زمانیکه یک شی از کلاس **GradeBook** ایجاد شد (شکل ۵-۳) عضو داده آن یعنی **courseName** بطور پیش‌فرض با یک رشته تهی مقداردهی اولیه شده بود. اگر بخواهیم به هنگام ایجاد یک شی از **GradeBook** نام دوره‌ای تدارک دیده شود، چه کاری باید انجام داد؟ هر کلاسی که اعلان می‌کنید می‌تواند یک سازنده (*constructor*) داشته باشد که می‌توان با استفاده از آن مبادرت به مقداردهی اولیه یک شی از کلاس به هنگام ایجاد شی کرد. سازنده یک تابع عضو ویژه است که بایستی همنام با نام کلاس تعریف شده باشد، از اینروست که کامپایلر می‌تواند آن را از دیگر توابع عضو کلاس تشخیص دهد. مهمترین تفاوت موجود مابین سازنده‌ها و توابع دیگر در این است که سازنده‌ها نمی‌تواند مقدار برگشت دهند، بنابر این نمی‌توانند نوع برگشتی داشته باشند (حتی **void**).

www.yourshop.4kia.ir

معمولاً، سازنده‌ها بصورت **public** اعلان می‌شود. غالباً کلمه "constructor" در برخی از نوشته‌ها بصورت خلاصه شده "ctor" بکار گرفته می‌شود، که استفاده از آن را ترجیح نداده‌ایم.

C++ نیازمند فراخوانی یک سازنده برای هر شی است که ایجاد می‌شود، در چنین حالتی مطمئن خواهیم بود که شی قبل از اینکه توسط برنامه بکار گرفته شود، بدرستی مقداردهی اولیه شده است. فراخوانی سازنده به هنگام ایجاد شی، بصورت غیرصریح یا ضمنی انجام می‌شود. در هر کلاسی که بصورت صریح سازنده‌ای را مشخص نکرده است، کامپایلر یک *سازنده پیش‌فرض* تدارک می‌بیند، این سازنده دارای پارامتر نمی‌باشد. برای مثال، زمانیکه خط 46 از برنامه شکل ۵-۳ یک شی **GradeBook** ایجاد می‌کند، سازنده پیش‌فرض فراخوانی می‌شود، چرا که در اعلان **myGradeBook** بطور صریح هیچ آرگومان سازنده‌ای مشخص نشده است. سازنده پیش‌فرض تدارک دیده شده از سوی کامپایلر یک شی **GradeBook** بدون هیچ گونه مقادیر اولیه برای اعضای داده شی ایجاد می‌کند. [نکته: برای اعضای داده که شی‌های از سایر کلاس‌ها هستند، سازنده پیش‌فرض بصورت ضمنی هر سازنده پیش‌فرض عضو داده را برای اطمینان از اینکه اعضاء داده به درستی مقداردهی اولیه شده‌اند، فراخوانی می‌کند. در واقع به این دلیل است که عضو داده **courseName** از نوع رشته (در برنامه شکل ۵-۳) با یک رشته تهی مقداردهی اولیه شده است. سازنده پیش‌فرض برای کلاس **string** مبادرت به تنظیم یک مقدار رشته‌ای با رشته تهی می‌کند، در بخش ۳-۱۰ مطالب بیشتری در ارتباط با مقداردهی اولیه اعضای داده که شی‌های از کلاس‌های دیگر هستند خواهید آموخت.]

در مثال برنامه شکل ۷-۳، نام یک دوره را به هنگام ایجاد یک شی از **GradeBook** مشخص کرده‌ایم (خط 49). در این مورد، آرگومان "CS101 Introduction to C++ Programming" به سازنده شی **GradeBook** ارسال می‌شود (خطوط 20-17) و مبادرت به مقداردهی اولیه **courseName** می‌نماید. برنامه شکل ۷-۳ یک کلاس **GradeBook** اصلاح شده تعریف کرده که حاوی یک سازنده با یک پارامتر رشته‌ای است که نام دورهٔ اولیه را دریافت می‌کند.

```
1  // Fig. 3.7: fig03_07.cpp
2  // Instantiating multiple objects of the GradeBook class and using
3  // the GradeBook constructor to specify the course name
4  // when each GradeBook object is created.
5  #include <iostream>
6  using std::cout;
7  using std::endl;
8
9  #include <string> // program uses C++ standard string class
10 using std::string;
11
12 // GradeBook class definition
```

www.yourshop.4kia.ir

```
13 class GradeBook
14 {
15 public:
16   // constructor initializes courseName with string supplied as argument
17    GradeBook( string name )
18    {
19       setCourseName( name );//call set function to initialize courseName
20    } // end GradeBook constructor
21
22    // function to set the course name
23    void setCourseName( string name )
24    {
25       courseName = name; // store the course name in the object
26    } // end function setCourseName
27
28    // function to get the course name
29    string getCourseName()
30    {
31       return courseName; // return object's courseName
32    } // end function getCourseName
33
34    // display a welcome message to the GradeBook user
35    void displayMessage()
36    {
37       // call getCourseName to get the courseName
38       cout << "Welcome to the grade book for\n" << getCourseName()
39          << "!" << endl;
40    } // end function displayMessage
41 private:
42    string courseName; // course name for this GradeBook
43 }; // end class GradeBook
44
45 // function main begins program execution
46 int main()
47 {
48    // create two GradeBook objects
49    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
50    GradeBook gradeBook2( "CS102 Data Structures in C++" );
51
52    // display initial value of courseName for each GradeBook
53    cout <<"gradeBook1 created for course:"<< gradeBook1.getCourseName()
54       <<"\ngradeBook2 created for course:"<< gradeBook2.getCourseName()
55       << endl;
56    return 0; // indicate successful termination
57 } // end main
```

```
gradeBook1 created for course : CS101 Introduction to C++ Programming
gradeBook2 created for course : CS102 Data Structures in C++
```

**شکل ۷-۳ | نمونه‌سازی شی‌های مضاعف از کلاس GradeBook و استفاده از سازنده GradeBook برای مشخص کردن نام دوره به هنگام ایجاد هر شی GradeBook.**

### *تعریف سازنده*

در خطوط 20-17 برنامه شکل ۷-۳ یک سازنده برای کلاس **GradeBook** تعریف شده است. توجه کنید که سازنده دارای نام مشابه همانند کلاس خود یعنی **GradeBook** است. یک سازنده توسط لیست پارامتری، داده مورد نیاز برای انجام وظیفه خود را مشخص می‌سازد. زمانیکه یک شی جدید ایجاد

www.yourshop.4kia.ir

می‌کنید، این داده را در درون پرانتزهای قرار گرفته پس از نام شی قرار می‌دهید (همانند خطوط 50-49). خط 17 بر این نکته دلالت دارد که سازنده کلاس **GradeBook** دارای یک پارامتر رشته‌ای بنام **name** است. دقت کنید که در خط 17 نوع برگشتی مشخص نشده است، چرا که سازنده‌ها نمی‌توانند مقدار برگشت دهند (حتی **void**).

خط 19 در بدنه سازنده مبادرت به ارسال پارامتر **name** سازنده به تابع عضو **setCourseName** می‌کند که مقداری به عضو داده **courseName** تخصیص می‌دهد. تابع عضو **setCourseName** (خطوط 26-23) فقط مبادرت به تخصیص پارامتر **name** خود به عضو داده **courseName** می‌کند، بنابر این ممکن است تعجب کنید که چرا زحمت فراخوانی **setCourseName** را در خط 19 به خود داده‌ایم، در حالیکه سازنده بطور مشخص عملیات تخصیص **courseName=name** را انجام می‌دهد. در بخش ۱۰-۳، مبادرت به اصلاح **setCourseName** برای انجام عملیات اعتبارسنجی خواهیم کرد. در این بخش است که مزیت فراخوانی **setCourseName** از سازنده آشکار خواهد شد. توجه نمائید که در سازنده (خط 17) و هم تابع **setCourseName** (خط 23) از پارامتری بنام **name** استفاده شده است. می‌توانید از اسامی پارامتری یکسان در توابع مختلف استفاده کنید چرا که پارامترها حالت محلی برای هر تابع دارند و سبب تداخل با دیگری نمی‌شوند.

***تست کلاس GradeBook***

خطوط 57-46 از برنامه شکل ۷-۳ حاوی تعریف تابع **main** است که مبادرت به تست کلاس **GradeBook** و اثبات مقداردهی اولیه شی **GradeBook** با استفاده از یک سازنده می‌کند. خط 49 در تابع **main** اقدام به ایجاد و مقداردهی اولیه یک شی **GradeBook** بنام **gradeBook1** می‌کند. زمانی که این خط از کد اجرا شود، سازنده **GradeBook** در خطوط 20-17 همراه با آرگومان "CS101 Introduction to C++ Programming" برای مقداردهی اولیه نام دوره **gradeBook1** فراخوانی می‌شود، البته این فراخوانی بصورت ضمنی و توسط C++ صورت می‌گیرد. خط 50 تکرار این فرآیند برای یک شی **GradeBook** بنام **gradeBook2** است، این بار، آرگومان "CS102 Data Structures in C++" برای مقداردهی اولیه نام دوره **gradeBook2** ارسال می‌شود. خطوط 54-53 از تابع عضو هر شی **getCourseName** برای بدست آوردن اسامی دوره و نمایش اینکه آنها در زمان ایجاد مقداردهی اولیه شده‌اند، استفاده کرده‌اند. خروجی برنامه تایید می‌کند که هر شی **GradeBook** از کپی عضو داده **courseName** متعلق بخود نگهداری کرده است.

***دو روش برای تدارک دیدن یک سازنده پیش‌فرض برای یک کلاس***

www.yourshop.4kia.ir

به هر سازنده‌ای که هیچ آرگومانی دریافت نکند، سازنده پیش‌فرض می‌گویند. یک کلاس به یکی از دو روش زیر سازندهٔ پیش‌فرض بدست می‌آورد:

۱- کامپایلر بطور ضمنی یک سازنده پیش‌فرض برای کلاسی که سازنده‌ای برای آن تعریف نشده است، ایجاد می‌کند. چنین سازنده‌ای مبادرت به مقداردهی اولیه اعضای داده کلاس نمی‌کند، اما برای هر عضو داده که شی از کلاس دیگری هستند، سازنده پیش‌فرض را فراخوانی می‌کند [نکته: معمولاً یک متغیر مقداردهی نشده حاوی یک مقدار «*اشغال*» است (مثلاً یک متغیر **int** مقداردهی نشده می‌تواند حاوی 858993460- باشد که این مقدار در بسیاری از برنامه‌ها برای این متغیر غلط می‌باشد).]

۲- برنامه‌نویس بصورت صریح مبادرت به تعریف سازنده‌ای نماید که آرگومانی دریافت نمی‌کند. چنین سازنده‌ای شروع به مقداردهی اولیه مطابق نظر برنامه‌نویس کرده و سازنده پیش‌فرض را برای هر داده عضو که شی از یک کلاس دیگر است فراخوانی می‌کند.

اگر برنامه‌نویس مبادرت به تعریف یک سازنده با آرگومان نماید، دیگر C++ بصورت ضمنی یک سازنده پیش‌فرض برای آن کلاس ایجاد نخواهد کرد. توجه نمائید که برای هر نسخه از کلاس **GradeBook** در برنامه‌های ۱-۳، ۳-۳ و ۵-۳ کامپایلر بصورت ضمنی یک سازنده پیش‌فرض تعریف کرده است.

**اجتناب از خطا**

*بجز در مواردی که نیازی به مقداردهی اولیه اعضای داده کلاس ضروری نیست (تقریباً هیچ وقت)، یک سازنده پیش‌فرض در نظر بگیرید که حتماً اعضای داده کلاس را با مقادیر مناسب به هنگام ایجاد هر شی جدید مقداردهی اولیه نماید.*

**مهندسی نرم‌افزار**

*اعضای داده می‌توانند توسط سازنده یک کلاس مقداردهی اولیه شوند یا پس از ایجاد شی تنظیم گردند. با این وجود، از منظر مهندسی نرم‌افزار بهتر خواهد بود تا یک شی بطور کامل و قبل از اینکه سرویس‌گیرنده‌ای مبادرت به فراخوانی توابع عضو آن شی نماید، مقداردهی اولیه شده باشد. بطور کلی، نبایستی فقط متکی به کد سرویس‌گیرنده باشید که بدقت و بدرستی یک شی را مقداردهی نماید.*

***افزودن سازنده به دیاگرام کلاس UML کلاس GradeBook***

دیاگرام کلاس UML در شکل ۸-۳ مبادرت به مدل کردن کلاس **GradeBook** برنامه ۷-۳ کرده است، که دارای یک سازنده با پارامتر **name** از نوع رشته است (نوع **String** در UML). همانند عملیات‌ها، UML مبادرت به مدل‌سازی سازنده‌ها در بخش سوم کلاس در دیاگرام کلاس می‌کند. برای

www.yourshop.4kia.ir

تمایز قائل شدن مابین یک سازنده از یک عملیات کلاس، UML مبادرت به قرار دادن کلمه "constructor" مابین گیومه (« و ») قبل از نام سازنده می‌کند. قرار دادن لیست سازنده کلاس قبل از دیگر عملیات‌ها در بخش سوم امری رایج است.

