فصل نهم

کلاس‌ها: نگاهی عمیق‌تر: بخش I

اهداف

- **نحوه استفاده از یک پوشاننده پیش‌پردازنده برای اجتناب از خطاهای آشکار که با کپی کردن بیش از یکبار فایل سرآیند در فایل کد منبع بوجود می‌آیند.**
- **آشنایی با مفهوم قلمرو کلاس و دسترسی به اعضاء کلاس از طریق نام یک شی، مراجعه به یک شی یا اشاره‌گر به یک شی.**
- **تعریف سازنده‌ها با آرگومان‌های پیش‌فرض.**
- **نحوه استفاده از سازنده‌ها برای انجام عملیات «خاتمه کار» بر روی یک شی قبل از نابود شدن و از بین رفتن آن.**
- **زمان فراخوانی سازنده‌ها و نابود کننده‌ها و ترتیب فراخوانی آنها.**
- **خطاهای منطقی که به هنگام برگشت دادن یک مراجعه به داده private توسط یک تابع عضو public رخ می‌دهند.**
- **انتصاب عضوهای داده یک شی به عضوهای یک شی دیگر با تخصیص Memberwise.**

رئوس مطالب

۹-۱ مقدمه
۹-۲ مبحث آموزشی: کلاس Time
۹-۳ قلمرو کلاس و دسترسی به اعضاء کلاس
۹-٤ جداسازی واسط از پیاده‌سازی
۹-٥ توابع دسترسی و توابع یوتیلیتی
۹-٦ مبحث آموزشی کلاس Time: سازنده‌ها همراه با آرگومان‌های پیش‌فرض
۹-۷ نابودکننده‌ها
۹-۸ زمان فراخوانی سازنده‌ها و نابودکننده‌ها
۹-۹ مبحث آموزشی کلاس Time: برگشت دادن یک مراجعه به داده عضو private
۹-۱۰ تخصیص Memberwise
۹-۱۱ استفاده مجدد از نرم‌افزار
۹-۱۲ مبحث آموزشی مهندسی نرم‌افزار: شروع برنامه‌نویسی کلاس‌های سیستم ATM

۹-۱ مقدمه

در فصل‌های قبلی، به معرفی برخی از مفاهیم پایه در برنامه‌نویسی شی‌گرا در C++ پرداختیم. همچنین در ارتباط با روش و اسلوب توسعه و ایجاد برنامه‌هایمان صحبت کردیم: صفات و رفتار مقتضی برای هر کلاس را انتخاب می‌کنیم و به یک روش معین مشخص می‌سازیم که کدام شی‌ها از کلاس‌هایمان با شی‌های موجود در کتابخانه کلاس‌های استاندارد C++ برای برآورده کردن هر هدف برنامه می‌توانند همکاری کنند.

در این فصل، نگاهی عمیق‌تر به کلاس‌ها خواهیم داشت. از کلاس یکپارچه **Time** بعنوان یک مبحث آموزشی در این فصل (سه مثال) و فصل دهم (دو مثال) استفاده کرده‌ایم تا به بیان روش‌های ایجاد کلاس بپردازیم. کار را با یک کلاس **Time** شروع می‌کنیم که نگاهی مجدد بر چندین ویژگی عرضه شده در فصل‌های قبلی داشته باشیم. همچنین این مثال به توصیف یک مفهوم اساسی در مهندسی نرم‌افزار C++ یعنی «*پوشاننده پیش پردازنده*» در ارتباط با فایلهای سرآیند می‌پردازد تا از قرار گرفتن بیش از یکبار کد سرآیند در همان فایل کد منبع جلوگیری شود. زمانیکه یک کلاس بتواند فقط یکبار تعریف شود، استفاده از چنین دستوردهنده‌های پیش‌پردازنده از وقوع خطاهای آشکار متعدد جلوگیری می‌کند.

سپس در ارتباط با قلمرو کلاس و رابطه موجود مابین اعضای کلاس صحبت خواهیم کرد. همچنین به توضیح اینکه چگونه کد سرویس‌گیرنده می‌تواند به اعضای **public** کلاس از طریق سه نوع «دستگیره» (نام شی، مراجعه به شی یا اشاره‌گر به شی) دسترسی پیدا کند، خواهیم پرداخت. همانطوری که خواهید

کلاس‌ها : نگاهی عمیق‌تر : بخش I ـــــــ

```
3  // Member functions are defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time class definition
10 class Time
11 {
12 public:
13    Time(); // constructor
14    void setTime( int, int, int ); // set hour, minute and second
15    void printUniversal(); // print time in universal-time format
16    void printStandard(); // print time in standard-time format
17 private:
18    int hour; // 0 - 23 (24-hour clock format)
19    int minute; // 0 - 59
20    int second; // 0 - 59
21 }; // end class Time
22
23 #endif
```

شکل ۱-۹ | تعریف کلاس Time.

تعریف کلاس Time

تعریف کلاس (شکل ۱-۹) حاوی نمونه اولیه (خطوط 16-13) برای توابع عضو **Time**، **setTime**، **printUniversal** و **printStandard** است. همچنین کلاس شامل اعضای صحیح خصوصی (private) بنام‌های **hour**، **minute** و **second** در خطوط 20-18 می‌باشد. اعضای داده خصوصی **Time** می‌توانند فقط از طریق چهار تابع عضو خود در دسترس قرار گیرند. در فصل ۱۲ به معرفی سومین تصریح کننده دسترسی بنام **protected** خواهیم پرداخت، زمانیکه به بررسی توارث و نقش آن در برنامه‌نویسی شی‌گرا پرداختیم.

برنامه‌نویسی ایده‌ال

*برای افزایش خوانایی و وضوح برنامه، از هر تصریح کننده دسترسی فقط یکبار در همه تعریف کلاس استفاده کنید. ابتدا اعضای **public** را قرار دهید، که پیدا کردن آنها آسان باشد.*

مهندسی نرم‌افزار

هر عنصر از کلاس باید در دید private قرار داشته باشد، مگر اینکه مسلم گردد آن عنصر نیاز به میدانی با دید public دارد.

در برنامه شکل ۱-۹ توجه کنید که تعریف کلاس در پوشاننده پیش‌پردازنده احاطه شده است (خطوط 23 ,57):

```
// prevent multiple inclusions of header file
# ifndef TIME_H
# define TIME_H
    …
# endif
```

زمانیکه برنامه‌های بزرگتر ایجاد می‌کنیم، تعاریف و اعلان‌های دیگر هم در فایلهای سرآیند جای داده خواهند شد. پوشاننده پیش‌پردازنده فوق سبب می‌شود تا از تکرار کد مابین **#ifndef** (به معنی "if not

"defined) و **#endif** اجتناب شود اگر شامل نام TIME_H بوده و تعریف شده باشد. اگر سرآیند قبلاً در فایلی بکار گرفته نشده باشد، نام TIME_H توسط دستور دهنده **#define** تعریف شده و فایل سرآیند بکار گرفته خواهد شد. اگر سرآیند قبلاً استفاده شده باشد TIME_H تعریف شده و فایل سرآیند مجدداً در برنامه وارد نخواهد شد.

اجتناب از خطا

*از دستوردهنده‌های پیش‌پردازنده **#ifndef** **#define** و **#endif** بعنوان پوشاننده پیش‌پردازنده استفاده کنید تا از وارد کردن بیش از یکبار فایلهای سرآیند به برنامه جلوگیری شود.*

برنامه‌نویسی ایده‌ال

*در نام فایل سرآیند از حروف بزرگ به همراه نقطه بجای خط زیرین در دستوردهنده‌های پیش‌پردازنده **#ifndef** و **#define** از یک فایل سرآیند استفاده کنید.*

توابع عضو کلاس Time

در برنامه شکل ۲-۹، سازنده **Time** در خطوط 17-14 مبادرت به مقداردهی اولیه اعضای داده با صفر کرده است (یعنی زمان جهانی معادل با 12 AM). با اینکار مطمئن خواهیم شد که شی کار خود را از یک وضعیت یا حالت پایدار آغاز خواهد کرد. مقادیر اشتباه یا نامعتبر نمی‌توانند در اعضای داده یک شی **Time** ذخیره شوند، چرا که سازنده به هنگام ایجاد شی **Time** فراخوانی شده و تمام فعالیت‌های که توسط یک سرویس‌گیرنده به منظور تغییر دادن اعضای داده صورت می‌گیرد، توسط تابع **setTime** بدقت بررسی می‌شود. توجه به این نکته مهم است که برنامه‌نویس قادر به تعریف چندین سازنده سربارگذاری شده برای یک کلاس باشد.

```
1  // Fig. 9.2: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  using std::cout;
5
6  #include <iomanip>
7  using std::setfill;
8  using std::setw;
9
10 #include "Time.h" // include definition of class Time from Time.h
11
12 // Time constructor initializes each data member to zero.
13 // Ensures all Time objects start in a consistent state.
14 Time::Time()
15 {
16    hour = minute = second = 0;
17 } // end Time constructor
18
19 // set new Time value using universal time; ensure that
20 // the data remains consistent by setting invalid values to zero
21 void Time::setTime( int h, int m, int s )
22 {
23    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
24    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
25    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
26 } // end function setTime
27
```

```
28 // print Time in universal-time format (HH:MM:SS)
29 void Time::printUniversal()
30 {
31    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
32       << setw( 2 ) << minute << ":" << setw( 2 ) << second;
33 } // end function printUniversal
34
35 // print Time in standard-time format (HH:MM:SS AM or PM)
36 void Time::printStandard()
37 {
38    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
39       << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
40       << second << ( hour < 12 ? " AM" : " PM" );
41 } // end function printStandard
```

شکل ۲-۹ | تعریف تابع عضو کلاس Time.

