فصل
دوازدهم

برنامه‌نویسی شی‌گرا: توارث

اهداف

- ایجاد کلاس‌ها با ارث‌بری از کلاس‌های جدید.
- نحوه استفاده مجدد از نرم‌افزار توسط توارث.
- مفهوم کلاس‌های مبنا و کلاس‌های مشتق شده و رابطه مابین آنها.
- عضو تصریح کننده دسترسی protected.
- کاربرد سازنده‌ها و نابود کننده‌ها در سلسله مراتب توارث.
- تفاوت مابین توارث public، protected و private.
- کاربرد توارث در بهینه‌سازی نرم‌افزارهای موجود.

رئوس مطالب

۱-۱۲ مقدمه
۲-۱۲ کلاس‌های مبنا وکلاس‌های مشتق شده
۳-۱۲ اعضای protected
٤-۱۲ رابطه مابین کلاس‌های مبنا و کلاس‌های مشتق شده
۱-٤-۱۲ ایجاد و استفاده از کلاس CommissionEmplyee
۲-٤-۱۲ ایجاد کلاس BasePlusCommissionEmployee بدون استفاده از توارث
۳-٤-۱۲ ایجاد سلسه مراتب توارث CommissionEmployee-BasePlusCommissionEmployee
٤-٤-۱۲ ایجاد سلسله مراتب توارث CommissionEmpolyee-BasePlusCommissionEmployee با استفاده از داده protected
٥-٤-۱۲ ایجاد سلسله مراتب توارث CommissionEmployee-BasePlusCommissionEmployee با استفاده از داده private
٦-۱۲ توارث public، protected و private
۷-۱۲ مهندسی نرم‌افزار به کمک توارث

۱-۱۲ مقدمه

در این فصل، بحث خود را با معرفی یکی از ویژگی‌های مهم برنامه‌نویسی شی‌گرا (OOP) یعنی توارث یا ارث‌بری آغاز می‌کنیم. توارث فرمی از بکارگیری مجدد نرم‌افزار است که در آن کلاس‌های ایجاد شده موجودیت و رفتار خود را براساس اطلاعات یک کلاس موجود بدست آورده و در صورت نیاز حاوی قابلیت‌های جدید هستند. بکارگیری مجدد نرم‌افزار سبب کاهش مدت زمان توسعه نرم‌افزار شده و کیفیت آنرا بطور موثری افزایش می‌دهد.

به هنگام ایجاد یک کلاس، بجای نوشتن کامل متغیرهای نمونه و متدها، برنامه‌نویس می‌تواند تعیین کند که کلاس جدید بایستی متغیرها، خصوصیات و متدهای کلاس را از یک کلاس دیگر به ارث ببرد. کلاسی که قبلاً تعریف شدهٔ، کلاس *مبنا* نامیده می‌شود و کلاس جدید بعنوان یک کلاس *مشتق شده* شناخته می‌شود. (در زبان‌های برنامه‌نویسی دیگری همانند جاوا، به کلاس مبنا، *سوپرکلاس* و کلاس مشتق شدهٔ، *زیرکلاس* گفته می‌شود.) پس از ایجاد کلاس، هر کلاس مشتق شده می‌تواند تبدیل به یک کلاس مبنا برای کلاس‌هایی شود که بعدها از آن مشتق خواهند شد. یک کلاس مشتق شده که دارای متغیرها، خصوصیات و متدهای منحصر بفرد است، معمولاً بزرگتر از کلاس مبنای خود می‌باشد. از اینرو، یک کلاس مشتق شده به نسبت کلاس مبنای خود تخصصی‌تر است و نشاندهنده گروهی خاص و مرتبط از

شی‌ها است. عموماً یک کلاس مشتق شدهٔ حاوی رفتار کلاس مبنای خود به همراه قابلیت‌ها و رفتارهای دیگر است. یک کلاس مبنای *مستقیم*، کلاس مبنائی است که کلاس‌های مشتق شده بصورت صریح از آن ارث بری دارند. یک کلاس مبنای *غیرمستقیم*، از دو یا بیش از چند سطح سلسله مراتب کلاس ارث بری دارد. در توارث یگانه، یک کلاس از یک کلاس مبنا مشتق می‌شود. C++ از توارث چندگانه پشتیبانی می‌کند. توارث چندگانه زمانی اتفاق می‌افتد که یک کلاس از بیش از یک کلاس مبنا مشتق شود.