**شکل ۸-۳ | دیاگرام کلاس UML نشان می‌دهد که کلاس GradeBook دارای یک سازنده با پارامتر name از نوع String است.**

## ۸-۳ قرار دادن کلاس در یک فایل مجزا برای استفاده مجدد

تا آنجا که از منظر برنامه‌نویسی نیاز داشته باشیم به توسعه کلاس **GradeBook** ادامه خواهیم داد، از اینرو اجازه دهید تا وارد برخی از مباحث مهندسی نرم‌افزار شویم. یکی از مزایای تعریف دقیق یک کلاس در این است که به هنگام بسته (*package*) کردن صحیح، کلاس‌های ما می‌توانند توسط سایر برنامه‌نویسان در سرتاسر جهان بکار گرفته شوند (استفاده مجدد). برای مثال، کتابخانه استاندارد C++ دارای نوع **string** است که می‌توانیم از آن در هر برنامه C++ استفاده کنیم (استفاده مجدد). البته این کار را با وارد کردن فایل سرآیند <**string**> در برنامه انجام می‌دهیم.

متاسفانه، برنامه‌نویسانی که مایل به استفاده از کلاس **GradeBook** ما هستند، نمی‌تواند بسادگی و فقط با وارد کردن فایل از برنامه ۷-۳ به یک برنامه دیگر از آن استفاده کنند. همانطوری که در فصل دوم آموختید، تابع **main** اجرای هر برنامه‌ای را آغاز می‌کند و هر برنامه باید دارای یک تابع **main** باشد. اگر برنامه‌نویسان دیگر مبادرت به قرار دادن کد برنامه ۷-۳ نمایند، چمدان بزرگی بدست خواهند گرفت، تابع **main** ما، و برنامه آنها حاوی دو تابع **main** خواهد بود. زمانیکه مبادرت به کامپایل برنامه کنند، کامپایلر متوجه خطا خواهد شد، چرا که هر برنامه‌ای فقط می‌تواند یک تابع main داشته باشد. برای مثال، اگر مبادرت به کامپایل برنامه‌ای با دو تابع **main** در برنامه Mircrosoft Visual C++ .NET کنید، خطای زیر تولید می‌شود:

```
error C2084: function 'int main(void)' already has a body
```

هنگامی که کامپایلر سعی در کامپایل دومین تابع **main** می‌کند با خطا مواجه می‌شود. به همین ترتیب کامپایلر GNU C++ خطای زیر را تولید می‌کند:

```
redefinition of 'int main()'
```

این خطاها بر این نکته دلالت دارند که برنامه در حال حاضر دارای یک تابع **main** است، بنابر این، قرار دادن **main** در همان فایل با تعریف کلاس از اینکه بتوان از کلاس در سایر برنامه‌ها استفاده مجدد

www.yourshop.4kia.ir

کرد، ممانعت بعمل می‌آورد. در این بخش، به توضیح نحوه ایجاد کلاس **GradeBook** با هدف استفاده مجدد خواهیم پرداخت. روشی که در آن کلاس را در یک فایل مجزا از تابع **main** قرار می‌دهیم.

***فایل‌های سرآیند***

هر کدام یک از مثال‌های مطرح شده تا بدین مرحله متشکل از یک فایل **.cpp** بودند که بعنوان فایل کد-منبع نیز شناخته می‌شوند. این مثال‌ها حاوی تعریف کلاس **GradeBook** و یک تابع **main** بودند. به هنگام ایجاد یک برنامه شی‌گرای C++، تعریف کد منبع با قابلیت استفاده مجدد (همانند یک کلاس) در یک فایل که دارای پسوند فایل **.h** (فایل سرآیند) است، امری رایج می‌باشد. در برنامه‌ها از رهنمودهای پیش‌پردازنده **#include** برای وارد کردن فایل‌های سرآیند و برخوردار شدن از مزیت کامپونت‌های نرم‌افزاری با قابلیت استفاده مجدد کمک گرفته می‌شود، همانند نوع **string** تدارک دیده شده در کتابخانه استاندارد C++ و نوع‌های تعریف شده توسط کاربر مانند کلاس **GradeBook**.

در مثال بعدی، مبادرت به متمایز کردن کد برنامه از ۷-۳ با دو فایل GradeBook.h برنامه شکل ۹-۳ و fig03_10.cpp برنامه شکل ۱۰-۳ می‌کنیم. همانطوری که در فایل سرآیند شکل ۹-۳ مشاهده می‌کنید، این فایل فقط حاوی تعریف کلاس **GradeBook** (خطوط 41-11) و خطوط 8-3 است که به کلاس **GradeBook** اجازه استفاده از **endl**، **cout** و نوع **string** را می‌دهند. تابع **main** که از کلاس **GradeBook** استفاده می‌کند در فایل کد منبع fig03_10.cpp شکل ۱۰-۳ تعریف شده است، در خطوط 21-10. برای کمک به شما برای مهیا شدن برای کار با برنامه‌های بزرگ که در این کتاب و بازار کار با آنها مواجه خواهید شد، غالباً از یک فایل کد منبع متمایز یا جداگانه حاوی تابع **main** برای تست کلاس‌های خود استفاده می‌کنیم (به این برنامه، برنامه راه‌انداز یا درایور می‌گویند). بزودی با نحوه استفاده یک فایل کد منبع با **main** از تعریف کلاس موجود در یک فایل سرآیند در ایجاد شی‌های از یک کلاس آشنا خواهید شد.

```
1  // Fig. 3.9: GradeBook.h
2  // GradeBook class definition in a separate file from main.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string> // class GradeBook uses C++ standard string class
8  using std::string;
9
10 // GradeBook class definition
11 class GradeBook
12 {
13 public:
14   // constructor initializes courseName with string supplied as argument
15    GradeBook( string name )
16    {
```

www.yourshop.4kia.ir

```
17      setCourseName( name );// call set function to initialize courseName
18    } // end GradeBook constructor
19
20    // function to set the course name
21    void setCourseName( string name )
22    {
23       courseName = name; // store the course name in the object
24    } // end function setCourseName
25
26    // function to get the course name
27    string getCourseName()
28    {
29       return courseName; // return object's courseName
30    } // end function getCourseName
31
32    // display a welcome message to the GradeBook user
33    void displayMessage()
34    {
35       // call getCourseName to get the courseName
36       cout << "Welcome to the grade book for\n" << getCourseName()
37          << "!" << endl;
38    } // end function displayMessage
39 private:
40    string courseName; // course name for this GradeBook
41 }; // end class GradeBook
```

**شکل ۹-۳ | تعریف کلاس GradeBook.**

***وارد کردن فایل سرآیند که حاوی یک کلاس تعریف شده از سوی کاربر است***

یک فایل سرآیند همانند GradeBook.h (برنامه شکل ۹-۳) نمی‌تواند برای شروع اجرای برنامه بکار گرفته شود، چرا که حاوی تابع **main** نمی‌باشد. اگر سعی در کامپایل و لینک خود GradeBook.h به منظور ایجاد یک برنامه اجرایی کنید، Microsoft Visual C++ .NET پیغام خطای لینکر را تولید خواهد کرد:

```
error LNK2019:unresolved external symbol _main referenced in
function _mainCRTStrartup
```

اگر در حال استفاده از GNU C++ بر روی لینوکس باشید، پیغام خطای لینکر بصورت زیر خواهد بود:

```
undefined reference to 'main'
```

این خطا بر این نکته دلالت دارد که لینکر قادر به یافتن تابع **main** برنامه نشده است. برای تست کلاس **GradeBook** تعریف شده در شکل ۹-۳ بایستی یک فایل کد منبع جداگانه حاوی یک تابع **main** (همانند شکل ۱۰-۳) بنویسید که مبادرت به نمونه‌سازی و استفاده از شی‌های کلاس کند.

از بخش ۴-۳ بخاطر دارید که کامپایلر از مفهوم نوع‌های بنیادین همانند **int** مطلع است، اما با **GradeBook** آشنا نیست چرا که این نوع یک نوع تعریف شده از سوی کاربر (برنامه‌نویس) است. در

www.yourshop.4kia.ir

واقع، کامپایلر حتی از کلاس‌های کتابخانه استاندارد ++C اطلاعی ندارد. برای کمک به کامپایلر برای اینکه متوجه نحوه استفاده از یک کلاس شود، بایستی بصورت صریح تعریف کلاس را در اختیار آن قرار دهیم. به همین دلیل است که برای مثال، به هنگام استفاده از نوع **string**، باید برنامه شامل فایل سرآیند <**string**> باشد. با انجام اینکار، کامپایلر قادر به تعیین میزان حافظه‌ای خواهد بود که باید برای هر شی از کلاس رزرو نماید و فراخوانی صحیح توابع عضو کلاس را تضمین می‌کند.

```
1  // Fig. 3.10: fig03_10.cpp
2  // Including class GradeBook from file GradeBook.h for use in main.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "GradeBook.h" // include definition of class GradeBook
8
9  // function main begins program execution
10 int main()
11 {
12    // create two GradeBook objects
13    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
14    GradeBook gradeBook2( "CS102 Data Structures in C++" );
15
16    // display initial value of courseName for each GradeBook
17    cout <<"gradeBook1 created for course:"<< gradeBook1.getCourseName()
18       <<"\ngradeBook2 created for course:"<< gradeBook2.getCourseName()
19       << endl;
20    return 0; // indicate successful termination
21 } // end main
```

```
gradeBook1 created for course : CS101 Introduction to C++ Programming
gradeBook2 created for course : CS102 Data Structures in C++
```

**شکل ۱۰-۳ | وارد کردن کلاس GradeBook از فایل GradeBook.h برای استفاده در main.**

برای ایجاد شی‌های **gradeBook1** و **gradeBook2** در خطوط 14-13 برنامه ۱۰-۳، باید کامپایلر از سایز یک شی **GradeBook** مطلع باشد. در حالیکه شی‌ها بصورت مفهومی حاوی اعضای داده و توابع عضو هستند، شی‌های ++C فقط حاوی داده می‌باشند. کامپایلر فقط یک کپی از توابع عضو کلاس ایجاد کرده و آن کپی را در میان سایر شی‌های کلاس به اشتراک می‌گذارد. البته هر شی نیازمند کپی متعلق بخود از اعضای داده کلاس است، چرا که محتویات آنها می‌توانند با سایر شی‌ها بسیار تفاوت داشته باشند (همانند دو شی مختلف **BankAccount** (حساب بانکی) که دو عضو داده متفاوت **balance** یعنی موجودی دارند). با این وجود، کد تابع عضو تغییر پذیر نیست، از اینرو می‌تواند در میان تمام شی‌های کلاس به اشتراک گذاشته شود. بنابر این، سایز یک شی وابسته به میزان فضای حافظه مورد نیاز برای ذخیره‌سازی عضوهای داده کلاس است. با وارد کردن GradeBook.h در خط 7، به کامپایلر امکان دسترسی به اطلاعات مورد نیاز (شکل ۹-۳، خط 40) برای تعیین سایز یک شی **GradeBook** و تعیین

www.yourshop.4kia.ir

اینکه آیا شی‌های از آن کلاس بدرستی بکار گرفته شده‌اند یا خیر، داده می‌شود (در خطوط 14-13 و -17 18 شکل ۱۰-۳).

خط 7 به پیش‌پردازنده C++ دستور جایگزین رهنمود با یک کپی از محتویات GradeBook.h (تعریف کلاس **GradeBook**) قبل از کامپایل برنامه را صادر می‌کند. زمانیکه فایل کد منبع fig03_10.cpp کامپایل می‌شود، حاوی تعریف کلاس **GradeBook** بوده (بدلیل **#include**)، و کامپایلر قادر به تعیین نحوه ایجاد شی‌های **GradeBook** و مشاهده فراخوانی صحیح توابع عضو خواهد بود. اکنون که تعریف کلاس در یک فایل سرآیند (بدون تابع **main**) قرار دارد، می‌توانیم سرآیند را در هر برنامه‌ای که نیاز به استفاده مجدد از کلاس **GradeBook** دارد وارد کنیم.

### *نحوه یافتن فایل‌های سرآیند*

توجه کنید که نام فایل سرآیند GradeBook.h در خط 7 شکل ۱۰-۳ در میان جفت گوتیشن (" ") بجای < > محدود شده است. معمولاً فایلهای کد منبع برنامه و فایل‌های سرآیند تعریف شده از سوی کاربر در همان شاخه قرار داده می‌شوند. زمانیکه پیش‌پردازنده با یک نام فایل سرآیند در میان گوتیشن‌ها مواجه می‌شود (مثل **"GradeBook.h"**)، مبادرت به مکان‌یابی فایل سرآیند در همان شاخه می‌کند که فایل حاوی **#include** در آن قرار دارد. اگر پیش‌پردازنده موفق به یافتن فایل سرآیند در آن شاخه نشود، شروع به جستجوی آن فایل در همان موقعیت(ها) همانند فایل‌های سرآیند کتابخانه استاندارد C++ می‌کند. زمانیکه پیش‌پردازنده با نام فایل سرآیند در میان < > مواجه شود (مثل **<iostream>**) فرض می‌کند که سرآیند بخشی از کتابخانه استاندارد C++ است و به شاخه‌ای که برنامه در آن قرار دارد نگاه نمی‌کند.