نمی‌توان اعضای داده یک کلاس را در مکانی که در بدنه کلاس اعلان شده‌اند، مقداردهی اولیه کرد. بشدت توصیه می‌شود که این اعضای داده توسط سازنده کلاس مقداردهی اولیه شوند. همچنین می‌توان توسط تابع *set* کلاس **Time** مبادرت به تخصیص مقادیر به داده‌های عضو کرد. [نکته: در فصل دهم نشان خواهیم داد که فقط اعضای داده **static const** یک کلاس از نوع‌های صحیح یا **enum** را می‌توان در بدنه کلاس مقداردهی اولیه کرد.]

خطای برنامه‌نویسی

اقدام به مقداردهی صریح اولیه یک عضو داده غیراستاتیک از یک کلاس در تعریف کلاس، خطای نحوی است.

تابع **setTime** در خطوط 26-21 یک تابع **public** است که سه پارامتر **int** اعلان کرده و از آنها برای تنظیم زمان استفاده می‌کند. یک عبارت شرطی مبادرت به تست هر آرگومان می‌کند تا تعیین کنید که آیا مقدار موجود در محدودهٔ خاص قرار دارد یا خیر. برای مثال، مقدار **hour** در خط 23 بایستی بزرگتر یا برابر صفر و کوچکتر از 24 باشد، چرا که در فرمت جهانی زمان، ساعت یک مقدار صحیح از صفر تا 23 است (برای مثال، 1PM نشاندهنده ساعت 13 و 11PM نشاندهنده 23 است، نیمه شب برابر ساعت 0 و نیمروز برابر ساعت 12 است). به همین ترتیب، مقادیر **minute** و **second** (خطوط 24 و 25) بایستی بزرگتر یا برابر صفر و کمتر از 60 باشند. هر مقداری خارج از این محدودها با صفر تنظیم می‌شود تا مطمئن شویم که شی **Time** همیشه حاوی داده سازگار است، در اینحالت حتی اگر آرگومان‌های ارسالی به تابع **setTime** صحیح نباشند، داده‌های شی همیشه در محدود صحیح نگهداری خواهند شد. در این مثل، صفر یک مقدار سازگار برای **hour**، **minute** و **secand** است. مقدار ارسالی به**setTime** یک مقدار صحیح خواهد بود اگر در محدودهٔ تعیین شده قرار داشته باشد. بنابر این هر عددی در محدوده 23-0 یک مقدار صحیح برای **hour** تلقی خواهد شد. با این همه، یک مقدار سازگار، ضرورتاً نمی‌تواند یک مقدار صحیح باشد. اگر **setTime** مبادرت به تنظیم **hour** با صفر کند به این دلیل که آرگومان دریافتی خارج از

محدودهٔ است، پس فقط **hour** (ساعت) در صورتی صحیح خواهد بود که زمان جاری همزمان با نیمه شب باشد.

تابع **printUniversal** (خطوط 33-29 از شکل ۲-۹) هیچ آرگومانی دریافت نمی‌کند و تاریخ را برحسب فرمت جهانی زمان، متشکل از سه جفت کولن متمایز کننده ارقام برای ساعت، دقیقه و ثانیه چاپ می‌کند. برای مثال اگر زمان 1:30:07PM باشد، تابع **printUniversal** مقدار 13:30:07 برگشت می‌دهد. دقت کنید که در خط 31 از تابع **setfill** برای مشخص کردن کاراکتر پرکننده استفاده شده است که مبادرت به نمایش یک مقدار صحیح در خروجی در یک فیلد بجای *ارزش عددی* از ارقام می‌کند. بطور پیش‌فرض، کاراکترهای پرکننده در سمت چپ ارقام یک عدد ظاهر می‌شوند. در این مثال، اگر مقدار **minute** (دقیقه) برابر 2 باشد، بصورت 02 به نمایش در خواهد آمد، چرا که کاراکتر پرکننده با صفر (**'0'**) تنظیم شده است. اگر عددی که قرار است در خروجی قرار گیرد کل فیلد تعیین شده را در برگیرد، کاراکتر پرکننده به نمایش در نخواهد آمد. توجه کنید زمانیکه کاراکتر پرکننده با **setfill** همراه می‌شود، این کاراکتر بر روی مابقی مقادیری که در پهنای آن فیلد به نمایش در خواهند آمد، هم اعمال خواهد گردید. نقطه مقابل آن **setw** است که فقط بر روی مقدار بعدی به نمایش در آمده اعمال می‌شود.

تابع **printStandard** (خطوط 41-36) هیچ آرگومانی دریافت نمی‌کند و تاریخ را در فرمت استاندارد زمانی به نمایش در می‌آورد. این فرمت متشکل از مقادیر ساعت، دقیقه و ثانیه است که توسط کولن‌های که بدنبال آن AM یا PM قرار دارد از هم جدا شده‌اند (مانند 1:27:06 PM). همانند تابع **printUniversal** تابع **printStandard** از **setfill('0')** برای قالب‌بندی دقیقه و ثانیه بعنوان دو مقدار رقمی با دنباله صفر در صورت نیاز استفاده کرده است. در خط 38 از عملگر شرطی (**:?**) برای تعیین نمایش مقدار ساعت استفاده شده است. اگر ساعت برابر 0 یا 12 (AM یا PM) باشد بصورت 12 و در غیر اینصورت ساعت بصورت مقداری از 1 تا 11 به نمایش در خواهد آمد. عملگر شرطی بکار رفته در خط 40 تعیین می‌کند که آیا AM یا PM به نمایش درآید یا خیر.

تعریف توابع عضو خارج از تعریف کلاس: قلمرو کلاس

حتی در صورتیکه یک تابع عضو در تعریف کلاسی اعلان شده باشد می‌تواند در خارج از تعریف کلاس، تعریف گردد (و به کلاس از طریق عملگر باینری تفکیک قلمرو مرتبط شود)، که هنوز هم تابع عضو در درون قلمرو کلاس قرار خواهد داشت، به این معنی که نام تابع فقط توسط اعضای دیگر کلاس شناخته شده خواهد بود مگر اینکه از طریق شیی از کلاس مورد مراجعه قرار گیرد یا اشاره‌گری به یک شی از کلاس یا عملگر باینری تفکیک قلمرو بکار گرفته شود.

اگر تابع عضوی در بدنه یک کلاس تعریف شده باشد، کامپایلر C++ مبادرت به فراخوانی inline آن تابع خواهد کرد. توابع عضو تعریف شده در خارج از تعریف کلاس می‌توانند بصورت صریح و خطی و توسط کلمه کلیدی ***inline*** فراخوانی شوند.

توابع عضو در مقابل توابع سراسری

نکته جالب توجه در این است که توابع عضو **printUniversal** و **printStandard** هیچ آرگومانی دریافت نمی‌کنند. به این دلیل که این توابع عضو بصورت غیرصریح می‌دانند که باید اعضای داده شی **Time** را به هنگام فعال شدن چاپ کنند. چنین عملی فراخوانی توابع عضو را به نسبت توابع عادی در برنامه نویسی روالی بسیار مختصر می‌کند.

استفاده از کلاس Time

پس از اینکه کلاس **Time** تعریف شد، می‌توان از آن بعنوان یک نوع در اعلان شی، آرایه و اشاره‌گر بصورت زیر استفاده کرد:

```
Time sunset; // object of type Time
Time arrayOfTimes [5], // array of 5 Time objects
Time &dinnerTime = swnset; // reference to a Time object
Time *timePtr = &dinnerTime, // pointer to a Time object
```

در برنامه شکل ۳-۹ از کلاس **Time** استفاده شده است. خط 12 مبادرت به نمونه‌سازی یک شی منفرد از کلاس **Time** بنام **t** می‌کند. زمانیکه یک شی معرفی می‌شود، سازنده **Time** برای مقداردهی اولیه هر داده عضو **private** با صفر فراخوانی می‌شود سپس، خطوط 16 و 18 زمان را با فرمت‌های استاندارد و جهانی برای تایید اینکه اعضا بدرستی مقداردهی اولیه شده‌اند چاپ می‌کنند. خط 20 مبادرت به تنظیم زمان جدید با فراخوانی تابع عضو **setTime** کرده و خطوط 24 و 26 مجدداً زمان را در هر دو فرمت چاپ می‌کنند. خط 28 مبادرت به استفاده از تابع **setTime** برای تنظیم اعضای داده با مقادیر اشتباه می‌کند. در اینحالت تابع **setTime** این اشتباه را تشخیص داده و مقادیر اشتباه را با صفر تنظیم می‌نماید تا شی در یک وضعیت سازگار (پایدار) باقی بماند. سرانجام، خطوط 33 و 35 زمان را در هر دو فرمت چاپ می‌کنند.