زبان **C++** سه نوع توارث یا ارث‌بری را پیشنهاد می‌کند: **public**، **protected** و **private** در این فصل بحث ما متمرکز بر روی ارث‌بری **public** بوده و تا حدودی به توضیح دو نوع دیگر هم خواهیم پرداخت. در فصل ۲۱، ساختمان‌های داده، نشان خواهیم داد که چگونه ارث‌بری **private** می‌تواند بعنوان جانشینی برای ترکیب بکار گرفته شود. از فرم سوم، یعنی ارث‌بری **protected** بندرت استفاده می‌شود. در ارث‌بری **public**، هر شی از یک کلاس مشتق شده یک شی از کلاس مبنای مشتق شده نیز محسوب می‌شود. با این وجود، شی‌های کلاس مبنا، شی‌های از کلاس‌ها مشتق شده از خودشان نیستند. برای مثال اگر وسیله نقلیه یک کلاس مبنا باشد و اتومبیل یک کلاس مشتق شده، پس تمام اتومبیل‌ها، وسیله نقلیه محسوب می‌شوند، اما تمام وسایل نقلیه اتومبیل نیستند. همانطوری که به آموزش برنامه‌نویسی شی‌گرا در فصل ۱۲ و ۱۳ ادامه می‌دهیم از مزایای موجود در این روابط در انجام کارهای جالب توجه استفاده خواهیم کرد.

تجربه ایجاد سیستم‌های نرم‌افزاری نشان داده است که بخش قابل توجهی از کد در ارتباط با حل موارد خاص هستند. زمانیکه برنامه‌نویسان گرفتار موارد خاص می‌شوند، جزئیات کار می‌تواند کل موضوع را مبهم سازد. با برنامه‌نویسی شی‌گرا، تمرکز برنامه‌نویسان بر روی نقاط مشترک شی‌ها در سیستم است بجای اینکه به موارد خاص متکی باشند.

مابین رابطه *است–یک* (*is-a*) و رابطه *دارد–یک* (*has-a*) تفاوت قائل هستیم. رابطه is-a نشاندهنده توارث است. در این رابطه با یک شی از یک کلاس مشتق شده می‌توان بعنوان یک شی از کلاس مبنا خود رفتار کرد، برای مثال اتومبیل یک وسیله است (رابطه است–یک)، از اینرو خصوصیات و رفتار یک وسیله نقلیه، خصوصیات اتومبیل هم محسوب می‌شوند. در مقابل رابطه has-a قرار دارد که نشاندهنده ترکیب می‌باشد. (ترکیب در فصل ۱۰ توضیح داده شده است). در رابطه has-a، یک شی حاوی یک یا چندین شی از کلاس‌های دیگر بعنوان عضو است. برای مثال، یک اتومبیل دارای کامپونت‌های متعددی است، دارای چرخ، پدال گاز، موتور و اجزای دیگر است.

ممکن است توابع عضو کلاس مشتق شده نیازمند دسترسی به اعضای داده و توابع عضو کلاس مبنای خود داشته باشند. یک کلاس مشتق شده می‌تواند به اعضای غیر ***private***، کلاس مبنای خود دسترسی

داشته باشد. اعضای کلاس مبنا که قادر به دستیابی به توابع عضو یک کلاس مشتق شده از کلاس مبنا به طریق توارث نیستند، باید بصورت ***private*** در کلاس مبنا اعلان شوند.

مهندسی نرم‌افزار

توابع عضو یک کلاس مشتق شده نمی‌توانند بصورت مستقیم به اعضای ***private*** کلاس مبنا دسترسی یابند.

یکی از مشکلاتی که در توارث وجود دارد این است که کلاس مشتق شده داده‌های عضو و توابع عضوی را که به آنها نیاز ندارد به ارث می‌برد. این وظیفه طراح کلاس است تا مطمئن شود قابلیت‌های تدارک دیده شده توسط کلاس، مناسب، کلاس‌های هستند که بعدها از آن مشتق خواهند شد. حتی در زمانیکه خصیصه یا متد کلاس مبنا برای کلاس‌های مشتق شده مناسب طراحی شده باشند، گاهاً کلاس‌های مشتق شده نیاز به توابع یا خصوصیات خاص خود دارند تا وظیفه خود را به انجام برسانند. در چنین مواردی، تابع عضو کلاس مبنا می‌تواند در کلاس مشتق شده بازنویسی (تعریف مجدد) شود.