**اجتناب از خطا**

*برای اطمینان از اینکه پیش‌پردازنده قادر به مکان‌یابی صحیح فایل‌های سرآیند خواهد بود، بایستی رهنمود پیش‌پردازنده **#include** مبادرت به قرار دادن اسامی فایل‌های سرآیند تعریف شده از سوی کاربر در میان گوتیشن‌ها کند (همانند **"GradeBook.h"**) و اسامی فایل‌های سرآیند کتابخانه استاندارد **C++** را در میان < > قرار دهد (همانند **<iostream>**).*

### *مبحث مهندسی نرم‌افزار*

اکنون که کلاس **GradeBook** در یک فایل سرآیند تعریف شده است، کلاس از قابلیت استفاده مجدد برخوردار شده است. متاسفانه، قرار دادن تعریف یک کلاس در یک فایل سرآیند همانند شکل ۹-۳ هنوز هم کل ساختار پیاده‌سازی کلاس را در دید سرویس‌گیرندگان کلاس قرار می‌دهد. GradeBook.h یک فایل متنی ساده است که هر کسی می‌تواند آن را باز کرده و بخواند. اصول مهندسی نرم‌افزار بر این نکته تاکید دارد که به هنگام استفاده از یک شی از کلاسی، کد سرویس‌گیرنده فقط نیاز به

www.yourshop.4kia.ir

فراخوانی توابع عضو مورد نیاز، اطلاع داشتن از تعداد آرگومان‌ها در هر تابع عضو و نوع برگشتی هر تابع عضو دارد. نیازی نیست که کد سرویس‌گیرنده از نحوه پیاده‌سازی توابع مطلع باشد.

اگر کد سرویس‌گیرنده از نحوه پیاده‌سازی یک کلاس مطلع باشد، امکان دارد برنامه‌نویس کد سرویس‌گیرنده براساس جزئیات ساختار پیاده‌سازی کلاس مبادرت به برنامه‌نویسی کد سرویس‌گیرنده کند. ایده‌آل نخواهد بود، اگر در پیاده‌سازی تغییری رخ دهد، سرویس‌گیرنده کلاس مجبور به تغییر در کد خود نباشد. با پنهان‌سازی جزئیات پیاده‌سازی کلاس، کار تغییر در ساختار کلاس آسانتر شده و احتمال تغییر در کد سرویس‌گیرنده‌های کلاس به حداقل می‌رسد. در بخش ۹-۳ شما را با نحوه تفکیک کلاس GradeBook به دو فایل آشنا خواهیم کرد که با اینکار

۱- کلاس قابلیت استفاده مجدد پیدا می‌کند

۲- سرویس گیرنده‌های کلاس از توابع عضو تدارک دیده شده توسط کلاس، نحوه فراخوانی و نوع برگشتی از آنها مطلع می‌شوند

۳- سرویس گیرنده‌ها از نحوه پیاده‌سازی توابع عضو کلاس مطلع نخواهند بود.

## ۹-۳ جداسازی واسط از پیاده‌سازی

در بخش قبلی، با نحوه افزودن قابلیت استفاده مجدد از نرم‌افزار توسط جداسازی تعریف از کد سرویس‌گیرنده که از کلاس استفاده می‌کند، آشنا شدید. اکنون بحث دیگری مطرح می‌کنیم که یکی از اصول کاربردی در مهندسی نرم‌افزار است، بحث *جداسازی واسط از پیاده‌سازی*.

### *واسط یک کلاس*

واسط‌ها تعریف‌کننده روش‌های استاندارد در برقراری تعامل چیزهای همانند مردم و سیستم‌ها با یکدیگر هستند. برای‌مثال، کنترل‌های(دکمه‌های) رادیو نقش واسط مابین کاربران رادیو و کامپونت‌های داخلی آن بازی می‌کنند. کنترل‌ها به کاربران امکان انجام کارهای مشخصی را می‌دهند، کارهای همانند تغییر ایستگاه، تنظیم صدا و انتخاب ایستگاه‌های FM و AM. امکان دارد رادیوهای مختلف این عملیات‌ها را به روش‌های متفاوتی انجام دهند، برخی از دکمه‌های فشاری، برخی از صفات شاخص‌دار و برخی از دستورات صوتی پشتیبانی می‌کنند. واسط، تصریح‌کننده نوع عملیاتی است که رادیو اجازه انجام آن را به کاربر می‌دهد، اما نشان‌دهنده نحوه پیاده‌سازی عملیات در داخل رادیو نمی‌باشد.

به همین ترتیب، واسط یک کلاس توصیف‌کننده سرویس‌های (خدماتی) است که کلاس در اختیار سرویس‌گیرندگان خود قرار می‌دهد و نحوه تقاضای این سرویس‌ها را مشخص می‌سازد، اما چیزی از نحوه انجام کار به سرویس‌ها ارائه نمی‌کند. واسط یک کلاس متشکل از توابع عضو **public** که بعنوان سرویس‌های سراسری یا **public** کلاس هم شناخته می‌شوند، است. برای مثال، واسط کلاس

www.yourshop.4kia.ir

**GradeBook** (شکل ۹-۳) حاوی یک سازنده و توابع عضو **setCourseName**، **getCourseName** و **displayMessage** است. سرویس‌گیرنده **GradeBook** (مثل **main** در شکل ۱۰-۳) از این توابع برای تقاضای سرویسی از کلاس استفاده می‌کند. همانطوری که بزودی مشاهده خواهید کرد، می‌توانید واسط یک کلاس را با نوشتن تعریف کلاسی که فقط لیستی از اسامی توابع عضو، نوع برگشتی و نوع پارامترها دارد، مشخص کنید.

***تفکیک واسط از پیاده‌سازی***

در مثال‌های قبلی، هر تعریف کلاس حاوی تعاریف کاملی از توابع عضو **public** کلاس و اعلان‌های از اعضای داده **private** آن بود. با این وجود، از لحاظ مهندسی نرم‌افزار بهتر خواهد بود تا توابع عضو در خارج از تعریف کلاس، تعریف گردند، بنابر این جزئیات پیاده‌سازی آنها از دید کد سرویس‌گیرنده‌ها پنهان خواهند ماند. با اینکار مطمئن خواهید شد که برنامه‌نویسان نخواهند توانست براساس جزئیات پیاده‌سازی کلاس شما، مبادرت به نوشتن کد سرویس‌گیرنده کنند. اگر چنین کاری کنند، در صورتی که ساختار پیاده‌سازی کلاس را تغییر دهید، به احتمال زیاد کد آنها با شکست مواجه خواهد شد.

برنامه موجود در شکل‌های ۱۱-۳ الی ۱۳-۳ مبادرت به جداسازی واسط کلاس **GradeBook** از بخش پیاده‌سازی آن با تقسیم تعریف کلاس شکل ۹-۳ به دو فایل، فایل سرآیند GradeBook.h (شکل ۱۱-۳) که در آن کلاس **GradeBook** تعریف شده و فایل کد منبع GradeBook.cpp (شکل ۱۲-۳) که توابع عضو **GradeBook** در آن تعریف شده‌اند، می‌کند. بطور قراردادی تعاریف توابع عضو در یک فایل کد منبع همنام با نام فایل سرآیند کلاس جای داده می‌شوند، بجز اینکه پسوند فایل **.cpp** است. فایل کد منبع fig03_13.cpp (شکل ۱۳-۳) تعریف‌کننده تابع **main** است (کد سرویس‌گیرنده). کد و خروجی شکل ۱۳-۳ یکسان با شکل ۱۰-۳ است. شکل ۱۴-۳ نشاندهنده نحوه کامپایل این فایل برنامه از منظر برنامه‌نویس کلاس **GradeBook** و برنامه‌نویس کد سرویس‌گیرنده است، در ارتباط با این تصویر توضیحاتی ارائه خواهیم داد.

***GradeBook.h: تعریف واسط کلاس با نمونه اولیه تابع***

فایل سرآیند GradeBook.h (شکل ۱۱-۳) حاوی نسخه دیگری از تعریف کلاس **GradeBook** است (خطوط 17-9). این نسخه شبیه به نسخه موجود در شکل ۹-۳ است، اما تعاریف تابع در شکل ۹-۳ با *نمونه اولیه تابع* (*function prototype*) جایگزین شده‌اند (خطوط 15-12)، که توصیف کننده واسط **public** کلاس است بدون اینکه پیاده‌سازی تابع عضو کلاس را آشکار کرده باشد. نمونه اولیه تابع، اعلانی از تابع است که به کامپایلر نام تابع، نوع برگشتی آن و نوع پارمترهای آن را بیان می‌کند. دقت کنید که فایل سرآیند هنوز هم مشخص‌کننده عضو داده **private** کلاس است (خط 17). مجدداً بایستی

www.yourshop.4kia.ir

کامپایلر از اعضای داده مطلع باشد تا بتواند میزان حافظه رزرو شده برای هر شی از کلاس را تعیین کند. با وارد کردن فایل سرآیند GradeBook.h در کد سرویس‌گیرنده (خط 8 از شکل ۱۳-۳) این اطلاعات در اختیار کامپایلر قرار می‌گیرد.

نمونه اولیه تابع در خط 12 از شکل ۱۱-۳ بر این نکته دلالت دارد که سازنده نیازمند یک پارامتر رشته‌ای است. بخاطر دارید که سازنده‌ها دارای نوع برگشتی نیستند، از اینرو نوع برگشتی در نمونه اولیه تابع قرار داده نشده است. نمونه اولیه تابع عضو **setCourseName** در خط 13 نشان می‌دهد که **setCourseName** نیازمند یک پارامتر رشته‌ای بوده و مقداری برگشت نمی‌دهد (نوع برگشتی آن **void** است). نمونه اولیه تابع عضو **getCourseName** نشان می‌دهد که تابع نیازمند پارامتر نبوده و رشته برگشت می‌دهد (خط 14).

```
1  .// Fig. 3.11: GradeBook.h
2  .// GradeBook class definition. This file presents GradeBook's public
3  .// interface without revealing the implementations of GradeBook's member
4  .// functions, which are defined in GradeBook.cpp.
5  .#include <string> // class GradeBook uses C++ standard string class
6  .using std::string;
7  .
8  .// GradeBook class definition
9  .class GradeBook
10 .{
11 .public:
12 .   GradeBook( string ); // constructor that initializes courseName
13 .   void setCourseName( string ); // function that sets the course name
14 .   string getCourseName(); // function that gets the course name
15 .   void displayMessage(); // function that displays a welcome message
16 .private:
17 .   string courseName; // course name for this GradeBook
18 .}; // end class GradeBook
```

**شکل ۱۱-۳ | تعریف کلاس GradeBook حاوی نمونه اولیه تابع که مشخص‌کننده واسط کلاس است.**

در پایان، نمونه اولیه تابع عضو displayMessage قرار دارد که مشخص می‌کند این تابع نیازمند پارامتر نبوده و مقداری هم برگشت نخواهد داد (خط 15). این نمونه‌های اولیه تابع همانند سرآیندهای تابع متناظر در شکل ۹-۳ هستند، بجز اینکه اسامی پارامتر (که در نمونه‌های اولیه اختیاری هستند) در نظر گرفته نشده‌اند و اینکه نمونه اولیه هر تابع باید با یک سیمکولن خاتمه پذیرد.

**خطای برنامه‌نویسی**

*فراموش کردن سیمکولن در انتهای نمونه اولیه یک تابع، خطای نحوی است.*

**برنامه‌نویسی ایده‌ال**

*اگر چه اسامی پارامتر در نمونه اولیه تابع اختیاری است (توسط کامپایلر نادیده گرفته می‌شوند) اما برخی از برنامه‌نویسان با هدف مستندسازی از این اسامی استفاده می‌کنند.*

www.yourshop.4kia.ir

**اجتناب از خطا**

اسامی پارامترها در نمونه اولیه تابع (که توسط کامپایلر نادیده گرفته می‌شوند) می‌توانند در صورت استفاده اشتباه یا تداخل اسامی، مشکل‌ساز شوند. به همین دلیل برخی از برنامهٔ‌نویسان ابتدا یک کپی از اولین خط تعریف تابع مربوطه تهیه و نمونه اولیه تابع را ایجاد (در صورتیکه منبع تابع در اختیار باشد)، سپس مبادرت به الحاق یک سیمکولن به انتهای هر نمونه اولیه می‌کنند.

***GradeBook.cpp: تعریف توابع عضو در یک فایل کد منبع جداگانه***

فایل کد منبع GradeBook.cpp شکل ۱۲-۳ تعریف‌کننده توابع عضو کلاس **GradeBook** است که در خطوط 15-12 از شکل ۱۱-۳ اعلان شده‌اند. تعریف تابع عضو در خطوط 34-11 قرار دارد و تقریباً با تعاریف موجود در خطوط 38-15 شکل ۹-۳ یکسان هستند.