نگاهی جلوتر ترکیب و توارث

غالباً، کلاس‌ها مجبور نیستند از «ابتدا» ایجاد شوند. بجای آن می‌توانند حاوی شی‌های از کلاس‌های دیگر بعنوان اعضا باشند یا می‌تواند از کلاس‌های دیگر مشتق شوند تا صفات و رفتارهای برای استفاده کلاس‌های جدید فراهم آورند. چنین قابلیتی که بعنوان استفاده مجدد از نرم‌افزار شناخته می‌شود می‌تواند در نگهداری کد و کارایی برنامه‌نویسی نقش مهمی بازی کند. وارد کردن شی‌های کلاس بعنوان اعضای کلاس‌های دیگر، *ترکیب* (یا *تجمع*) نامیده می‌شود و در فصل دهم توضیح داده شده است. مشتق کردن کلاس‌های جدید از کلاس‌های موجود، *توارث* نامیده می‌شود و در فصل دوازدهم به توضیح آن پرداخته شده است.

```
// Fig. 9.3: fig09_03.cpp
// Program to test class Time.
// NOTE: This file must be compiled with Time.cpp.
#include <iostream>
using std::cout;
using std::endl;

#include "Time.h" // include definition of class Time from Time.h

int main()
{
   Time t; // instantiate object t of class Time

   // output Time object t's initial values
   cout << "The initial universal time is ";
   t.printUniversal(); // 00:00:00
   cout << "\nThe initial standard time is ";
   t.printStandard(); // 12:00:00 AM

   t.setTime( 13, 27, 6 ); // change time

   // output Time object t's new values
   cout << "\n\nUniversal time after setTime is ";
   t.printUniversal(); // 13:27:06
   cout << "\nStandard time after setTime is ";
   t.printStandard(); // 1:27:06 PM

   t.setTime( 99, 99, 99 ); // attempt invalid settings

   // output t's values after specifying invalid values
   cout << "\n\nAfter attempting invalid settings:"
      << "\nUniversal time: ";
   t.printUniversal(); // 00:00:00
   cout << "\nStandard time: ";
   t.printStandard(); // 12:00:00 AM
   cout << endl;
   return 0;
} // end main
```

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

شکل ۳-۹ | برنامه تست کلاس Time.

سایزشی

افراد کم تجربه در برنامه‌نویسی شی‌گرا تصور می‌کنند که بایستی شی‌ها بکلی بزرگ باشند چرا که آنها حاوی اعضای داده و توابع عضو می‌باشند. به لحاظ منطقی این تصور صحیح است، و برنامه‌نویس می‌تواند چنین ذهنیتی داشته باشد. با این همه، به لحاظ فیزیکی این امر صادق نیست.

۳-۹ قلمرو کلاس و دسترسی به اعضاء کلاس

اعضای داده کلاس (متغیرهای اعلان شده در تعریف کلاس) و توابع عضو (توابع اعلان شده در تعریف کلاس) به قلمرو کلاس تعلق دارند. توابع غیرعضو در *قلمرو فایل* جای دارند.

۲٦٤فصل نهم ـــــــ

```
1  // Fig. 9.4: fig09_04.cpp
2  // Demonstrating the class member access operators . and ->
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // class Count definition
8  class Count
9  {
10 public: // public data is dangerous
11    // sets the value of private data member x
12    void setX( int value )
13    {
14       x = value;
15    } // end function setX
16
17    // prints the value of private data member x
18    void print()
19    {
20       cout << x << endl;
21    } // end function print
22
23 private:
24    int x;
25 }; // end class Count
26
27 int main()
28 {
29    Count counter; // create counter object
30    Count *counterPtr = &counter; // create pointer to counter
31    Count &counterRef = counter; // create reference to counter
32
33    cout << "Set x to 1 and print using the object's name: ";
34    counter.setX( 1 ); // set data member x to 1
35    counter.print(); // call member function print
36
37    cout << "Set x to 2 and print using a reference to an object: ";
38    counterRef.setX( 2 ); // set data member x to 2
39    counterRef.print(); // call member function print
40
41    cout << "Set x to 3 and print using a pointer to an object: ";
42    counterPtr->setX( 3 ); // set data member x to 3
43    counterPtr->print(); // call member function print
44    return 0;
45 } // end main
```

```
Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

شکل ٤-٩ | دسترسی به توابع عضو یک شی از طریق نوع شی.

٤-٩ جداسازی واسط از پیاده‌سازی

در فصل سوم، شروع به وارد کردن تعریف کلاس و تعریف تابع عضو در یک فایل کردیم. سپس به توضیح نحوه جداسازی این کد به دو فایل پرداختیم، یک فایل سرآیند برای تعریف کلاس (یعنی واسط کلاس) و فایل کد منبع برای تعریف تابع عضو کلاس (یعنی پیاده‌سازی کلاس). بخاطر دارید که با انجام اینکار انجام تغییرات در برنامه‌ها آسانتر می‌شود، تا آنجا که به سرویس‌گیرندگان کلاس مربوط می‌شود، تغییر در پیاده‌سازی کلاس تاثیری در سرویس‌گیرنده ندارد تا مادامیکه واسط تدارک دیده شده توسط کلاس برای سرویس‌گیرنده بدون تغییر باقی مانده باشد.

۲٦٦فصل نهم ـــــــ

```
17 }; // end class SalesPerson
18
19 #endif
```

شکل ۵-۹ | تعریف کلاس SalesPerson.

در برنامه شکل ۶-۹ سازنده **SalesPerson** (خطوط 15-19) مبادرت به مقداردهی اولیه آرایه **sales** با صفر کرده است. تابع عضو سراسری **setSales** (خطوط 36-43) مبادرت به تنظیم فروش برای یک ماه در آرایه **sales** می‌کند. تابع عضو سراسری **public** بنام **printAnnualSales** (خطوط 46-51) مجموع فروش دوازده ماهه را چاپ می‌کند. تابع یوتیلیتی خصوصی (private) بنام **totalAnnualSales** (خطوط 54-62) مجموع فروش دوازده ماهه را با استفاده از **printAnnualSales** بدست می‌آورد. در برنامه شکل ۷-۹، توجه کنید که کاربرد تابع **main** فقط در فراخوانی پشت سرهم توابع عضو بوده و دارای عبارات کنترلی نمی‌باشد. منطق کار با آرایه **sales** در این است که این آرایه بطور کامل در توابع عضو کلاس **SalesPerson** کپسوله شود.

```
1  // Fig. 9.6: SalesPerson.cpp
2  // Member functions for class SalesPerson.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10 using std::setprecision;
11
12 #include "SalesPerson.h" // include SalesPerson class definition
13
14 // initialize elements of array sales to 0.0
15 SalesPerson::SalesPerson()
16 {
17    for ( int i = 0; i < 12; i++ )
18       sales[ i ] = 0.0;
19 } // end SalesPerson constructor
20
21 // get 12 sales figures from the user at the keyboard
22 void SalesPerson::getSalesFromUser()
23 {
24    double salesFigure;
25
26    for ( int i = 1; i <= 12; i++ )
27    {
28       cout << "Enter sales amount for month " << i << ": ";
29       cin >> salesFigure;
30       setSales( i, salesFigure );
31    } // end for
32 } // end function getSalesFromUser
33
34 // set one of the 12 monthly sales figures; function subtracts
35 // one from month value for proper subscript in sales array
36 void SalesPerson::setSales( int month, double amount )
37 {
38    // test for valid month and amount values
39    if ( month >= 1 && month <= 12 && amount > 0 )
40       sales[ month - 1 ] = amount; // adjust for subscripts 0-11
41    else // invalid month or amount value
42       cout << "Invalid month or sales figure" << endl;
43 } // end function setSales
44
45 // print total annual sales (with the help of utility function)
```

۲۶۸فصل نهم ـــــــ

Time تعریف کرده‌اند که مقادیری برای پارامترهای **hr**، **min** و **sec** دریافت می‌کند که در مقداردهی اولیه اعضاء داده ساعت، دقیقه و ثانیه کاربرد دارند.

```
1  // Fig. 9.8: Time.h
2  // Declaration of class Time.
3  // Member functions defined in Time.cpp.
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time abstract data type definition
10 class Time
11 {
12 public:
13    Time( int = 0, int = 0, int = 0 ); // default constructor
14
15    // set functions
16    void setTime( int, int, int ); // set hour, minute, second
17    void setHour( int ); // set hour (after validation)
18    void setMinute( int ); // set minute (after validation)
19    void setSecond( int ); // set second (after validation)
20
21    // get functions
22    int getHour(); // return hour
23    int getMinute(); // return minute
24    int getSecond(); // return second
25
26    void printUniversal(); // output time in universal-time format
27    void printStandard(); // output time in standard-time format
28 private:
29    int hour; // 0 - 23 (24-hour clock format)
30    int minute; // 0 - 59
31    int second; // 0 - 59
32 }; // end class Time
33
34 #endif
```

شکل ۸-۹ | کلاس Time حاوی یک سازنده با آرگومان‌های پیش‌فرض.

توجه کنید که کلاس **Time** مبادرت به تدارک دیدن توابع *set* و *get* برای هر عضو داده کرده است. اکنون سازنده **Time** اقدام به فراخوانی **setTime** می‌کند و آن هم توابع **setHour**، **setMinute** و **setSecond** را برای اعتبار سنجی و تخصیص مقادیر به اعضای داده فراخوانی می‌کند. آرگومان‌های پیش‌فرض، سازنده را مطمئن می‌سازند که حتی اگر مقادیر در فراخوانی سازنده در نظر گرفته نشده باشند، سازنده قادر به مقداردهی اولیه اعضای داده باشد تا بتواند شی **Time** را در یک وضعیت پایدار نگهداری کند. سازنده‌ای که تمام آرگومان‌های آن پیش‌فرض هستند، یک سازنده پیش‌فرض محسوب می‌شود، یعنی سازنده‌ای که می‌تواند بدون آرگومان فراخوانی یا فعال گردد. حداکثر یک سازنده پیش‌فرض در هر کلاس می‌تواند وجود داشته باشد.