توجه کنید که نام هر تابع عضو در سرآیندهای تابع (خطوط 11، 17، 23 و 29) پس از نام کلاس و **::** قرار گرفته است، که بعنوان عملگر *تفکیک قلمرو باینری* شناخته می‌شود. این عملگر مبادرت به «گره زدن» هر تابع عضو با تعریف کلاس **GradeBook** (که هم اکنون جدا شده) می‌کند، که توابع عضو کلاس و اعضای داده را اعلان کرده است. در صورتیکه "**GradeBook::**" قبل از نام تابع قرار داده نشود، این توابع توسط کامپایلر بعنوان توابع عضو از کلاس **GradeBook** تشخیص داده نشده و کامپایلر آنها را همانند توابع «*آزاد*» یا «*بی‌قاعده*» همانند **main** در نظر می‌گیرد. چنین توابعی قادر به دسترسی به داده **private** کلاس **GradeBook** یا فراخوانی توابع عضو کلاس نخواهند بود. بنابر این، کامپایلر نمی‌تواند این توابع را کامپایل نماید. برای مثال، خطوط 19 و 25 که به متغیر **courseName** دسترسی پیدا می‌کنند می‌تواند سبب‌ساز خطای کامپایل شوند، چرا که **courseName** بعنوان یک متغیر محلی در هر تابع اعلان نشده است. کامپایلر اطلاعی ندارد که **courseName** بصورت یک عضو داده کلاس **GradeBook** اعلان شده است.

**خطای برنامه‌نویسی**

*به هنگام تعریف توابع عضو کلاس درخارج از آن کلاس، فراموش کردن نام کلاس و عملگر تفکیک قلمرو باینری (::) قبل از اسامی توابع، خطای کامپایل بدنبال خواهد داشت.*

برای نشان دادن این که توابع عضو در GradeBook.cpp بخشی از کلاس **GradeBook** هستند، ابتدا فایل سرآیند GradeBook.h را وارد کرده‌ایم (خط 8 از شکل ۱۲-۳). این کار به ما اجازه دسترسی به کلاسی بنام **GradeBook** در فایل GradeBook.cpp را می‌دهد. به هنگام کامپایل GradeBook.cpp، کامپایلر از اطلاعات موجود در **GradeBook** استفاده می‌کند تا مطمئن شود که

www.yourshop.4kia.ir

۱- اولین خط هر تابع عضو (خطوط 11، 17، 23 و 29) با نمونه اولیه خود در فایل GradeBook.h مطابقت دارد. برای مثال، کامپایلر مطمئن می‌شود که **getCourseName** پارامتری نمی‌پذیرد و یک رشته برگشت می‌دهد.

۲- هر تابع عضو، اعضای داده کلاس و سایر توابع را می‌شناسد. برای مثال، خط 19 و 25 می‌توانند به متغیر **courseName** دسترسی پیدا کنند، چرا که در GradeBook.h بعنوان یک عضو داده اعلان شده است، و خطوط 13 و 32 می‌تواند توابع **setCourseName** و **getCourseName** را به ترتیب فراخوانی نمایند، چرا که هر کدامیک از آنها بعنوان یک تابع عضو کلاس در GradeBook.h اعلان شده‌اند.

```
1  // Fig. 3.12: GradeBook.cpp
2  // GradeBook member-function definitions. This file contains
3  // implementations of the member functions prototyped in GradeBook.h.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13    setCourseName( name ); // call set function to initialize courseName
14 } // end GradeBook constructor
15
16 // function to set the course name
17 void GradeBook::setCourseName( string name )
18 {
19    courseName = name; // store the course name in the object
20 } // end function setCourseName
21
22 // function to get the course name
23 string GradeBook::getCourseName()
24 {
25    return courseName; // return object's courseName
26 } // end function getCourseName
27
28 // display a welcome message to the GradeBook user
29 void GradeBook::displayMessage()
30 {
31    // call getCourseName to get the courseName
32    cout << "Welcome to the grade book for\n" << getCourseName()
33       << "!" << endl;
34 } // end function displayMessage
```

**شکل ۱۲-۳ | تعاریف تابع عضو GradeBook نشاندهنده ساختار پیاده‌سازی کلاس GradeBook.**

***تست کلاس GradeBook***

برنامه شکل ۱۳-۳ همان کار دستکاری شی GradeBook بکار رفته در برنامه ۱۰-۳ را انجام می‌دهد. جداسازی واسط GradeBook از بخش پیاده‌سازی توابع عضو تاثیری در روش استفاده این کد

www.yourshop.4kia.ir

سرویس‌گیرنده از کلاس ندارد و فقط بر نحوه کامپایل برنامه و لینک آن تاثیر دارد که در مورد آن صحبت خواهیم کرد.

```
1  // Fig. 3.13: fig03_13.cpp
2  // GradeBook class demonstration after separating
3  // its interface from its implementation.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // function main begins program execution
11 int main()
12 {
13    // create two GradeBook objects
14    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
15    GradeBook gradeBook2( "CS102 Data Structures in C++" );
16
17    // display initial value of courseName for each GradeBook
18    cout<<"gradeBook1 created for course:"<< gradeBook1.getCourseName()
19       <<"\ngradeBook2 created for course:"<< gradeBook2.getCourseName()
20       << endl;
21    return 0; // indicate successful termination
22 } // end main
```

```
gradeBook1 created for course : CS101 Introduction to C++ Programming
gradeBook2 created for course : CS102 Data Structures in C++
```

**شکل ۱۳-۳ | کلاس GradeBook پس از جداسازی واسط از پیاده‌سازی.**

همانند برنامه شکل ۱۰-۳، خط 8 برنامه شکل ۱۳-۳ شامل فایل سرآیند GradeBook.h است و از اینرو است که کامپایلر می‌تواند از ایجاد و دستکاری صحیح شی‌های **GradeBook** در کد سرویس‌گیرنده مطمئن گردد. قبل از اجرای این برنامه، باید فایل‌های کد منبع در شکل ۱۲-۳ و ۱۳-۳ هر دو کامپایل‌شده و سپس به هم لینک گردند. فراخوانی تابع عضو در کد سرویس‌گیرنده نیازمند گره خوردن با پیاده‌سازی توابع عضو کلاس دارد، کاری که لینکر آن را انجام می‌دهد.

***فرآیند کامپایل و لینک***

دیاگرام شکل ۱۴-۳ نمایشی از فرآیند کامپایل و لینک است که نتیجه آن یک برنامه اجرایی **GradeBook** بوده که می‌تواند توسط استاد بکار گرفته شود. غالباً واسط کلاس و ساختار پیاده‌سازی توسط یک برنامه‌نویس ایجاد و کامپایل می‌شود و توسط برنامه‌نویس دیگری که کد سرویس‌گیرنده کلاس را پیاده‌سازی کرده است، بکار گرفته می‌شود. بنابر این دیاگرام نشان‌دهنده نیازهای هر دو طرف برانه‌نویس کلاس و برنامه‌نویس کد سرویس‌گیرنده است. خطوط خط‌چین در دیاگرام نشاندهنده قسمت‌های مورد نیاز برنامه‌نویس کلاس، برنامه‌نویس کد سرویس‌گیرنده و کاربر برنامه **GradeBook** هستند. [نکته: شکل ۱۴-۳ یک دیاگرام UML نیست.]

www.yourshop.4kia.ir

برنامه‌نویس کلاس مسئول ایجاد یک کلاس **GradeBook** با قابلیت استفاده مجدد در ایجاد فایل سرآیند GradeBook.h و فایل کد منبع GradeBook.cpp است که فایل سرآیند را وارد برنامه کرده (**#include**)، سپس فایل کد منبع را برای ایجاد کد شی **GradeBook** کامپایل می‌کند. برای پنهان ساختن جزئیات پیاده‌سازی توابع عضو **GradeBook**، برنامه‌نویس کلاس مبادرت به تدارک دیدن فایل سرآیند GradeBook.h برای برنامه‌نویس کد سرویس‌گیرنده و کد شی برای کلاس **GradeBook** می‌کند که حاوی دستورالعمل‌های زبان ماشین است که نشاندهنده توابع عضو می‌باشد. برنامه‌نویس کد سرویس‌گیرنده، فایل کد منبع را بطور آشکار بدست نمی‌آورد، از اینرو سرویس‌گیرنده از نحوه پیاده‌سازی توابع عضو **GradeBook** بی‌اطلاع باقی خواهد ماند.

**شکل ۱٤-۳ | فرآیند کامپایل و لینک که یک برنامه اجرایی تولید می‌کند.**

کد سرویس‌گیرنده فقط نیاز به شناخت واسط **GradeBook** به منظور نحوه استفاده از کلاس داشته و بایستی قادر به لینک آن به کد شی خود باشد. از آنجا که واسط کلاس بخشی از تعریف کلاس در فایل سرآیند GradeBook.h است، باید برنامه‌نویس کد سرویس‌گیرنده به این فایل دسترسی داشته و آن را در فایل کد منبع سرویس‌گیرنده وارد سازد (**#include**). زمانیکه کد سرویس‌گیرنده کامپایل می‌شود، کامپایلر از تعریف کلاس در GradeBook.h برای اطمینان از اینکه تابع **main** مبادرت به ایجاد و دستکاری صحیح شی‌های کلاس **GradeBook** می‌کند، استفاده می‌نماید. برای ایجاد برنامه اجرایی **GradeBook** قابل استفاده برای استاد (مربی)، آخرین مرحله لینک بصورت زیر است

۱- کد شی برای تابع **main** (یعنی، کد سرویس‌گیرنده)

۲- کد شی برای کلاس پیاده‌سازی‌کننده تابع عضو کلاس **GradeBook**

۳- کد شی کتابخانه استاندارد C++ برای کلاس‌های C++ (همانند **string**) بکار رفته توسط برنامه‌نویس پیاده‌سازی‌کننده کلاس و برنامه‌نویس کد سرویس‌گیرنده.

خروجی لینکر، برنامه اجرایی GradeBook است که استاد می‌تواند با استفاده از آن نمرات دانشجویان را مدیریت نماید. برای کسب اطلاعات بیشتر در ارتباط با کامپایل برنامه‌های با چند فایل منبع، به مستندات کامپایلر خود مراجعه کنید.

## ۱۰-۳ اعتبارسنجی داده با توابع set

در بخش ۶-۳ به معرفی توابع set پرداختیم که به سرویس‌گیرنده‌های کلاس اجازه تغییر در مقدار یک عضو داده **private** را می‌دادند. در برنامه شکل ۵-۳، کلاس **GradeBook** مبادرت به تعریف تابع عضو **setCourseName** کرده که فقط مقدار دریافتی از پارامتر **name** خود را به عضو داده **courseName**

www.yourshop.4kia.ir

تخصیص می‌دهد. این عضو داده مطمئن نیست که نام دورهٔ دریافتی مطابق با یک فرمت مشخص یا معتبر است.

همانطوری‌که در ابتدا بحث مطرح کردیم، فرض می‌کنیم که دانشگاه می‌تواند از برگه ثبت‌نام دانشجو که در آن اسامی دوره فقط 25 کاراکتر یا کمتر طول دارند، چاپ بگیرد. اگر دانشگاه از سیستمی استفاده نماید که حاوی شی‌های **GradeBook** برای تولید رونوشت ثبت‌نامی است، باید کاری کنیم که کلاس **GradeBook** مطمئن شود که عضو داده **courseName** هرگز بیش از 25 کاراکتر نخواهد داشت. برنامه شکل‌های ۱۵-۳ الی ۱۷-۳ سبب افزایش قابلیت تابع عضو **setCourseName** برای انجام فرآیند اعتبارسنجی می‌شوند.

***تعریف کلاس GradeBook***

توجه کنید که تعریف کلاس **GradeBook** در شکل ۱۵-۳ و واسط آن، یکسان با شکل ۱۱-۳ است. از آنجا که واسط بدون تغییر باقی مانده، سرویس‌گیرنده‌های این کلاس به هنگام تغییر در تعریف تابع **setCourseName**، نیازی به اصلاح نخواهند داشت. این ویژگی سبب می‌شود که سرویس‌گیرنده‌ها از مزیت کلاس ارتقاء یافته **GradeBook** به آسانی و با لینک کد سرویس‌گیرنده به کد شی **GradeBook** ارتقاء یافته، برخوردار شوند.

***اعتبارسنجی نام دوره با تابع عضو setCourseName***

ارتقاء و بهبود کلاس **GradeBook** در تعریف تابع عضو **setCourseName** صورت می‌گیرد (شکل ۱۶-۳، خطوط 31-18). عبارت **if** در خطوط 21-20 تعیین می‌کند که آیا پارامتر **name** حاوی نام یک دوره معتبر (رشته‌ای به طول 25 کاراکتر یا کمتر) است یا خیر.

اگر نام دوره معتبر باشد، خط 21 مبادرت به ذخیره نام دوره در عضو داده **courseName** می‌کند. به عبارت **name.length()** در خط 20 توجه کنید. این عبارت فراخوانی یک تابع عضو همانند **myGradeBook.displayMessage()** است. کلاس **string** متعلق به کتابخانه استاندارد C++ دارای تابع عضوی بنام **length** است که تعداد کاراکترهای موجود در یک شی **string** را برگشت می‌دهد. پارامتر **name** یک شی از نوع رشته (*string*) است و از اینرو فراخوانی **name.length()** تعداد کاراکترهای موجود در **name** را برگشت می‌دهد. اگر این مقدار کمتر یا برابر 25 باشد، نام دریافتی معتبر بوده و خط 21 اجرا می‌شود.

```
1  // Fig. 3.15: GradeBook.h
2  // GradeBook class definition presents the public interface of
3  // the class. Member-function definitions appear in GradeBook.cpp.
4  #include <string> // program uses C++ standard string class
5  using std::string;
6
```

www.yourshop.4kia.ir

```
7  // GradeBook class definition
8  class GradeBook
9  {
10 public:
11    GradeBook(string);//constructor that initializes a GradeBook object
12    void setCourseName(string);//function that sets the course name
13   string getCourseName();//function that gets the course name
14    void displayMessage(); //function that displays a welcome message
15 private:
16    string courseName;//course name for this GradeBook
17 }; // end class GradeBook
```

**شکل ۱۵-۳ | تعریف کلاس GradeBook.**

```
1  // Fig. 3.16: GradeBook.cpp
2  // Implementations of the GradeBook member-function definitions.
3  // The setCourseName function performs validation.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13    setCourseName( name ); // validate and store courseName
14 } // end GradeBook constructor
15
16 // function that sets the course name;
17 // ensures that the course name has at most 25 characters
18 void GradeBook::setCourseName( string name )
19 {
20    if ( name.length() <= 25 ) // if name has 25 or fewer characters
21       courseName = name; // store the course name in the object
22
23    if ( name.length() > 25 ) // if name has more than 25 characters
24    {
25       // set courseName to first 25 characters of parameter name
26       courseName = name.substr( 0, 25 ); // start at 0, length of 25
27
28       cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
29          << "Limiting courseName to first 25 characters.\n" << endl;
30    } // end if
31 } // end function setCourseName
32
33 // function to get the course name
34 string GradeBook::getCourseName()
35 {
36    return courseName; // return object's courseName
37 } // end function getCourseName
38
39 // display a welcome message to the GradeBook user
40 void GradeBook::displayMessage()
41 {
42    // call getCourseName to get the courseName
43    cout << "Welcome to the grade book for\n" << getCourseName()
44       << "!" << endl;
45 } // end function displayMessage
```

www.yourshop.4kia.ir

**شکل ١٦-٣ | تعریف تابع عضو برای کلاس GradeBook با تابع set که مبادرت به اعتبارسنجی طول عضو داده courseName می‌کند.**

عبارت **if** در خطوط 30-23 به حالتی رسیدگی می‌کند که **setCourseName** نام یک دورهٔ نامعتبر دریافت کرده است (نامی که بیش از 25 کاراکتر طول دارد). حتی اگر پارامتر **name** بسیار طولانی باشد، می‌خواهیم که شی **GradeBook** را در یک وضعیت پایدار حفظ کنیم. وضعیتی که در آن عضو داده **courseName** حاوی یک مقدار معتبر باشد (رشته‌ای بطول 25 کاراکتر یا کمتر). از اینرو، مبادرت به کوتاه کردن نام دوره و تخصیص 25 کاراکتر اول **name** به عضو داده **courseName** می‌کنیم (البته این روش کوتاه‌سازی نام دوره چندان جالب نیست). کلاس استاندارد **string** دارای تابع عضوی بنام **substr** (کوتاه شده جمله "*substring*") است که یک شی جدید **string** با کپی کردن بخشی از شی **string** موجود، تهیه می‌کند. با فراخوانی خط 26، عبارت **name.substr(0,25)** دو عدد صحیح (0, 25) به تابع عضو **substr** شی **name** ارسال می‌شوند. این آرگومان‌ها نشاندهنده بخشی از رشته **name** هستند که **substr** آن را برگشت خواهد داد. آرگومان اول نشان‌دهنده موقعیت شروع در رشته اصلی است که کاراکترها از آن موقعیت شروع به کپی شدن خواهند کرد، توجه کنید که موقعیت اولین کاراکتر در هر رشته‌ای با صفر شروع می‌شود. آرگومان دوم نشان‌دهنده تعداد کاراکترهایی است که باید کپی شوند. بنابر این با فراخوانی خط 26، بیست و پنج کاراکتر از رشته **name** از موقعیت صفر برگشت داده خواهد شد. برای مثال اگر نام موجود نگهداری شده "CS101 Introduction to Programming in C++" باشد، تابع **substr** رشته "CS101 Introduction to Pro" را برگشت خواهد داد. پس از فراخوانی **substr**، خط 26 زیر رشته برگشتی توسط **substr** را به عضو داده **courseName** تخصیص می‌دهد. در این حالت، تابع عضو **setCourseName** مطمئن خواهد بود که **courseName** همیشه یک رشته بطول 25 کاراکتر یا کمتر خواهد داشت. اگر تابع عضو مجبور به کوتاه کردن نام دوره برای تبدیل آن به یک مقدار معتبر شود، خطوط 29-28 یک پیغام هشدار به نمایش در می‌آورند.

توجه کنید عبارت **if** در خطوط 30-23 حاوی دو عبارت در بدنه خود است، یکی برای تنظیم **courseName** با 25 کاراکتر اول از پارامتر **name** و یکی برای چاپ پیغام اطلاع دهنده به کاربر. مایل بودیم تا هر دو این عبارات در زمانیکه طول name طولانی‌تر از حد مجاز باشند اجرا گردند، بنابر این هر دو آنها را در درون یک جفت براکت، **{}** قرار داده‌ایم. از فصل دوم بخاطر دارید که این براکت‌ها بلوک ایجاد می‌کنند. در فصل چهارم اطلاعات بیشتری در زمینه قرار دادن عبارات مضاعف در بدنه یک عبارت کنترلی بدست خواهید آورد.

www.yourshop.4kia.ir

دقت کنید که عبارت cout در خطوط 29-28 بدون عملگر درج در ابتدای خط دوم ظاهر شده است:

```
cout<<"Name\""<<name<<"\"exceeds maximum length(25).\n"
   "Limiting courseName to first 25 characters.\n"<<endl;
```

کامپایلر C++ مبادرت به ترکیب رشته‌های لیترال مجاور هم می‌کند، حتی اگر این رشته‌ها بر روی خطوط مجزا شده از هم در برنامه قرار داشته باشند. بنابر این در عبارت فوق، کامپایلر C++ شروع به ترکیب رشته‌های لیترال "exceeds maximum length(25).\n"\" و "Limiting courseName to first 25 characters.\n" در یک رشته لیترال واحد در خروجی همانند خطوط 29-28 برنامه شکل ۱۶-۳ می‌کند. چنین رفتاری امکان می‌دهد تا رشته‌های طولانی را در چندین خط قرار دهید بدون اینکه مجبور به استفاده از چندین عملگر درج باشید.

```
1  // Fig. 3.17: fig03_16.cpp
2  // Create and manipulate a GradeBook object; illustrate validation.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "GradeBook.h" // include definition of class GradeBook
8
9  // function main begins program execution
10 int main()
11 {
12    // create two GradeBook objects;
13    // initial course name of gradeBook1 is too long
14    GradeBook gradeBook1( "CS101 Introduction to Programming in C++" );
15    GradeBook gradeBook2( "CS102 C++ Data Structures" );
16
17    // display each GradeBook's courseName
18    cout << "gradeBook1's initial course name is: "
19       << gradeBook1.getCourseName()
20       << "\ngradeBook2's initial course name is: "
21       << gradeBook2.getCourseName() << endl;
22
23    // modify myGradeBook's courseName (with a valid-length string)
24    gradeBook1.setCourseName( "CS101 C++ Programming" );
25
26    // display each GradeBook's courseName
27    cout << "\ngradeBook1's course name is: "
28       << gradeBook1.getCourseName()
29       << "\ngradeBook2's course name is: "
30       << gradeBook2.getCourseName() << endl;
31    return 0; // indicate successful termination
32 } // end main
```

```
Name "CS101 Introduction to Programming in C++" exceeds maximum length
(25).
Limiting courseName to first 25 characters.

gradeBook1's initial course name is : CS101 C++ Introduction to pro
gradeBook2's initial course name is : CS102 C++ Data Structures

gradeBook1's course name is : CS101 C++ Programing
```

www.yourshop.4kia.ir

```
gradeBook2's course name is : CS102 C++ Data Structures
```

**شکل ۱۷-۳ | ایجاد و دستکاری شی GradeBook که در آن نام دوره محدود به 25 کاراکتر است.**

***تست کلاس GradeBook***

برنامه شکل ۱۷-۳ به توصیف نسخه اصلاح شده کلاس **GradeBook** (برنامه‌های ۱۵-۳ و ۱۶-۳) در زمینه اعتبارسنجی است. در خط 14 یک شی **GradeBook** بنام **gradeBook1** ایجاد شده است. بخاطر دارید که سازنده **GradeBook** تابع عضو **setCourseName** را برای مقداردهی اولیه عضو داده **courseName** فراخوانی می‌کرد. در نسخه‌های قبلی این کلاس، مزیت فراخوانی **setCourseName** توس‌ط سازنده مورد بررسی قرار نگرفت، که در اینجا به بررسی آن می‌پردازیم. سازنده از مزیت اعتبارسنجی تدارک دیده شده توسط **setCourseName** برخوردار می‌شود. سازنده بجای اینکه کد اعتبارسنجی خود را تکثیر یا تکرار کند، فقط مبادرت به فراخوانی ساده **setCourseName** می‌کند. زمانیکه خط 14 از برنامه شکل ۷-۳ نام اولیه دورهٔ "CS101 Introduction to Programming in C++" را به سازنده **GradeBook** ارسال می‌کند، سازنده این مقدار را به **setCourseName** انتقال می‌دهد، جائیکه مقداردهی اولیه واقعی در آنجا اتفاق می‌افتد. به دلیل اینکه نام دورهٔ حاوی بیش از 25 کاراکتر است، بدنه دومین عبارت **if** اجرا می‌شود، و در نتیجه **courseName** با 25 کاراکتر اول نام دوره یعنی "CS101 Introduction to Pro" مقداردهی می‌گردد. توجه کنید که خروجی در شکل ۱۷-۳ حاوی پیغام هشدار ایجاد شده توسط خطوط 29-28 از شکل ۱۶-۳ در تابع عضو **setCourseName** است. خط 15 یک شی دیگر از **GradeBook** بنام **gradeBook2** ایجاد می‌کند. نام یک دوره معتبر که دقیقا برابر 25 کاراکتر است به سازنده ارسال شده است.

خطوط 21-18 از شکل ۱۷-۳ نام دوره کوتاه شده برای **gradeBook1** و نام دوره برای **gradeBook2** را به نمایش در می‌آورند. خط 24 مستقیماً تابع عضو **setCourseName** شی **gradeBook1** را فراخوانی می‌کند تا نام دوره در شی **GradeBook** را به یک نام کوتاهتر تغییر دهد تا دیگر نیازی به کوتاه‌سازی آن نباشد سپس خطوط 30-27 اسامی دوره را مجدداً به نمایش در می‌آورند.

***نکاتی دیگر در ارتباط توابع set***

یک تابع *set* **public** همانند **setCourseName** بایستی بدقت مراقب هرگونه تغییر در مقدار یک عضو داده (همانند **courseName**) باشد تا مطمئن گردد که مقدار جدید مناسب این ایتم داده است. برای مثال، مبادرت به تنظیم روزی از ماه به 37 نبایستی قبول شود، مباردت به تنظیم وزن یک شخص با صفر یا مقدار منفی برای آن نبایستی پذیرفته شود یا در صورتیکه حداکثر امتیاز یا نمره شخصی می‌تواند بین صفر تا 100 باشد، نبایستی امتیاز 185 تایید شود.

www.yourshop.4kia.ir

**مهندسی نرم‌افزار**

*با ایجاد عضوهای دادهٔ **private** و کنترل دسترسی به آنها بویژه دسترسی نوشتاری مناسب از طریق توابع عضو **public** می‌تواند در حفظ جامعیت داده کمک‌کننده باشد.*

**اجتناب از خطا**

*حفظ جامعیت داده بخودی خود بوجود نمی‌آید چرا که اعضای داده بصورت **private** ایجاد می‌شوند و از اینرو بایستی برنامه‌نویس اعتبارسنجی‌های دقیق را انجام داده و خطاها را گزارش نماید.*

**مهندسی نرم‌افزار**

*توابع عضوی که مبادرت به تنظیم مقادیر داده **private** می‌کنند بایستی مراقب صحیح بودن مقادیر جدید باشند، در صورتیکه این مقادیر معتبر نباشند، باید این توابع تنظیم‌کننده، سعی نمایند تا اعضای داده **private** در وضعیت مناسبی قرار گیرند.*

توابع set یک کلاس می‌توانند مقادیری به سرویس‌گیرنده‌های کلاس برگشت دهند تا نشان دهند که مبادرت به تخصیص یک داده نامعتبر به شی از کلاس شده است. سرویس‌گیرنده کلاس می‌تواند مبادرت به تست مقدار برگشتی از یک تابع set کند تا تعیین نماید که آیا عملیات اصلاح یا تغییر در شی موفقیت‌آمیز بوده است یا خیر و در هر دو حالت کار مقتضی را انجام دهد. در فصل ۱۶ به بررسی نحوه اطلاع‌دهی؛ سرویس‌گیرنده‌های کلاس از طریق مکانیزم رسیدگی به استثناء خواهیم پرداخت. برای اینکه برنامه ۱۵-۳ الی ۱۷-۳ را فعلاً در یک سطح ساده نگهداری کنیم، تابع **setCourseName** در برنامه ۱۶-۳ فقط مبادرت به چاپ پیغام مقتضی در صفحه نمایش می‌کند.