```
1  // Fig. 9.9: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  using std::cout;
5
6  #include <iomanip>
7  using std::setfill;
8  using std::setw;
9
```

```
#include "Time.h" // include definition of class Time from Time.h

// Time constructor initializes each data member to zero;
// ensures that Time objects start in a consistent state
Time::Time( int hr, int min, int sec )
{
   setTime( hr, min, sec ); // validate and set time
} // end Time constructor

// set new Time value using universal time; ensure that
// the data remains consistent by setting invalid values to zero
void Time::setTime( int h, int m, int s )
{
   setHour( h ); // set private field hour
   setMinute( m ); // set private field minute
   setSecond( s ); // set private field second
} // end function setTime

// set hour value
void Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
} // end function setHour

// set minute value
void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
} // end function setMinute

// set second value
void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setSecond

// return hour value
int Time::getHour()
{
   return hour;
} // end function getHour

// return minute value
int Time::getMinute()
{
   return minute;
} // end function getMinute

// return second value
int Time::getSecond()
{
   return second;
} // end function getSecond

// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << getHour() << ":"
      << setw( 2 ) << getMinute() << ":" << setw( 2 ) << getSecond();
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
   cout << (( getHour() == 0 || getHour() == 12) ? 12: getHour() % 12 )
      << ":" << setfill( '0' ) << setw( 2 ) << getMinute()
      << ":" << setw(2)<< getSecond() << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

شکل ۹-۹ | تعریف تابع عضو کلاس Time شامل یک سازنده که آرگومان دریافت می‌کند.

کلاس‌ها : نگاهی عمیق‌تر : بخش I ـــــــ

```
using std::cout;
using std::endl;

#include "Time.h" // include definition of class Time from Time.h

int main()
{
   Time t1; // all arguments defaulted
   Time t2( 2 ); // hour specified; minute and second defaulted
   Time t3( 21, 34 ); // hour and minute specified; second defaulted
   Time t4( 12, 25, 42 ); // hour, minute and second specified
   Time t5( 27, 74, 99 ); // all bad values specified

   cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
   t1.printUniversal(); // 00:00:00
   cout << "\n  ";
   t1.printStandard(); // 12:00:00 AM

   cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
   t2.printUniversal(); // 02:00:00
   cout << "\n  ";
   t2.printStandard(); // 2:00:00 AM

   cout << "\n\nt3: hour and minute specified; second defaulted\n  ";
   t3.printUniversal(); // 21:34:00
   cout << "\n  ";
   t3.printStandard(); // 9:34:00 PM

   cout << "\n\nt4: hour, minute and second specified\n  ";
   t4.printUniversal(); // 12:25:42
   cout << "\n  ";
   t4.printStandard(); // 12:25:42 PM

   cout << "\n\nt5: all invalid values specified\n  ";
   t5.printUniversal(); // 00:00:00
   cout << "\n  ";
   t5.printStandard(); // 12:00:00 AM
   cout << endl;
   return 0;
} // end main
```

```
Constructed with:

t1: all arguments defaulted
   00:00:00
   12:00:00 AM

t2: hour specified; minute and second defaulted
   02:00:00
   2:00:00 AM

t3: hour and minute specified; second defaulted
  12:34:00
  9:34:00 PM

t4: hour, minute and second defaulted
  12:25:42
  12:25:42 AM

t5: all invalid values specified
  00:00:00
  12:00:00 AM
```

شکل ۱۰-۹ | سازنده با آرگومان‌های پیش‌فرض.

۹-۷ نابود کننده‌ها

نابودکننده (*تخریب‌کننده*) نوع دیگری از تابع عضو می‌باشد. نام نابودکننده یک کلاس همراه با کاراکتر مد (~) و نام کلاس مشخص می‌شود. این قاعده نامگذاری دارای جاذبه شهودی است، زیرا همانطوری که

کلاس‌ها : نگاهی عمیق‌تر : بخش I ـــــــ

برای پردازش باز نشود استفاده می‌شود. تابع **abort** کار مشابهی با تابع **exit** انجام می‌دهد، اما برنامه را بلافاصله مجبور به خاتمه می‌کند بدون اینکه به نابودکننده‌ای هر شی اجازه دهد تا فراخوانی گردند. معمولاً از تابع **abort** برای تشخیص خاتمه غیرعادی برنامه استفاده می‌شود.

سازنده برای یک شی محلی اتوماتیک زمانی فراخوانی می‌شود که اجرا برنامه به مکانی برسد که شی در آنجا تعریف شده است، نابودکننده متناظر هم زمانی فراخوانی می‌گردد که اجرا، قلمرو شی را ترک می‌کند. سازنده‌ها و نابودکننده‌های متعلق به شی‌های اتوماتیک در هر بار ورود و خروج اجرای برنامه به قلمرو شی فراخوانی می‌شوند. نابودکننده‌ها برای شی‌های اتوماتیک فراخوانی نمی‌شوند اگر برنامه با فراخوانی تابع **exit** یا **abort** خاتمه یابد.

سازنده برای یک شی محلی **static** (استاتیک) فقط یک بار فراخوانی می‌شود و آن هم زمانی است که اجرا به مکانی برسد که شی در آنجا تعریف شده است. نابودکننده متناظر هم زمانی فراخوانی می‌شود که **main** خاتمه یافته باشد یا برنامه، تابع **exit** را فراخوانی کرده باشد. شی‌های سراسری و استاتیک به ترتیب معکوس از ایجاد خود نابود می‌شوند. نابودکننده‌ها در صورتیکه برنامه با فراخوانی تابع **abort** خاتمه یافته باشد، برای شی‌های استاتیک فراخوانی نخواهند شد. برنامه بکار رفته در شکل‌های۱۳-۹ الی۱۱-۹ نشاندهنده ترتیبی است که سازنده‌ها و نابودکننده‌های برای شی‌های کلاس **CreateAndDestory** فراخوانی می‌شوند (شکل ۱۱-۹ و شکل ۱۲-۹) از کلاس‌های ذخیره‌سازی مختلف در قلمروهای گوناگون. هر شی از کلاس **CreateAndDestory** حاوی (خطوط 17-16) یک نوع صحیح (objectID) و یک رشته (message) است که در خروجی برنامه بکار گرفته شده‌اند تا هویت شی باشند. این مثال صرفاً جنبه آموزشی دارد. از اینرو، خط 23 از نابودکننده در شکل ۱۲-۹ تعیین می‌کند که آیا شی نابود شده دارای شناسه (objectID) با مقدار 1 یا 6 است یا خیر، و اگر چنین باشد یک کاراکتر خط جدید در خروجی قرار داده می‌شود. این خط کمک می‌کند تا درک خروجی برنامه آسانتر شود.

```
1  // Fig. 9.11: CreateAndDestroy.h
2  // Definition of class CreateAndDestroy.
3  // Member functions defined in CreateAndDestroy.cpp.
4  #include <string>
5  using std::string;
6
7  #ifndef CREATE_H
8  #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13    CreateAndDestroy( int, string ); // constructor
14    ~CreateAndDestroy(); // destructor
15 private:
16    int objectID; // ID number for object
17    string message; // message describing object
18 }; // end class CreateAndDestroy
19
20 #endif
```

شکل ۱۱-۹ | تعریف کلاس CreateAndDestory.

```
1  // Fig. 9.12: CreateAndDestroy.cpp
2  // Member-function definitions for class CreateAndDestroy.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "CreateAndDestroy.h" // include CreateAndDestroy class definition
8
9  // constructor
10 CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
11 {
12    objectID = ID; // set object's ID number
13    message = messageString; // set object's descriptive message
14
15    cout << "Object " << objectID << "   constructor runs   "
16       << message << endl;
17 } // end CreateAndDestroy constructor
18
19 // destructor
20 CreateAndDestroy::~CreateAndDestroy()
21 {
22    // output newline for certain objects; helps readability
23    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
24
25    cout << "Object " << objectID << "   destructor runs    "
26       << message << endl;
27 } // end ~CreateAndDestroy destructor
```

شکل ۱۲-۹ | تعریف تابع عضو کلاس CreateAndDestory.

در شکل ۱۳-۹ شی **first** در قلمرو سراسری تعریف شده است (خط 12). در واقع سازنده آن قبل از اجرای هر عبارتی در **main** فراخوانی می‌شود و نابودکننده آن در خاتمه برنامه و پس از اینکه نابودکننده سایر شی‌ها اجرا شدند، فراخوانی می‌گردد.

تابع **main** (خطوط 26-14) سه شی اعلان کرده است. شی‌های **second** (خط 17) و **fourth** (خط 23) شی‌های اتوماتیک محلی بوده و شی **third** (خط 18) یک شی محلی استاتیک است. سازنده هر یک از این شی‌ها به هنگام رسیدن اجرا به نقطه‌ای که شی در آن اعلان شده فراخوانی می‌شود. نابودکننده شی‌های **fourth** و سپس **second** زمانی فراخوانی می‌شوند که اجرا به انتهای **main** رسیده باشد. بدلیل اینکه شی **third** استاتیک است، تا زمان خاتمه برنامه باقی می‌ماند. نابودکننده شی **third** قبل از نابودکننده شی سراسری **first**، اما پس از اینکه تمام شی‌های دیگر نابود شدند فراخوانی می‌گردد.

تابع **create** (خطوط 36-29) سه شی اعلان کرده است، **fifth** (خط 32) و **seventh** (خط 34) بعنوان شی‌های اتوماتیک محلی و **sixth** (خط 33) بعنوان یک شی استاتیک محلی. نابودکننده شی‌های **seventh** و سپس **fifth** فراخوانی می‌شوند زمانیکه **create** خاتمه می‌پذیرد. بدلیل اینکه **sixth** استاتیک است تا زمان خاتمه برنامه باقی می‌ماند. نابودکننده **sixth** قبل از نابودکننده **third** و **first** فراخوانی می‌شود، اما پس از نابودی تمام شی‌های دیگر نابود می‌شود.

```
1  // Fig. 9.13: fig09_13.cpp
2  // Demonstrating the order in which constructors and
3  // destructors are called.
4  #include <iostream>
```

```
using std::cout;
using std::endl;

#include "CreateAndDestroy.h" // include CreateAndDestroy class definition

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object

int main()
{
   cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
   CreateAndDestroy second( 2, "(local automatic in main)" );
   static CreateAndDestroy third( 3, "(local static in main)" );

   create(); // call function to create objects

   cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
   CreateAndDestroy fourth( 4, "(local automatic in main)" );
   cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
   return 0;
} // end main

// function to create objects
void create( void )
{
   cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
   CreateAndDestroy fifth( 5, "(local automatic in create)" );
   static CreateAndDestroy sixth( 6, "(local static in create)" );
   CreateAndDestroy seventh( 7, "(local automatic in create)" );
   cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
} // end function create
```

```
Object 1   constructor     runs     (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2       constructor  runs    (local automatic in main)
Object 3       constructor  runs    (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5       constructor  runs    (local automatic in create)
Object 6       constructor  runs    (local static in create)
Object 7       constructor  runs    (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7       destructor   runs    (local automatic in create)
Object 5       destructor   runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4       constructor  runs    (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4       destructor   runs    (local automatic in main)
Object 2       destructor   runs    (local automatic in main)

Object 6       destructor   runs    (local static in create)
Object 3       destructor   runs    (local static in main)

Object 1       destructor   runs    (global before main)
```

شکل ۱۳-۹ | ترتیب فراخوانی سازنده‌ها و نابودکننده‌ها.

۹-۹ مبحث آموزشی کلاس Time : برگشت دادن یک مراجعه به داده عضو private

یک مراجعه به یک شی نام مستعار برای نام شی بوده و از اینرو، می‌تواند در سمت چپ یک عبارت تخصیص بکار گرفته شود. در این زمینه، مراجعه‌ها بخوبی پذیرای نقش lvalue هستند و می‌توانند مقداری را دریافت کنند. یک روش استفاده از این قابلیت (متاسفانه) داشتن یک تابع عضو **public** از کلاسی است

که یک مراجعه به یک عضو داده **private** از آن کلاس برگشت می‌دهد. دقت کنید که اگر تابعی یک مراجعه ثابت (**const**) برگشت دهد، از آن مراجعه نمی‌توان به‌عنوان یک lvalue اصلاح‌پذیر استفاده کرد. در برنامه شکل‌های ۱۶-۹ الی ۱۴-۹ از یک کلاس ساده شده **Time** (شکل ۱۴-۹ و ۱۵-۹) استفاده شده تا به بررسی برگشت دادن یک مراجعه به یک داده عضو **private** با تابع عضو **badSetHour** (اعلان شده در شکل ۱۴-۹ از خط 15 و تعریف شده در شکل ۱۵-۹ از خطوط 33-29) پرداخته شود. در واقع برگشت دادن چنین مراجعه‌ای سبب فراخوانی تابع عضو **badSetHour** بعنوان نام جانشین برای عضو داده خصوصی **hour** می‌کند. می‌توان از فراخوانی تابع به هر روشی استفاده کرد که در آن عضو داده **private** (خصوصی) می‌تواند حتی بعنوان یک lvalue در یک عبارت تخصیص بکار گرفته شود، از اینرو سرویس‌گیرنده‌های کلاس قادر خواهند بود تا داده **private** کلاس را بطور دلخواه پاک کنند (دستکاری). توجه کنید که همین مشکل می‌تواند در صورتیکه یک اشاره‌گر به داده **private** توسط تابعی برگشت داده شود، رخ دهد.

```
1  // Fig. 9.14: Time.h
2  // Declaration of class Time.
3  // Member functions defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  class Time
10 {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int ); // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 }; // end class Time
21
22 #endif
```

شکل ۱۴-۹ | برگشت دادن یک مراجعه به داده عضو private.

```
1  // Fig. 9.15: Time.cpp
2  // Member-function definitions for Time class.
3  #include "Time.h" // include definition of class Time
4
5  // constructor function to initialize private data;
6  // calls member function setTime to set variables;
7  // default values are 0 (see class definition)
8  Time::Time( int hr, int min, int sec )
9  {
10    setTime( hr, min, sec );
11 } // end Time constructor
12
13 // set values of hour, minute and second
14 void Time::setTime( int h, int m, int s )
15 {
16    hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
17    minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
18    second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
```

```
19 } // end function setTime
20
21 // return hour value
22 int Time::getHour()
23 {
24    return hour;
25 } // end function getHour
26
27 // POOR PROGRAMMING PRACTICE:
28 // Returning a reference to a private data member.
29 int &Time::badSetHour( int hh )
30 {
31    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
32    return hour; // DANGEROUS reference return
33 } // end function badSetHour
```

شکل ۱۵-۹ | برگشت دادن یک مراجعه به یک داده عضو private.

در برنامه شکل ۱۶-۹ یک شی **Time** بنام **t** (خط 12) و یک مراجعه بنام **hourRef** (خط 15) اعلان شده است که با مراجعه برگشتی توسط فراخوانی **t.badSetHour(20)** مقداردهی اولیه می‌شود. خط 17 مقدار مستعار **hourRef** را نشان می‌دهد. با این عمل نشان داده می‌شود که چگونه **hourRef** ویژگی کپسوله‌سازی کلاس را شکسته است، عبارات موجود در **main** نبایستی به داده **private** کلاس دسترسی داشته باشند. سپس، خط 18 از نام مستعار برای تنظیم مقدار **hour** با 30 استفاده کرده (یک مقدار نامعتبر) و خط 19 مقدار برگشتی توسط تابع **getHour** را برای نمایش اینکه مقدار تخصیص‌ی به **hourRef** واقعاً داده **private** در شی **t** را تغییر داده است، به نمایش در می‌آورد. سرانجام، خط 23 از فراخوانی خود تابع **badSetHour** بعنوان یک lvalue استفاده کرده و 74 (یک مقدار نامعتبر دیگر) به مراجعه برگشتی توسط تابع تخصیص می‌دهد.

```
1  // Fig. 9.16: fig09_16.cpp
2  // Demonstrating a public member function that
3  // returns a reference to a private data member.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "Time.h" // include definition of class Time
9
10 int main()
11 {
12    Time t; // create Time object
13
14    // initialize hourRef with the reference returned by badSetHour
15    int &hourRef = t.badSetHour( 20 ); // 20 is a valid hour
16
17    cout << "Valid hour before modification: " << hourRef;
18    hourRef = 30; // use hourRef to set invalid value in Time object t
19    cout << "\nInvalid hour after modification: " << t.getHour();
20
21    // Dangerous: Function call that returns
22    // a reference can be used as an lvalue!
23    t.badSetHour( 12 ) = 74; // assign another invalid value to hour
24
25    cout << "\n\n**********************************************\n"
26       << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
27       << "t.badSetHour( 12 ) as an lvalue, invalid hour: "
28       << t.getHour()
29       << "\n**********************************************" << endl;
30    return 0;
31 } // end main
```

```
Valid hour before modification: 20
Invalid hour after modification: 30

**************************************************
POOR PROGRAMMING PRACTICE!!!!!!!!
t.badSetHour( 12 ) as an lvalue, invalid hour: 74
**************************************************
```

شکل ۱۶-۹ | برگشت دادن یک مراجعه به یک عضو داده private.

خط 28 مجدداً مقدار برگشتی توسط تابع **getHour** را برای نمایش اینکه مقدار تخصیص یافته به نتیجه فراخوانی تابع در خط 23 داده **private** در شی **t** را تغییر داده است، به نمایش در می‌آورد.

۱۰-۹ تخصیص Memberwise

می‌توان از عملگر تخصیص (=) برای تخصیص یک شی به شی دیگر از همان نوع استفاده کرد. بطور پیش‌فرض، چنین تخصیصی توسط *تخصیص memberwise* صورت می‌گیرد، هر عضو داده از شی در سمت راست عملگر تخصیص بطور جداگانه به همان عضو داده در شی قرار گرفته در سمت چپ عملگر تخصیص، انتساب داده می‌شود. در شکل‌های ۱۷-۹ و ۱۸-۹ کلاس **Date** برای استفاده در این مثال تعریف شده است. در خط 20 از شکل ۱۹-۹ از تخصیص memberwie برای انتساب اعضای داده **date1** از کلاس **Date** به اعضای داده متناظر **date2** از کلاس **Date** استفاده شده است.

```
1  // Fig. 9.17: Date.h
2  // Declaration of class Date.
3  // Member functions are defined in Date.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef DATE_H
7  #define DATE_H
8
9  // class Date definition
10 class Date
11 {
12 public:
13    Date( int = 1, int = 1, int = 2000 ); // default constructor
14    void print();
15 private:
16    int month;
17    int day;
18    int year;
19 }; // end class Date
20
21 #endif
```

شکل ۱۷-۹ | فایل سرآیند کلاس Date.

```
1  // Fig. 9.18: Date.cpp
2  // Member-function definitions for class Date.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Date.h" // include definition of class Date from Date.h
8
9  // Date constructor (should do range checking)
10 Date::Date( int m, int d, int y )
11 {
12    month = m;
13    day = d;
14    year = y;
```

```
15 } // end constructor Date
16
17 // print Date in the format mm/dd/yyyy
18 void Date::print()
19 {
20    cout << month << '/' << day << '/' << year;
21 } // end function print
```

شکل ۱۸-۹ | تعریف عضو داده کلاس Date.

```
1  // Fig. 9.19: fig09_19.cpp
2  // Demonstrating that class objects can be assigned
3  // to each other using default memberwise assignment.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "Date.h" // include definition of class Date from Date.h
9
10 int main()
11 {
12    Date date1( 7, 4, 2004 );
13    Date date2; // date2 defaults to 1/1/2000
14
15    cout << "date1 = ";
16    date1.print();
17    cout << "\ndate2 = ";
18    date2.print();
19
20    date2 = date1; // default memberwise assignment
21
22    cout << "\n\nAfter default memberwise assignment, date2 = ";
23    date2.print();
24    cout << endl;
25    return 0;
26 } // end main
```

```
date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004
```

شکل ۱۹-۹ | تخصیص memberwise.