## ۱۱-۳ مبحث آموزشی مهندسی نرم‌افزار: شناسایی کلاس‌های موجود در مستند نیازهای ATM

در این بخش شروع به طراحی سیستم ATM می‌کنیم که در فصل دوم به معرفی آن پرداختیم. در این بخش به شناسایی کلاس‌ها می‌پردازیم که در ایجاد سیستم ATM مورد نیاز هستند و این کار را با تحلیل اسامی و اصطلاحات آمده در مستند نیازهای ATM می‌کنیم. همچنین به معرفی دیاگرام‌های کلاس UML به منظور مدل کردن روابط مابین این کلاس‌ها خواهیم پرداخت. این کار بعنوان اولین گام در تعریف ساختار سیستم از اهمیت خاصی برخوردار است.

### *شناسایی کلاس‌ها در سیستم*

فرآیند OOD خود را با شناسایی کلاس‌های مورد نیاز در ایجاد یک سیستم ATM آغاز می‌کنیم. سرانجام این کلاس‌ها را با استفاده از دیاگرام‌های کلاس UML و پیاده‌سازی این کلاس‌ها در C++ توصیف خواهیم کرد. ابتدا، نگاهی به مستند نیازها در بخش ۸-۲ می‌اندازیم تا اسامی و اصطلاحات کلیدی را که می‌توانند به ما در شناسایی کلاس‌های که در ایجاد سیستم ATM نقش دارند، پیدا کنیم.

www.yourshop.4kia.ir

امکان دارد تصمیم بگیریم که برخی از این اسامی و اصطلاحات جزو صفات کلاس‌های دیگر در سیستم باشند. همچنین می‌توانیم استنتاج کنیم که برخی از اسامی ارتباطی با بخش‌های سیستم ندارند و نیازی نیست که آنها را مدل‌سازی نمائیم. کلاس‌های اضافی می‌توانند میزان حرکت ما را در فرآیند طراحی آشکار کنند.

جدول شکل ۱۸-۳ لیستی از اسامی و اصطلاحات موجود در مستند نیازها را به نمایش در آورده است. در این لیست ترتیب از سمت چپ به راست و با توجه به ظاهر شدن این اسامی در مستند نیازها است.

| اسامی و اصطلاحات موجود در مستند نیازها | | |
|---|---|---|
| بانک | پول/ سرمایه | شماره حساب |
| ATM | صفحه نمایش | PIN |
| کاربر | صفحه کلی‌د | پایگاه داده بانک |
| مشتری | تحویل‌دار خودکار | درخواست موجودی |
| تراکنش | اسکناس 20 دلاری/ نقد | برداشت پول |
| حساب | شکاف سپرده‌گذاری | سپرده |
| موجودی | پاکت سپرده | |

**شکل ۱۸-۳ | اسامی و اصطلاحات موجود در مستند نیازها.**

کلاس‌ها را فقط برای اسامی و اصطلاحاتی که در سیستم ATM از اهمیت برخوردار هستند ایجاد می‌کنیم. نیازی به مدل کردن بانک بعنوان یک کلاس نداریم، چرا که بانک بخشی از سیستم ATM نیست، بانک فقط از ما خواسته که سیستم ATM را ایجاد کنیم. مشتری و کاربر نیز نشاندهنده موجودیت‌های خارج از سیستم هستند، این موجودیت‌ها از اهمیت برخوردار می‌باشند چرا که آنها در تعامل با سیستم قرار می‌گیرند، اما نیازی به مدل کردن آنها به عنوان کلاس‌ها در نرم‌افزار ATM نداریم. بخاطر دارید که کاربر ATM (مشتری بانک) را بصورت یک بازیگر در دیاگرام حالت شکل ۱۸-۲ به نمایش در آوردیم.

نیازی به مدل کردن «اسکناس 20 دلاری» یا «پاکت سپرده» بعنوان کلاس نیست. این موارد شی‌های فیزیکی در دنیای واقعی هستند، اما بخشی از فرآیند اتوماتیک‌سازی نمی‌باشند. می‌توانیم بقدر کفایت

www.yourshop.4kia.ir

اسکناس را در سیستم با استفاده از صفات کلاسی که پرداخت‌کننده یا تحویل‌دار اتوماتیک را مدل می‌کند، عرضه کنیم. در بخش ۱۳-۴ به تخصیص صفات به کلاس‌ها خواهیم پرداخت. برای مثال، تحویل‌دار اتوماتیک مبادرت به نگهداری تعدادی اسکناس در خود می‌کند. مستند نیازها چیزی در ارتباط با کاری که سیستم باید با پاکت سپرده‌ها پس از دریافت آنها انجام دهد، بیان نمی‌کند. می‌توانیم فرض کنیم که فقط تایید وصول پاکت صورت می‌گیرد، عملیاتی که توسط کلاس مدل شده برای شکاف سپرده انجام می‌شود. در بخش ۲۲-۶ با نحوه تخصیص عملیات به کلاس‌ها آشنا خواهید شد.

در این سیستم ATM ساده شده، نمایش مقادیر مختلف "پولی" شامل "موجودی" یک حساب، همانند صفات سایر کلاس‌ها ضروری بنظر می‌رسد. همچنین، اسامی "شماره‌حساب" و "PIN" نشاندهنده اطلاعات با ارزش در سیستم ATM هستند. این موارد از صفات مهم در یک حساب بانکی هستند. با این وجود، نشاندهنده رفتاری نمی‌باشند. از اینرو بهتر خواهد بود تا آنها را بعنوان صفات کلاس حساب مدل‌سازی کنیم.

با وجود آنکه در مستند نیازها کلمه "تراکنش" بصورت کلی بکار گرفته شده است، اما فعلاً قصد مدل کردن این مفهم کلی در تراکنش مالی را نداریم. بجای آن سه نوع تراکنش ("نمایش موجودی"، "برداشت پول" و "سپرده") را بعنوان کلاس‌های مجزا مدل می‌کنیم. این کلاس‌ها دارای صفات خاص مورد نیاز برای انجام تراکنش‌های متعلق بخود را دارا هستند. برای مثال، کلاس برداشت پول نیاز به داشتن میزان پولی دارد که کاربر می‌خواهد برداشت کند. با این وجود، کلاس موجودی، نیازی به داده‌های اضافی ندارد. علاوه بر اینها، سه کلاس تراکنشی رفتارهای منحصر به فردی را در معرض دید قرار می‌دهند. کلاس برداشت پول شامل پرداخت پول به کاربر است، در حالیکه سپرده‌گذاری مستلزم دریافت پاکت سپرده‌گذاری از سوی کاربر است. [نکته: در بخش ۱۰-۱۳، مبادرت به فاکتورگیری ویژگی‌های مشترک از تمام تراکنش‌ها بصورت یک کلاس «تراکنش» کلی خواهیم کرد. با استفاده از مفهوم کلاس‌های انتزاعی و توارث در برنامه‌نویسی شی‌گرا.]

کلاس‌های مورد نیاز سیستم را بر پایه اسامی و اصطلاحات موجود در جدول ۱۸-۳ تعیین می‌کنیم. هر کدامیک از این اسامی به یک یا چند مورد از عبارات زیر مراجعه دارند:

- ATM
- صفحه نمایش
- صفحه کلید

www.yourshop.4kia.ir

- تحویل‌دار اتوماتیک
- شکاف سپرده‌گذاری
- حساب
- پایگاه داده بانک
- نمایش موجودی (درخواست موجودی)
- برداشت پول
- سپرده‌گذاری

عناصر موجود در این لیست از قابلیت تبدیل شدن به کلاس‌های مورد نیاز در پیاده‌سازی سیستم ATM برخوردار هستند. اکنون می‌توانیم شروع به مدلسازی کلاس‌ها در سیستم خود بر پایه لیست فوق کنیم. اسامی کلاس‌ها را در فرآیند طراحی با حروف بزرگ نشان می‌دهیم (قاعده UML)، همین کار را به هنگام پیاده‌سازی طراحی به زبان C++ انجام خواهیم داد. اگر نام کلاسی بیش از یک کلمه باشد، کلمات را در کنار هم قرار می‌دهیم و هر کلمه را با حرف بزرگ شروع می‌کنیم (مثلاً **Multiple WordName**). با استفاده از این روش، مبادرت به ایجاد کلاس‌های ATM، **Screen** (صفحه نمایش)، صفحه‌کلید (**Keypad**)، تحویل‌دار خودکار یا پرداخت‌کننده پول (**CashDispencer**)، شکاف سپرده (**DepositSlot**)، حساب (**Account**)، پایگاه داده بانک (**BankDatabase**)، پرس‌وجوی میزان موجودی (**BalanceInquiry**)، برداشت پول (**Withdrawal**) و سپره (**Deposit**). سیستم خود را با استفاده از تمام این کلاس‌ها بعنوان بلوک‌های سازنده ایجاد خواهیم کرد. قبل از شروع به ایجاد بلوک‌های سازنده سیستم، بایستی درک مناسبی از روابط مابین کلاس‌ها بدست آوریم.

### *مدل‌سازی کلاس‌ها*

زبان UML امکان مدل‌سازی کلاس‌های موجود در سیستم ATM و روابط داخلی آنها را از طریق دیاگرام‌های کلاس فراهم آورده است. در شکل ۱۹-۳ کلاس **ATM** نشان داده شده است. در UML، هر کلاس بصورت یک مستطیل با سه بخش مدل می‌شود.

بخش فوقانی حاوی نام کلاس در وسط و بصورت توپر (پر رنگ) نوشته می‌شود. بخش میانی حاوی صفحات کلاس است (در بخش ۱۳-۴ و بخش ۱۱-۵ به بررسی صفات خواهیم پرداخت) بخش تحتانی حاوی عملیات کلاس است (در بخش ۲۲-۶ بحث خواهد شد). در شکل ۱۹-۳ بخش میانی و تحتانی خالی هستند، چرا که هنوز به تعیین صفات و عملیات کلاس نپرداخته‌ایم.

www.yourshop.4kia.ir

دیاگرام‌های کلاس از قابلیت نمایش روابط مابین کلاس‌های سیستم برخوردار هستند. شکل ۲۰-۳ نشاندهنده نحوه رابطه کلاس‌های **ATM** و **Withdrawal** با یکدیگر است. برای سادگی کار، در این لحظه فقط مبادرت به مدل‌سازی این زیر مجموعه می‌کنیم. دیاگرام کامل کلاس را در ادامه این بخش شاهد خواهید بود. دقت کنید که مستطیل‌های نشاندهنده کلاس‌ها در این دیاگرام به زیربخش‌ها تقسیم نمی‌شوند.

در شکل ۲۰-۳، خط مستقیم و یکپارچه دو کلاس را به هم پیوند زده، نشاندهنده یک *وابستگی* است (رابطه مابین کلاس‌ها). اعداد قرار گرفته در انتهای خط مقادیر تعدد هستند که نشان می‌دهند که چند شی از هر کلاس در وابستگی (رابطه) شرکت یا دخالت دارند.

**شکل ۱۹-۳ | نمایش کلاس در UML با استفاده از دیاگرام کلاس.**

**شکل ۲۰-۳ | دیاگرام‌های کلاس در حال نمایش وابستگی مابین کلاس‌ها.**

در این مورد، با دنبال کردن خط از یک طرف به طرف دیگر معلوم می‌شود که در هر لحظه، یک شی ATM در رابطه (وابستگی) با صفر یا یک شی **Withdrawal** شرکت دارد. اگر کاربر جاری در حال حاضر تراکنشی انجام ندهد یا تقاضای یک تراکنش از نوع دیگری را نماید، صفر، و در صورتیکه تقاضای برداشت پول (withdrawal) کند، مقدار 1 بکار گرفته می‌شود. زبان UML قادر به مدل کردن انواع تعدد یا کثرت است. در جدول شکل ۲۱-۳ لیستی از انواع تعددها و مفهوم آنها آورده شده است.

| نماد | مفهوم |
|---|---|
| **0** | هیچ- اصلاً |
| **1** | یک |
| **m** | مقدار صحیح |
| **0..1** | صفر یا یک |
| **m,n** | m یا n |
| **m..n** | حداقل m، اما نه بیشتر از n |
| ***** | هر مقدار غیرمنفی (صفر یا بیشتر) |

www.yourshop.4kia.ir

| | |
|---|---|
| **0..*** | صفر یا بیشتر (همان‌د *) |
| **1..*** | یک یا بیشتر |

**شکل ۲۱-۳ | انواع تعدد.**

وابستگی می‌تواند نام داشته باشد. برای مثال، کلمه **Executes** در بالای خط متصل‌کننده کلاس‌های **ATM** و **Withdrawal** در شکل ۲۰-۳ نشان‌دهنده نام این وابستگی (ارتباط) است. این بخش از دیاگرام بصورت زیر معنی می‌شود، "یک شی از کلاس **ATM** صفر یا یک شی از کلاس **Withdrawal** را به اجرا در می‌آورد." توجه نمایید که اسامی وابستگی، حالت هدایت‌کننده و نشان‌دهنده جهت هستند، همانند جهت فلش توپر، از اینرو جهت تفسیر دیاگرام مهم است، در صورتی که برای مثال تفسیر را از سمت راست به چپ انجام دهیم، جمله‌ای به این مضمون خواهیم داشت "صفر یا یک شی از کلاس **Withdrawal** یک شی از کلاس **ATM** را به اجرا درمی‌آورد."