در این مورد، عضو **month** از **date1** به عضو **month** از **date2**، عضو **day** از **date1** به عضو **day** از **date2** و عضو **year** از **date1** به عضو **year** از **date2** تخصیص می‌یابد. توجه کنید که سازنده **Date** حاوی هیچ بخشی برای بررسی خطا نیست.

شی‌ها می‌توانند بعنوان آرگومان‌های تابع ارسال شده و می‌توانند از توابع برگشت داده شوند. چنین ارسال و برگشتی بطور پیش‌فرض به *روش ارسال با مقدار* صورت می‌گیرد که در آن یک کپی از شی ارسال یا برگشت داده می‌شود. در چنین حالتی، C++ یک شی جدید ایجاد و از یک سازنده کپی کننده برای کپی مقدار شی اصلی به شی جدید استفاده می‌کند. برای هر کلاسی، کامپایلر یک سازنده کپی‌کننده پیش‌فرض تدارک دیده که هر عضو از شی اصلی را به عضو متناظر در شی جدید کپی می‌کند. همانند تخصیص memberwise، سازنده‌های کپی کننده می‌توانند به هنگام استفاده با کلاسی که حاوی اشاره‌گرهای با حافظه اخذ شده دینامیکی هستند، مشکل‌ساز شوند. در فصل یازدهم به بررسی این موضوع خواهیم پرداخت.

۹-۱۱ استفاده مجدد از نرم‌افزار

سعی افرادی که سرگرم نوشتن برنامه‌های شی‌گرا هستند، پیاده‌سازی کلاس‌های سودمند و کاربردی‌تر است. انگیزه بسیار چشمگیری وجود دارد که از کلاس‌های تدارک دیده شده توسط جامعه‌های برنامه‌نویسی استفاده شود. تعدادی زیادی از کتابخانه‌های کلاس وجود دارند و برخی در سرتاسر جهان در حال ایجاد می‌باشند. بایستی نرم‌افزار از همان آغاز کار، خوش تعریف، بدقت تست شده، بخوبی مستند شده، قابل حمل با کارایی بالا و از کامپونت‌های قابل دسترس ایجاد شده باشد. چنین نرم‌افزاری با قابلیت استفاده مجدد سرعت نرم‌افزارهای قدرتمند و با کیفیت بالا را افزایش می‌دهد. توسعه سریع برنامه‌های کاربردی (RAD) بواسطه مکانیزم قابل استفاده بودن مجدد اجزاء اهمیت خاصی پیدا کرده است.

مسائل علمی باید حل شوند، اما قبل از آن باید به بررسی دقیق مسائل استفاده مجدد از نرم‌افزار پرداخت. نیاز به فهرست کردن طرح، اخذ مجوز طرح‌ها، مکانیزم‌های حفاظتی برای اطمینان از اینکه کپی‌های اصلی از کلاس‌ها معیوب و خراب تهیه نخواهد شد. توصیف طرح‌ها را داریم تا طراحان سیستم‌های جدید بتواند به آسانی تعیین کنند که آیا شی‌های موجود می‌توانند نیاز آنها را برآورده سازند. همچنین نیاز به مکانیزم مرور داریم تا کلاس‌های موجود و در دسترس را مشخص کرده و نشان دهد که کدام کلاس به خواست طراح نرم‌افزار نزدیکتر است.

۹-۱۲ مبحث آموزشی مهندسی نرم‌افزار: شروع برنامه‌نویسی کلاس‌های سیستم ATM

در بخش‌های مبحث آموزشی مهندسی نرم‌افزار در فصل‌های یک الی هفتم، به معرفی اصول و مفاهیم بنیادین شی‌گرا و طراحی شی‌گرا بر روی سیستم ATM پرداخته شد. در ابتدای این فصل هم به بررسی برخی از جزئیات برنامه‌نویسی کلاس‌ها در C++ پرداختیم. حال شروع به پیاده‌سازی طرح شی‌گرای خود در C++ می‌کنیم. در انتهای این بخش، شما را با نحوه تبدیل دیاگرام‌های کلاس به فایل‌های سرآیند C++ آشنا خواهیم کرد. در بخش پایانی «مبحث آموزشی مهندسی نرم‌افزار» (بخش ۱۰-۱۳)، این فایل‌های سرآیند را برای هماهنگی با مفهوم ارث‌بری در برنامه‌نویسی شی‌گرا اصلاح خواهیم کرد.

رویت

می‌خواهیم تصریح‌کننده‌های دسترسی به اعضای کلاس‌ها را فراهم آوریم. در فصل سوم، به معرفی تصریح‌کننده‌های دسترسی **public** و **private** پرداختیم. این تصریح‌کننده‌ها قابل رویت یا دسترس پذیر بودن صفات و عملیات یک شی را که در اختیار شی‌های دیگر هستند، تعیین می‌کنند. قبل از اینکه بتوانیم شروع به پیاده‌سازی طرح خود نمائیم، بایستی بررسی کنیم که کدام صفات و عملیاتی از کلاس‌ها حالت **public** دارند و کدامیک حالت **private**. در فصل سوم، مشاهده کردید که معمولاً اعضای داده بایستی **private** باشند و آن دسته از توابع عضو که توسط سرویس‌گیرنده‌ها فعال می‌شوند بایستی از نوع **public**

تعیین شوند. توابع عضو که فقط توسط سایر توابع عضو یک کلاس فراخوانی می‌گردند، بعنوان «توابع یوتیلیتی» شناخته می‌شوند، با این همه، معمولاً باید **private** باشند. زبان UML از *نشانگر رویت* برای مدل کردن میزان رویت صفات و عملیات استفاده می‌کند. رویت عمومی (public) با قرار دادن یک نماد جمع (+) قبل از یک عملیات یا صفت شناخته می‌شود. رویت خصوصی (private) هم با یک نماد منفی (-) تعیین می‌گردد. در شکل ۲۰-۹ دیاگرام کلاس با اعمال نشانگرهای رویت به روز شده است. [نکته: در شکل ۲۰-۹ تمام پارامترهای عملیاتی لحاظ نشده است و این کاملاً عادی است. افزودن نشانگرهای رویت تاثیری در پارامترهای مدل شده در دیاگرام‌های کلاس در شکل‌های ۲۲-۶ الی ۲۵-۶ ندارد.]

هدایت

قبل از اینکه شروع به پیاده‌سازی طرح خود با C++ کنیم، به معرفی یکی دیگر از نمادهای UML می‌پردازیم. دیاگرام کلاس در شکل ۲۱-۹ با در اختیار گرفتن *فلش‌های هدایت* به همراه خطوط ارتباطی در میان کلاس‌های سیستم ATM به روز شده است. فلش‌های هدایت نشان می‌دهند که کدام جهت ارتباطی را می‌توان طی کرد که بر پایه مدل همکاری و دیاگرام‌های توالی است (بخش ۱۲-۷). به هنگام پیاده‌سازی یک سیستم طراحی شده با استفاده از UML، برنامه‌نویسان از فلش‌های هدایت برای کمک در تعیین اینکه کدام شی‌ها نیاز به مراجعه یا اشاره به سایر شی‌ها دارند، استفاده می‌کنند. برای مثال فلش هدایت که از کلاس **ATM** به کلاس **BankDatabase** اشاره می‌کند بر این نکته دلالت دارد که می‌توانیم از ابتدا به انتها حرکت کنیم، و در نتیجه **ATM** قادر به فعال کردن عملیات **BankDatabase** می‌شود. با این وجود، در حالیکه شکل ۲۱-۹ حاوی یک فلش هدایت از کلاس **BankDatabase** به کلاس **ATM** نیست، پس **BankDatabase** قادر به دسترسی به عملیات **ATM** نمی‌باشد. دقت کنید که وابستگی‌های موجود در یک دیاگرام کلاس که دارای فلش‌های هدایت در هر دو انتهای خود هستند یا دارای این فلش‌های نیستند، نشاندهنده هدایت دو طرفه (دو سویه) می‌باشند.

شکل ۲۰-۹ | دیاگرام کلاس با نشانگرهای رویت.

همانند دیاگرام کلاس در شکل ۲۳-۳، دیاگرام کلاس در شکل ۲۱-۹ برای حفظ سادگی کلاس‌های **BalanceInquiry** و **Deposit** را در نظر نگرفته است. هدایت اعمال شده در این کلاس‌ها بسیار به هدایت اعمال شده در کلاس **Withdrawal** نزدیک است. از بخش ۱۱-۳ بخاطر دارید که **BalanceInquiry** دارای یک رابطه با کلاس **Screen** است. می‌توانیم از طریق این رابطه از کلاس **BalanceInquiry** به کلاس **Screen** برسیم، اما نمی‌توانیم از کلاس **Screen** به کلاس **BalanceInquiry** هدایت شویم. از اینرو، اگر به سراغ مدل کردن کلاس **BalanceInquiry** در شکل ۲۱-۹ برویم، می‌توانیم یک فلش هدایت در انتهای کلاس **Screen** این رابطه قرار دهیم. همچنین بخاطر دارید یک کلاس **Deposit** با

کلاس‌های **Screen**، **Keypad** و **DepositSlot** رابطه دارد. می‌توانیم از کلاس **Deposit** به هر کدامیک از این کلاس‌ها هدایت شویم، اما عکس اینحالت صادق نیست. از اینرو فلش‌های هدایت را در انتهای رابطه با این کلاس‌ها قرار داده‌ایم.