کلمه **currentTransaction** در کنار و زیرخط وابستگی **Withdrawal** در شکل ۲۰-۳، نام یک *نقش* (*role*) است، که هویت‌دهنده نقشی است که شی **Withdrawal** در رابطه خود با **ATM** بازی می‌کند. نام نقش، هدف و منظوری را به رابطه مابین کلاس‌ها اضافه می‌کند، و اینکار را با شناسایی نقشی که کلاس در بافت رابطه ایفاء می‌کند، انجام می‌دهد. یک کلاس می‌تواند چندین نقش در همان سیستم بازی کند. برای مثال، در یک سیستم پرسنلی دانشگاه یک نفر می‌تواند نقش یک «پورفسور» را به هنگام رابطه داشتن با دانشجویان بازی کند. امکان دارد همان شخص نقش «همکار» در کنار پورفسور دیگر و نقش «مربی» را به هنگام مسابقات دانشجویی بازی کند. در شکل ۲۰-۳، نام نقش **currentTransaction** بر این نکته دلالت دارد که شی **Withdrawal** در اجرای (**Executes**) رابطه با یک شی از کلاس **ATM** دخالت دارد و این کلاس در پردزاش تراکنش جاری عمل می‌کند. در سایر محیط‌ها امکان دارد یک شی **Withdrawal** نقش‌های متفاوتی بخود گیرد. زمانیکه مفهوم رابطه در دیاگرام به قدر کافی گویا باشد، از اسامی نقش در دیاگرام‌های کلاس استفاده نمی‌شود.

علاوه بر وجود روابط ساده، وابستگی می‌تواند تصریح‌کننده روابط بسیار پیچیده و مرکب باشد، همانند زمانیکه شی‌های از یک کلاس با شی‌های از کلاس‌های دیگر ترکیب شوند. به سیستم ATM واقعی توجه کنید. سازنده ATM باید چه قسمت‌های را در کنار هم قرار دهد تا ATM قادر به کار باشد؟ مستند نیازها به ما می‌گوید که ATM دستگاهی متشکل از یک صفحه نمایش، یک صفحه‌کلید، یک پرداخت‌کننده یا تحویل‌دار خودکار و یک شکاف سپرده‌گذاری است.

www.yourshop.4kia.ir

در شکل ۲۲-۳ *لوزی‌های توپر* متصل شده به خطوط وابستگی کلاس **ATM** بر این نکته دلالت دارند که کلاس **ATM** دارای یک رابطه *ترکیبی* با کلاس‌های **Screen**، **Keypad**، **CashDispenser** و DepositSlot است. ترکیب مفهومی از یک رابطه کامل/بخش (قطعه) است. کلاسی که دارای نماد ترکیب (لوزی توپر) در انتها خط وابستگی خود می‌باشد، کامل بوده (در این مورد **ATM**) و کلاس‌های قرار گرفته در آن سوی خطوط وابستگی، بخش یا قطعه می‌باشند، در این مورد کلاس‌های **Screen**، **Keypad**، **CashDispenser** و **DepositSlot**. ترکیب بنمایش درآمده در شکل ۲۲-۳ نشان می‌دهد که یک شی از کلاس **ATM** از یک شی از کلاس **Screen**، یک شی از کلاس **CashDispenser**، یک شی از کلاس **Keypad** و یک شی از کلاس **DepositSlot** تشکیل یافته است. ATM «*دارای*» یک صفحه نمایش، یک صفحه‌کلید، یک تحویل‌دار خودکار و شکاف سپرده است. رابطه «*داشتن*» تعریف‌کننده ترکیب است. در فصل سیزدهم نشان خواهیم داد که رابطه «*است یک*» تعریف‌کننده توارث است.

بر طبق مشخصات UML، رابطه ترکیب دارای خصوصیات زیر است:

۱- فقط یک کلاس در رابطه می‌تواند نشاندهنده رابطه کامل باشد (لوزی می‌تواند فقط در انتهای یک طرف خط وابستگی قرار گرفته باشد). برای مثال خواه صفحه نمایش بخشی از ATM باشد یا ATM بخشی از صفحه نمایش باشد، صفحه نمایش و ATM هر دو نمی‌توانند نشاندهنده رابطه کامل (سراسری) باشند.

۲- قطعات یا بخش‌ها در رابطه ترکیب تا زمانیکه رابطه کامل وجود دارد، وجود خواهند داشت و این رابطه کامل مسئول ایجاد و نابود کردن قطعات متعلق بخود است. برای مثال، اقدام به ایجاد یک ATM مستلزم ساخت قطعات آن است. علاوه بر این، اگر ATM نابود گردد، صفحه نمایش، صفحه‌کلید، پرداخت‌کننده اتوماتیک و شکاف سپرده آن نیز نابود خواهند شد.

۳- یک قطعه می‌تواند فقط در یک زمان متعلق به یک رابطه کامل باشد، اگر چه امکان دارد قطعه‌ای حذف و به رابطه کامل دیگری متصل گردد.

لوزی‌های بکار رفته در دیاگرام کلاس ما بر این نکته دلالت دارند که در رابطه ترکیبی این سه خصیصه وجود دارند. اگر رابطه «داشتن» قادر به برآوردن یکی از چند ضابطه فوق نباشد، UML بر استفاده از لوزی‌های توخالی متصل شده به انتهای خطوط وابستگی تصریح می‌کند تا نشاندهنده *اجتماع* یا *تراکم* باشند. اجتماع نسخه ضعیف‌تر ترکیب است. برای مثال، یک کامپیوتر و مانیتور در رابطه اجتماع قرار دارند، کامپیوتر «دارای» مانیتور است، اما دو قطعه می‌توانند بصورت مستقل وجود داشته باشند و

www.yourshop.4kia.ir

همان مانیتور می‌تواند در یک زمان به چندین کامپیوتر متصل گردد، از اینرو اینحالت نقض خصیصه دوم و سوم ترکیب است.

**شکل ۲۲-۳ | دیاگرام کلاس نشاندهنده رابطه ترکیب.**

در شکل ۲۳-۳ نمایشی از دیاگرام کلاس سیستم ATM ارائه شده است. این دیاگرام اکثر کلاس‌هایی که در ابتدای این بخش شناسایی کرده بودیم را به همراه وابستگی مابین آنها را که از مستند نیازها استنتاج کرده‌ایم، مدل کرده است. [نکته: کلاس‌های **BalanceInquiry** و **Deposit** در رابطه مشابهی با کلاس **Withdrawal** شرکت دارند، از اینرو برای حفظ سادگی دیاگرام، آنها را در نظر نگرفته‌ایم. در فصل ۱۳، دیاگرام کلاس را گسترش داده و تمام کلاس‌های موجود در سیستم ATM را وارد آن می‌کنیم.]

شکل ۲۳-۳ نمایشی از مدل گرافیکی ساختار سیستم ATM است. این دیاگرام کلاس حاوی کلاس‌های **BankDatabase** و **Account** و چندین رابطه (وابستگی) است که در شکل‌های ۲۰-۳ یا ۲۲-۳ عرضه نشده بودند. دیاگرام کلاس نشان می‌دهد که کلاس ATM دارای یک رابطه یک به یک با کلاس **BankDatabase** است، یک شی **ATM** مبادرت به تصدیق کاربران در برابر یک شی **BankDatabase** می‌کند. همچنین در شکل ۲۳-۳، مبادرت به مدل کردن این واقعیت کرده‌ایم که پایگاه بانک حاوی اطلاعاتی در ارتباط با حساب‌های متعدد است، یک شی از کلاس **BankDatabase** در یک رابطه ترکیبی با صفر یا چندین شی از کلاس **Account** شرکت دارد. از جدول ۲۱-۳ بخاطر دارید که مقدار تعدد یا کثرت ***..0** در سمت وابستگی **Account** مابین کلاس **BankDatabase** و کلاس **Account** بر این نکته دلالت دارد که صفر یا چندین شی از کلاس **Account** بخشی در رابطه را تشکیل می‌دهند. کلاس **BankDatabase** دارای رابطه یک به چند با کلاس **Account** است. **BankDatabase** مبادرت به ذخیره حساب‌های (*accounts*) متعدد در خود می‌کند. به همین ترتیب، کلاس **Account** دارای رابطه چند به یک با کلاس **BankDatabase** است، چرا که حساب‌های متعدد در پایگاه داده بانک (*bank database*) ذخیره می‌شوند. [نکته: از جدول شکل ۲۱-۳ بخاطر دارید که مقدار * معادل با ***..0** است. برای افزایش وضوح دیاگرام‌های کلاس از نماد ***..0** استفاده کرده‌ایم.]

**شکل ۲۳-۳ | دیاگرام کلاس برای مدل کردن سیستم ATM.**

همچنین شکل ۲۳-۳ نشان می‌دهد که اگر کاربر مبادرت به برداشت پول کند، «یک شی از کلاس **Withdrawal** به موجودی حساب دسترسی پیدا کرده/آنرا از طریق یک شی از کلاس **BankDatabase** تغییر می‌دهد.» می‌توانیم یک رابطه مستقیم مابین کلاس **Withdrawal** و کلاس **Account** ایجاد کنیم. با این وجود، مستند نیازها شرح می‌دهد که «**ATM** باید با پایگاه داده اطلاعات حساب بانک در تعامل قرار داشته باشد» تا تراکنش‌ها قابل انجام باشند. حساب بانکی حاوی اطلاعات حساس بوده و مهندسان

www.yourshop.4kia.ir

سیستم بایستی همیشه مراقب امنیت داده افراد به هنگام طراحی سیستم باشند. از اینرو، فقط **BankDatabase** می‌تواند مبادرت به دسترسی و اعمال تغییر مستقیم در یک حساب کند. تمام قسمت‌های دیگر سیستم باید با پایگاه داده در تعامل قرار گیرند تا بتوانند اطلاعاتی بدست آورده یا حساب را به روز نمایند.

همچنین دیاگرام کلاس در شکل ۲۳-۳ مبادرت به مدل کردن رابطه موجود مابین کلاس **Withdrawal** و کلاس‌های **Screen**، **CashDispenser** و **Keypad** می‌کند. تراکنش برداشت پول شامل اعلان پیغامی به کاربر برای تعیین میزان پول برداشتی و دریافت ورودی عددی است. این اعمال به ترتیب مستلزم استفاده از صفحه نمایش و صفحه کلید است. علاوه بر اینها، پرداخت پول نقد به کاربر مستلزم دسترسی به تحویل‌دار خودکار (پرداخت‌کننده پول خودکار) می‌باشد.

کلاس‌های **BalanceInquiry** و **Deposit** که در شکل ۲۳-۳ آورده نشده‌اند، دارای چندین رابطه با کلاس‌های دیگر در سیستم ATM هستند. همانند کلاس **Withdrawl**، هر کدامیک از این کلاس‌ها با کلاس‌های **ATM** و **BankDatabase** دارای رابطه (وابستگی) هستند. یک شی از کلاس **BalanceInquiry** دارای رابطه‌ای با یک شی از کلاس **Screen** برای نمایش میزان موجودی در حساب یک کاربر نیز است. کلاس **Deposit** با کلاس‌های **Screen**، **Keypad** و **DepositSlot** در ارتباط است. همانند برداشت پول، تراکنش سپرده‌گذاری مستلزم استفاده از صفحه‌نمایش و صفحه‌کلید برای نمایش پیغامی به کاربر و دریافت ورودی است. برای دریافت پاکت سپرده، یک شی از کلاس **Deposit** به شکاف سپرده دسترسی پیدا می‌کند.

اکنون کلاس‌های موجود در سیستم ATM خود را شناسایی کرده‌ایم. در بخش ۱۳-۴ به تعیین صفات هر کدامیک از این کلاس‌ها خواهیم پرداخت. در بخش ۱۱-۵ از این صفات برای بررسی نحوه تغییر عملکرد سیستم در زمان استفاده می‌کنیم. در بخش ۲۲-۶ به تعیین عملیاتی که کلاس‌ها در سیستم انجام خواهند داد، می‌پردازیم.

### *تمرینات خودآزمایی مبحث آموزشی مهندسی نرم‌افزار*

۱-۳ فرض کنید کلاسی بنام **Car** داریم که نشان‌دهنده یک اتومبیل است. در مورد قسمت‌های مختلف آن که سازنده باید درکنار هم قرار دهد تا یک اتومبیل کامل ایجاد گردد، فکر کنید. یک دیاگرام کلاس (شبیه به شکل ۲۲-۳) ایجاد کنید که برخی از روابط ترکیبی موجود در کلاس **Car** را مدل‌سازی کند.

www.yourshop.4kia.ir

۳-۲ فرض کنید کلاسی بنام **File** داریم که نشاندهنده یک مستند الکترونیکی بر روی یک سیستم کامپیوتری منفرد (بدون اتصال به شبکه) است که توسط کلاس **Computer** نشان داده می‌شود، قرار دارد. چه نوع رابطه (وابستگی) مابین کلاس **Computer** و کلاس **File** وجود دارد؟

a) کلاس **Computer** دارای رابطه یک به یک با کلاس **File** است.

b) کلاس **Computer** دارای رابطه چند به یک با کلاس **File** است.

c) کلاس **Computer** دارای رابطه یک به چند با کلاس **File** است.

d) کلاس **Computer** دارای رابطه چند به چند با کلاس **File** است.