شکل ۲۱-۹ | دیاگرام کلاس با فلش‌های هدایت.

پیاده‌سازی سیستم ATM از روی طرح UML آن

اکنون آماده هستیم تا شروع به پیاده‌سازی سیستم ATM نمائیم. ابتدا کلاس‌های موجود در دیاگرام‌های شکل‌های ۲۰-۹ و ۲۱-۹ را به فایلهای سرآیند C++ تبدیل می‌کنیم. این کد عرضه کننده «اسکلت» سیستم خواهد بود. در فصل سیزدهم، فایلهای سرآیند را برای بهره‌گیری از مفهوم ارث‌بری اصلاح خواهیم کرد.

بعنوان یک مثال، شروع به ایجاد فایل سرآیند متعلق به کلاس **Withdrawal** از روی طرح موجود این کلاس در شکل ۲۰-۹ می‌کنیم. از این تصویر برای تعیین صفات و عملیات کلاس استفاده کنیم. از مدل UML در شکل ۲۱-۹ برای تعیین وابستگی‌های موجود مابین کلاس‌ها استفاده می‌کنیم. برای هر کلاسی پنج مرحله زیر را دنبال می‌کنیم:

۱ـ از نام قرار گرفته در بخش اول یک کلاس در دیاگرام کلاس برای تعریف کلاس در فایل سرآیند استفاده می‌کنیم (شکل ۲۲-۹). از دستوردهنده‌های پیش‌پردازندهٔ **#ifndef**، **#define** و **#endif** برای اجتناب از اعمال بیش از یکبار فایل سرآیند در برنامه استفاده کنید.

۲ـ از صفات موجود در بخش دوم کلاس برای اعلان اعضای داده استفاده کنید. برای مثال، صفات **private** از کلاس **Withdrawal** عبارتند از **accountNumber** و **amount** که حاصل آن در شکل ۲۳-۹ آورده شده است.

۳- از وابستگی توصیف شده در دیاگرام کلاس برای اعلان مراجعه‌ها (یا اشاره‌گرها در صورت نیاز) به شی‌های دیگر استفاده کنید. برای مثال، مطابق شکل ۲۱-۹، کلاس **Withdrawal** می‌تواند به یک شی از کلاس **Screen**، یک شی از کلاس **Keypad**، یک شی از کلاس **CashDispenser** و یک شی از کلاس **BankDatabase** دسترسی داشته باشد. کلاس **Withdrawal** باید مبادرت به حفظ هندل‌هایی (دستگیره) به این شی‌ها کند، تا پیغام‌های به آنها ارسال نماید. از اینرو خطوط 22-19 از شکل ۲۴-۹ مبادرت به اعلان چهار مراجعه بعنوان اعضای داده **private** کرده‌اند. در پیاده‌سازی **Withdrawal** در ضمیمه G، یک سازنده این اعضای داده را با مراجعه‌های به شی‌های واقعی مقداردهی اولیه کرده است. توجه کنید که در خطوط 9-6، **#include** فایلهای سرآیند حاوی تعاریفی از کلاس‌های **Screen**، **Keypad**، **CashDispenser** و **BankDatabase** است، از اینروست که می‌توانیم مراجعه‌های به شی‌هایی از این کلاس‌ها در خطوط 22-19 اعلان کنیم.

۲۸٤ فصل نهم ـــــــ

```
7  {
8  private:
9     // attributes
10    int accountNumber; // account to withdraw funds from
11    double amount; // amount to withdraw
12
13 }; // end class Withdrawal
14
15 #endif // WITHDRAWAL_H
```

شکل ۲۳-۹ | افزودن صفات به فایل سرآیند کلاس Withdrawal.

```
1  // Fig. 9.24: Withdrawal.h
2  // Definition of class Withdrawal that represents a withdrawal transaction.
3  #ifndef WITHDRAWAL_H
4  #define WITHDRAWAL_H
5
6  #include "Screen.h" // Screen class definition
7  #include "BankDatabase.h" // BankDatabase class definition
8  #include "Keypad.h" // Keypad class definition
9  #include "CashDispenser.h" // CashDispenser class definition
10
11 class Withdrawal
12 {
13 private:
14    // attributes
15    int accountNumber; // account to withdraw funds from
16    double amount; // amount to withdraw
17
18    // references to associated objects
19    Screen &screen; // reference to ATM's screen
20    Keypad &keypad; // reference to ATM's keypad
21    CashDispenser &cashDispenser; // reference to ATM's cash dispenser
22    BankDatabase &bankDatabase; // reference to the account info database
23 }; // end class Withdrawal
24
25 #endif // WITHDRAWAL_H
```

شکل ۲٤-۹ | اعلان مراجعه‌ها به شی‌های مرتبط با کلاس Withdrawal.

توجه کنید که استفاده از اعلان رو به جلو (تا حد امکان) بجای وارد کردن کل فایل سرآیند از یک مشکل پیش‌پردازنده بنام *وارد کردن دایره‌ای* (*circular include*) جلوگیری می‌کند. این مشکل زمانی برای فایل سرآیند کلاس A رخ می‌دهد که فایل سرآیندی برای کلاس B را #include کرده باشد و برعکس برخی از پیش‌پردازنده‌ها قادر به رفع چنین دستوردهنده‌ها یا رهنمودهای #include نیستند و در نتیجه یک خطای کامپایل رخ می‌دهد. برای مثال، اگر کلاس A فقط از یک مراجعه به یک شی از کلاس B استفاده نماید، پس #include در فایل سرآیند کلاس A می‌تواند توسط یک اعلان رو به جلو از کلاس B جایگزین شود تا از مشکل circular include جلوگیری شود.

```
1  // Fig. 9.25: Withdrawal.h
2  // Definition of class Withdrawal that represents a withdrawal transaction.
3  #ifndef WITHDRAWAL_H
4  #define WITHDRAWAL_H
5
6  class Screen; // forward declaration of class Screen
7  class Keypad; // forward declaration of class Keypad
8  class CashDispenser; // forward declaration of class CashDispenser
9  class BankDatabase; // forward declaration of class BankDatabase
10
11 class Withdrawal
12 {
```

۲۸۶ فصل نهم ـــــــ

۹-۱ تعیین کنید عبارت زیر صحیح است یا اشتباه، در صورت اشتباه بودن، توضیح دهید چرا:

اگر صفتی از یک کلاس با علامت منفی (-) در دیاگرام کلاس نشانه‌گذاری شود، آن صفت بطور مستقیم از خارج از کلاس در دسترس نمی‌تواند باشد.

۹-۲ در شکل ۲۱-۹، رابطه مابین **ATM** و **Screen** براین نکته دلالت دارد که:

a) می‌توانیم از **Screen** به **ATM** هدایت شویم.

b) می‌توانیم از **ATM** به **Screen** هدایت شویم.

c) a و b هر دو صحیح هستند، هدایت دوسویه است.

d) هیچ کدامیک از موارد فوق.

۹-۳ کدی به زبان C++ بنویسید که طرح بکار رفته برای کلاس **Account** را پیاده‌سازی کند.

پاسخ خودآزمایی

۹-۱ صحیح. علامت منفی (-) نشاندهنده رویت private است.

۹-۲ b.

۹-۳ نتیجه طراحی کلاس **Account** در فایل سرآیند شکل ۲۷-۹ آورده شده است.

```
// Fig. 9.27: Account.h
// Account class definition. Represents a bank account.
#ifndef ACCOUNT_H
#define ACCOUNT_H

class Account
{
public:
   Account( int, int, double, double ); // constructor sets attributes
   bool validatePIN( int ) const; // is user-specified PIN correct?
   double getAvailableBalance() const; // returns available balance
   double getTotalBalance() const; // returns total balance
   void credit( double ); // adds an amount to the Account balance
   void debit( double ); // subtracts an amount from the Account balance
   int getAccountNumber() const; // returns account number
private:
   int accountNumber; // account number
   int pin; // PIN for authentication
   double availableBalance; // funds available for withdrawal
   double totalBalance; // funds available + funds waiting to clear
}; // end class Account

#endif // ACCOUNT_H
```

شکل ۲۷-۹ | فایل سرآیند کلاس Account براساس شکل‌های ۲۰-۹ و ۲۱-۹.

خودآزمایی

۹-۱ جاهای خالی را با کلمات مناسب پر کنید:

a) اعضای کلاس از طریق عملگر ..... در ترکیب با نام یک شی از کلاس یا از طریق عملگر..... در ترکیب با اشاره‌گر به یک شی از کلاس در دسترس قرار می‌گیرند.

b) اعضای کلاس بعنوان ..... مشخص می‌شوند و فقط برای توابع عضو کلاس و دوستان کلاس در دسترس قرار می‌گیرند.

c) اعضای کلاس بعنوان..... مشخص می‌شوند و از هر کجای قلمرو کلاس در دسترس می‌گیرند.

d) از ..... می‌توان برای تخصیص یک شی از کلاس به شی از همان کلاس استفاده کرد.