۳-۳ تعیین کنید که آیا عبارت زیر صحیح است یا خیر و در صورتیکه اشتباه باشد علت را توضیح دهید: به یک دیاگرام کلاس UML که بخش دوم و سوم آن مدل نشده‌اند گفته می‌شود یک دیاگرام ادغام شده است.

۳-۴ دیاگرام کلاس شکل ۳-۲۳ را برای وارد کردن کلاس **Deposit** بجای کلاس **Withdrawal** تغییر دهید.

## *پاسخ خودآزمایی مبحث مهندسی نرم‌افزار*

۳-۱ [نکته: پاسخ‌های می‌توانند متفاوت باشند.] شکل ۳-۲۴ نشاندهنده دیاگرام کلاسی است که برخی از روابط ترکیبی در کلاس **Car** را عرضه می‌کند.

۳-۲ c. [نکته: در یک کامپیوتر شبکه، این رابطه می‌تواند بصورت چند به چند باشد.]

۳-۳ صحیح.

۳-۴ شکل ۳-۲۵ نشاندهنده یک دیاگرام کلاس برای **ATM** شامل کلاس **Deposit** بجای کلاس **Withdrawal** (همانند شکل ۳-۲۳) است. توجه کنید که **Deposit** به **CashDispenser** دسترسی ندارد، اما به **DepositSlot** دسترسی دارد.

**شکل ۳-۲٤ | دیاگرام کلاس نشاندهنده رابطه ترکیبی در کلاس Car.**

**شکل ۳-۲۵ | دیاگرام کلاس سیستم ATM حاوی کلاس Deposit.**

## خودآزمایی

**۳-۱** جاهای خالی را با عبارت مناسب پر کنید:

a) نقشه ترسیمی یک خانه همانند یک _____ برای یک کلاس است.

b) تعریف هر کلاس حاوی کلمه کلیدی _____ و بدنبال آن نام کلاس است.

www.yourshop.4kia.ir

c) تعریف کلاس در فایلی با پسوند ـــــــــ ذخیره می‌گردد.

d) هر پارامتر در سرآیند یک تابع بایستی توسط ـــــــــ و ـــــــــ مشخص گردد.

e) زمانیکه هر شی از یک کلاس مبادرت به نگهداری کپی از صفات خود می‌کند، به متغیری که نشاندهنده صفات است، ـــــــــ گفته می‌شود.

f) کلمه کلیدی **public**، یک ـــــــــ است.

g) نوع برگشتی ـــــــــ بر این نکته دلالت دارد که تابع وظیفه خود را انجام داده اما پس از انجام وظیفه خود مقداری برگشت نمی‌دهد.

h) تابع ـــــــــ از کتابخانه <**string**> مبادرت به قرائت کاراکترها تا رسیدن به کاراکتر خط جدید می‌کند، سپس این کاراکترها را به رشته مشخص‌شده کپی می‌نماید.

i) به هنگام تعریف تابع عضو در خارج از تعریف کلاس، بایستی سرآیند تابع شامل نام کلاس و ـــــــــ، بدنبال نام تابع برای «گره زدن» تابع عضو به تعریف کلاس باشد.

j) فایل کد منبع و سایر فایل‌های که از یک کلاس استفاده می‌کنند می‌توانند از طریق رهنمود پیش‌پردازنده ـــــــــ سرآیند فایل کلاس را شامل گردند.

**۳-۲** تعیین کنید کدامیک از عبارات زیر صحیح و کدامیک اشتباه است.

a) بطور قراردادی، اسامی تابع با یک حرف بزرگ و تمام کلمات متعاقب آن در نام با حرف بزرگ شروع می‌شوند.

b) پرانتزهای خالی پس از نام تابع در یک نمونه اولیه تابع نشان می‌دهند که تابع به هیچ پارامتری برای انجام وظیفه خود نیاز ندارد.

c) اعضای داده یا توابع عضو اعلان شده با تصریح‌کننده دسترسی **private** در دسترسی توابع عضو کلاسی قرار دارند که در آن اعلان شده‌اند.

d) متغیرهای اعلان شده در بدنه یک تابع عضو خاص بعنوان اعضای داده شناخته می‌شوند و می‌توانند در تمام توابع عضو کلاس بکار گرفته شوند.

e) بدنه هر تابع توسط یک براکت چپ و راست ({ و }) تعیین می‌شود.

f) هر فایل کد منبع که حاوی **int main()** است می‌تواند در اجرای برنامه بکار گرفته شود.

g) نوع آرگومان‌های موجود در یک تابع فراخوانی شده بایستی با نوع پارامترهای متناظر در لیست پارامتری نوع اولیه تابع مطابقت داشته باشد.

**۳-۳** تفاوت موجود مابین یک متغیر محلی و عضو داده در چیست؟

**۳-٤** هدف از پارامتر تابع چیست؟ تفاوت موجود مابین یک پارامتر و آرگومان را توضیح دهید.

## پاسخ خودآزمایی

**۳-۱** a) شی b) **class**. c) نوع، نام. d. e ) عضو داده. f) تصریح‌کننده دسترسی. g) **void** h) **getline** i)عملگر تفکیک قلمرو باینری (::) j) **#include**

www.yourshop.4kia.ir

**۳-۲** a) اشتباه. بطور قراردادی، اسامی توابع با حرف کوچک شروع و تمام کلمات متعاقب آن در نام با حرف بزرگ آغاز می‌شوند. b) صحیح. c) صحیح. d) اشتباه. چنین متغیرهای، متغیرهای محلی نامیده می‌شوند و می‌توانند فقط در تابع عضوی که در آن اعلام شده‌اند بکار گرفته شوند. e) صحیح f) صحیح. g) صحیح.

**۳-۳** متغیر محلی در بدنه یک تابع اعلان می‌شود و می‌تواند فقط از نقطه‌ای که تعریف شده تا رسیدن به براکت خاتمه بکار گرفته شود. عضو داده در تعریف کلاس اعلان می‌شود اما در بدنه توابع عضو کلاس قرار ندارد. هر شی (نمونه) یک کلاس دارای یک کپی متمایز از اعضای داده کلاس است. همچنین اعضای داده برای تمام عضو کلاس در دسترس هستند.

**۳-٤** پارامتر نشاندهنده اطلاعات اضافی است که تابع برای انجام وظیفه خود به آن نیاز دارد. هر پارامتر مورد نیاز تابع در سرآیند تابع جای داده می‌شود. آرگومان مقداری است که در فراخوانی تابع تدارک دیده می‌شود. زمانیکه تابع فراخوانی می‌شود، مقدار آرگومان به پارامتر تابع ارسال می‌شود، از اینروست که تابع می‌تواند وظیفه خود را انجام دهد.

## تمرینات

**۳-۵** تفاوت موجود مابین نمونه اولیه تابع و تعریف تابع را بیان کنید.

**۳-٦** سازنده پیش‌فرض چیست؟ اگر کلاسی دارای فقط یک سازنده پیش‌فرض ضمنی باشد، اعضای داده شی چگونه مقداردهی اولیه خواهند شد؟

**۳-۷** منظور از عضو داده چیست؟

**۳-۸** سرآیند فایل چیست؟ فایل کد منبع چیست؟ هدف از هر یک را توضیح دهید.

**۳-۹** توضیح دهید چگونه برنامه از کلاس **string** بدون اعلان **using** استفاده می‌کند.

**۳-۱۰** توضیح دهید چرا کلاسی مبادرت به تدارک دیدن یک تابع **set** و **get** برای کار با عضو داده می‌کند.

**۳-۱۱** (اصلاح کلاس **GradeBook**). کلاس **GradeBook** در شکل‌های ۳-۱۱ و ۳-۱۲ را بصورت زیر تغییر دهید یا اصلاح کنید:

a) یک عضو داده رشته‌ای دیگر اضافه کنید که نشاندهنده نام استاد دوره باشد.

b) یک تابع **set** برای تغییر دادن نام استاد و یک تابع **get** برای بازیابی آن در نظر بگیرید.

c) سازنده را با دو پارامتر، یکی برای نام دورهٔ و دیگری برای نام استاد، تغییر دهید.

d) تابع عضو **displayMessage** را به نحوی تغییر دهید که ابتدا پیغام خوش‌آمدگویی و نام دوره را چاپ کرده و سپس جمله "This course is presented by:" را چاپ و بدنبال آن نام استاد را به نمایش در آورد. کلاس اصلاح شده خود را در برنامه تست بکار گیرید تا قابلیت‌های جدید کلاس عرضه گردد.

**۳-۱۲** (کلاس **Account**) کلاسی بنام **Account** ایجاد کنید که در بانک از آن برای نمایش حساب بانکی مشتری استفاده شود. این کلاس بایستی شامل یک داده عضو از نوع **int** برای نمایش میزان موجودی حساب باشد. [نکته: در

www.yourshop.4kia.ir

فصل‌های بعدی از اعداد اعشاری استفاده خواهیم کرد.] این کلاس باید یک سازنده داشته باشد که موجودی اولیه را دریافت و با استفاده از آن مبادرت به مقداردهی اولیه عضو داده نماید. همچنین سازنده باید میزان موجودی اولیه را اعتبارسنجی کند و مطمئن گردد که این مقدار بزرگتر یا برابر صفر است. اگر چنین نباشد، موجودی را با صفر تنظیم کند و پیغام خطا را به نمایش در آورده تا نشان دهد که موجودی اولیه معتبر نیست. همچنین کلاس باید سه تابع عضو تدارک دیده باشد. تابع عضو **credit** باید مقداری پول به موجودی جاری اضافه کند. تابع عضو **debit** باید از حساب (**Account**) برداشت کند و مطمئن باشد که میزان برداشتی از میزان موجودی حساب تجاوز نکند. اگر چنین باشد، باید موجودی دست نخورده باقی بماند و تابع پیغامی مبنی بر اینکه «میزان درخواستی بیش از میزان موجودی است» موضوع را اطلاع دهد. تابع **getBalance** باید میزان موجودی جاری را برگشت دهد. برنامه‌ای ایجاد کنید که دو شی **Account** ایجاد کرده و توابع عضو کلاس **Account** را تست کنید.

**۱۳-۳** (کلاس **Invoice**) کلاسی بنام **Invocie** (فاکتور) ایجاد کنید که در یک فروشگاه سخت‌افزار با هدف نمایش فاکتور از ایتم‌های فروخته شده بکار گرفته شود. این فاکتور باید شامل چهار قسمت اطلاعاتی بعنوان اعضای داده باشد، شماره قطعه (از نوع رشته)، توضیح قطعه (از نوع رشته)، تعداد قطعه فروخته شده (از نوع **int**) و قیمت هر قطعه (از نوع **int**). این کلاس باید دارای سازنده‌ای باشد که مبادرت به مقداردهی اولیه چهار عضو داده کند. یک تابع **set** و **get** برای هر عضو داده در نظر بگیرید. علاوه بر این، یک تابع عضو بنام **getInvoiceAmount** تدارک ببینید که مبادرت به محاسبه فاکتور (با ضرب تعداد در قیمت هر قطعه) کرده و سپس قیمت فاکتور را بصورت یک مقدار **int** برگشت دهد. اگر تعداد، مقدار مثبتی نباشد، باید آنرا با صفر تنظیم کند. اگر قیمت قطعه‌ای مثبت نباشد، آن را با صفر تنظیم کند. یک برنامه تست‌کننده برای این کلاس بنویسید.

**۱٤-۳** (کلاس **Employee**) کلاسی بنام **Employee** ایجاد کنید که شامل سه قسمت اطلاعاتی بعنوان اعضای داده باشد، نام (از نوع رشته)، نام خانوادگی (از نوع رشته) و حقوق ماهانه (از نوع **int**). این کلاس باید دارای سازنده‌ای باشد که مبادرت به مقداردهی اولیه سه عضو داده کند. یک برنامه تست برای بررسی قابلیت کلاس **Employee** بنویسید. دو شی **Employee** ایجاد کرده و نمایش حقوق سالیانه هر شی را به نمایش در آورید. سپس افزایش حقوق ده درصدی را برای هر کارمند در نظر گرفته و مجدداً حقوق سالیانه را برای هر کارمند محاسبه و به نمایش در آورید.

**۱۵-۳** (کلاس **Date**) کلاسی به نامِ **Date** ایجاد کنید که شامل سه قسمت اطلاعاتی بعنوان اعضای داده باشد، ماه (از نوع **int**)، روز (از نوع **int**) و سال (از نوع **int**). این کلاس باید دارای یک سازنده با سه پارامتر باشد که از پارامترها برای مقداردهی اولیه این سه عضو داده استفاده می‌کند. در این تمرین فرض کنید که مقادیر تدارک دیده شده برای سال و روز صحیح باشند، اما مطمئن گردید که مقدار ماه حتماً در محدودهٔ 1 الی 12 قرار داشته باشد. اگر چنین نباشد، ماه را با 1 مقداردهی یا تنظیم کنید. برای هر عضو داده یک تابع **set** و **get** در نظر بگیرید. یک تابع **displayDate** بنویسید که ماه، روز و سال را که با یک اسلش (/) از هم جدا شده‌اند به نمایش درآورد. یک برنامه تست برای نمایش قابلیت‌های کلاس **Date** ایجاد کنید.

www.yourshop.4kia.ir