۹-۲ خطا یا خطاهای موجود در هر یک از عبارات زیر را یافته و آنها را اصلاح کنید.

a) فرض کنید نمونه اولیه زیر در کلاس Time اعلان شده‌است:

```
void ~Time(int);
```

b) عبارت زیر بخشی از تعریف کلاس Time است:

```
class Time
{
 public:
        //Function prototype:
 private:
    int hour = ;
    int minute = ;
    int second = ;
} ; // end class Time
```

c) با فرض اینکه نمونه اولیه زیر در کلاس Employee اعلان شده است.

```
int Employee(const char *, const char *);
```

پاسخ خودآزمایی

۹-۱ a) نقطه، b ->) private c) public d) تخصیص memberwise

۹-۲

a) خطا: نابود کننده‌ها اجازه برگشت دادن مقدار یا گرفتن آرگومان را ندارند.

اصلاح: حذف نوع برگشتی void و پارامتر int از اعلان.

b) خطا: اعضا نمی‌توانند بصورت صریح در تعریف کلاس مقداردهی اولیه شوند.

اصلاح: حذف مقداردهی صریح از تعریف کلاس و مقداردهی اولیه اعضای داده در یک سازنده.

c) خطا: سازنده‌ها قادر به برگشت دادن مقدار نیستند.

اصلاح : حذف نوع برگشتی int از اعلان.

تمرینات

۹-۳ منظور از عملگر تفکیک قلمرو چیست؟

۹-۴ سازنده‌ای تدارک ببینید که قادر به استفاده از زمان جاری از تابع time( ) باشد، اعلان شده در کتابخانه استاندارد C++ با سرآیند <ctime>، تا یک شی از کلاس time را مقداردهی اولیه نماید.

۹-۵ کلاسی بنام Complex ایجاد کنید که قادر به کار با مقادیر مختلط باشد. برنامه‌ای برای تست کلاس خود بنویسید. اعداد مختلط بفرم زیر هستند

realPart + imaginaryPart * i (بخش موهومی + بخش حقیقی * i)

که در آن i برابر است با $\sqrt{-1}$

از متغیرهای double برای عرضه داده private کلاس استفاده کنید. یک سازنده در نظر بگیرید که به شی از این کلاس امکان مقداردهی اولیه را در زمان اعلان فراهم آورد. باید سازنده حاوی مقادیر پیش فرض باشد. توابع عضو public را در نظر بگیرید که وظایف زیر را انجام دهند:

a) جمع دو عدد complex : بخش‌های حقیقی با یکدیگر و بخش‌های موهومی با یکدیگر جمع می‌شوند.

b) تفریق دو عدد complex : بخش حقیقی قرار گرفته در سمت راست تفریق از بخش حقیقی قرار گرفته در سمت چپ عملوند، کاسته می‌شود، و بخش موهومی قرار گرفته در سمت راست عملوند از بخش موهومی قرار گرفته در سمت چپ عملوند کاسته می‌شود.

c) چاپ اعداد complex بفرم (a,b) که در آن a بخش حقیقی و b بخش موهومی است.

۹-۶ کلاسی بنام Rational ایجاد کنید تا عملیات ریاضی را با کسرها انجام دهد. برنامه‌ای برای تست کلاس بنویسید. از متغیرهای صحیح برای عرضه داده private کلاس، numerator و denominator استفاده کنید. یک سازنده در نظر بگیرید که به شی از این کلاس امکان مقداردهی اولیه را در زمان اعلان فراهم آورد. سازنده باید حاوی مقادیر پیش فرض باشد و باید کسر را بفرم کاسته شده ذخیره کند. برای مثال، کسر 2/4

می‌تواند در یک شی بصورت 1 در numerator و 2 در denominator ذخیره شود. توابع عضو public را برای انجام وظایف زیر در نظر بگیرید:

a) جمع دو عدد Rational. نتیجه باید بفرم کاسته شده ذخیره شود.

b) تفریق دو عدد Rational. نتیجه باید بفرم کاسته شده ذخیره شود.

c) ضرب دو عدد Rational. نتیجه باید بفرم کاسته شده ذخیره شود.

d) تقسیم دو عدد Rational. نتیجه باید بفرم کاسته شده ذخیره شود.

e) چاپ اعداد Rational. بفرم a/b ، که در آن a صورت و b مخرج کسر است.

f) چاپ اعداد Rational. با فرمت اعشاری.

۹-۷ برنامه شکل‌های ۸-۹ و ۹-۹ را به نحوی اصلاح کنید که حاوی تابع عضو tick باشد تا زمان ذخیره شده درشی Time را در هر ثانیه افزایش دهد. بایستی شی time همیشه در وضعیت پایدار باقی بماند. برنامه‌ای بنویسید که تابع عضو tick را در حلقه‌ای که زمان را در فرمت استاندارد در هر بار تکرار چاپ می‌کند، تست نماید تا از عملکرد صحیح آن مطمئن گردیم . حتماً حالات زیر تست شوند:

a) ورود به دقیقه بعد.

b) ورود به ساعت بعد.

c) ورود به روز بعد (یعنی 11:59:59 PM به 12:00:00AM).

۹-۸ کلاس Date بکار رفته در شکل‌های ۹-۱۷ و ۹-۱۸ را برای انجام تست خطا در مقداردهی مقادیر برای اعضای داده month ، day و year تغییر دهید. همچنین تابع عضو nextDay را برای افزایش یک روز در هر بار در نظر بگیرید. شی Date باید همیشه در وضعیت پایدار باقی بماند. برنامه‌ای بنویسید که تابع nextDay را در حلقه‌ای که

تاریخ جاری را در هر بار تکرار چاپ می‌کند، تست نماید تا از عملکرد صحیح nextDay مطمئن گردیم. حتماً حالات زیر تست شوند:

a) ورود به ماه بعد.

b) ورود به سال بعد.

۹-۹ کلاس اصلاح شده Time در تمرین ۷-۹ و کلاس اصلاح شده Date در تمرین ۸-۹ را بصورت یک کلاس بنام DateAndTime با هم ترکیب کنید. اگر زمان برای ورود به روز بعدی افزایش یابد، تابع tick را برای فراخوانی تابع nextDay اصلاح کنید. توابع printStandard , PrintUniversal را برای چاپ تاریخ و زمان اصلاح کنید. برنامه‌ای بنویسید که کلاس جدید DateAndTime را تست کند.

۱۰-۹ توابع set بکار رفته در کلاس Time شکل‌های ۸-۹ و ۹-۹ را به نحوی اصلاح کنید تا در صورتیکه مبادرت به مقداردهی یک شی از کلاس Time با مقدار نامعتبر شود، خطای متناسب با آن برگشت داده شود. برنامه‌ای برای تست این نسخه از کلاس بنویسید. پیغام‌های خطا را در صورت برگشت مقادیر خطا از توابع set بنمایش درآورید.

۱۱-۹ کلاس بنام Rectangle با صفات width، length ایجاد کنید که هر یک دارای مقدار پیش فرض 1 هستند. توابع عضوی در نظر بگیرید که اقدام به محاسبه مساحت و محیط مستطیل کنند. همچنین توابع get و set را برای صفات length, width در نظر بگیرید. بایستی توابع set مراقب باشند که length, width دارای مقادیر اعشاری بزرگتر از 0.0 و کمتر از 20.0 باشند.

۱۲-۹ کلاس پیشرفته Rectangle را به نسبت کلاس یاد شده در تمرین ۱۱-۹ ایجاد کنید. این کلاس فقط مختصصات دکارت را برای چهار گوشه مستطیل ذخیره می‌کند. سازنده مبادرت به فراخوانی یک تابع set می‌کند که مجموعه چهار مختصصاتی را پذیرفته و بررسی می‌کند که هر کدام از آنها در اولین ربع قرار دارند و مقدار آنها بزرگتر از 2.0 نمی‌باشد. همچنین تابع set بررسی می‌کند که مختصات تدارک دیده شده، خاص یک مستطیل باشند. توابع عضو در نظر بگیرید که مبادرت به محاسبه طول، عرض، محیط و مساحت نمایند. همچنین یک تابع مسند بنام square در نظر بگیرید که تعیین کنید آیا با مستطیل طرف هستیم یا مربع.

۱۳-۹ کلاس Rectangle مطرح شده در تمرین ۱۲-۹ را برای داشتن تابع draw اصلاح کنید. این تابع اقدام به نمایش مستطیل در درون یک جعبه 25 در 25 می‌کند که بخشی از ربع اول مستطیل در آن مقیم است. از تابع setFillCharacter برای پر کردن داخل مستطیل با کاراکتر مشخص شده استفاده کنید. همچنین از تابعی بنام setPerimeterCharacter برای تعیین کاراکتری که از آن برای رسم بدنه یا حاشیه مستطیل استفاده خواهد شد، کمک بگیرید.

۱۴-۹ کلاس بنام HugeInteger ایجاد کنید که از یک آرایه 40 عنصری از ارقام برای ذخیره‌سازی ارقام به بزرگی اعداد 40 رقمی استفاده کند. توابع عضو substract , add, output, input را در نظر بگیرید. برای مقایسه شی‌های HugeInteger، توابع isEqualTo، isNotEqualTo، isGreaterThan، isGreaterThanOrEqualTo, isLessThan و isLessThanOrEqualTo را در نظر بگیرید. هر یک از این توابع یک تابع پیشگو یا مسند هستند که در صورت

برقرار بودن رابطه مابین دو شی IntegerHuge مقدار true و در صورت برقرار نبودن رابطه مقدار false برگشت می‌دهند. همچنین تابع مسند isZero را در نظر بگیرید. در صورت تمایل می‌توانید توابع عضو modulus , divide, multiply را هم بکار گیرید.