۱۲-۲ کلاس‌های مبنا و کلاس‌های مشتق شده

غالباً یک شی از یک کلاس، به همان اندازه شیی از یک کلاس دیگر است. برای مثال در علم هندسه، یک مستطیل یک چهار ضلعی است. از اینرو می‌توان گفت که کلاس **Rectangle** از کلاس **Quadrilateral** ارث‌بری داشته است. در اینحالت کلاس **Quardrilateral** کلاس مبنا است و کلاس **Rectangle** کلاس مشتق شده از آن می‌باشد. مستطیل نوع خاصی از چهار ضلعی است، اما تصور اشتباهی است که بگوییم که یک چهارضلعی یک مستطیل است، چرا که چهارضلعی می‌تواند یک متوازی‌الاضلاع یا نوع دیگری از **Quardrilaterd** باشد. در جدول شکل ۱-۱۲ لیستی از چند مثال ساده در ارتباط با کلاس‌های مبنا و کلاس‌های مشتق شده، به نمایش درآمده است.

کلاس مبنا	کلاس‌های مشتق شده
Student	GranduateStudent
	UndergraduateStudent
Shape	Circle
	Triangle
	Rectangle
	Sphere
	Cube
Loan	CarLoan
	HomeImprovementLoan
	MortgageLoan

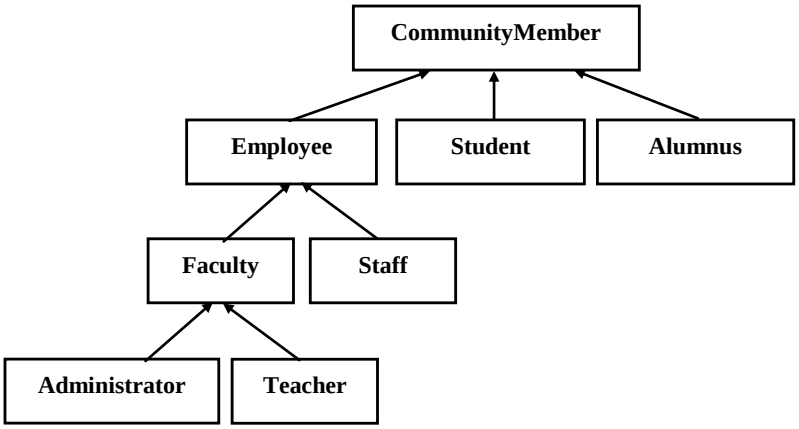
Employee	Faculty
	Staff
Account	CheckingAccount
	SavingsAccount

شکل ۱-۱۲ مثال‌های از توارث.

بدلیل اینکه هر شی از کلاس مشتق شده، شی از کلاس مبنای خود است و یک کلاس مبنا می‌تواند تعداد زیادی کلاس مشتق شده داشته باشد، از اینرو، مجموعه شی‌های به نمایش در آمده توسط کلاس مبنا بیشتر از مجموعه شی‌های عرضه شده توسط هر کلاس مشتق شده از خود کلاس مبنا است. برای مثال، کلاس مبنای **Vehicle** نشاندهنده تمام وسایل نقلیه، شامل اتومبیل‌ها، کامیون‌ها، دوچرخه‌ها و غیره است. در مقابل، کلاس مشتق شده **Car** فقط نشاندهنده زیر مجموعه کوچکی از تمام **Vehicle** (وسایل نقلیه) است.

رابطه توارث را می‌توان به فرم یک سلسله مراتب درختی به نمایش در آورد. موجودیت یک کلاس در رابطه توارث با کلاس‌های مشتق شده آن مشخص می‌گردد. اگر چه کلاس‌ها می‌توانند موجودیت‌های مستقلی داشته باشند، اما زمانیکه در ترتیبات توارثی بکار گرفته می‌شوند، با کلاس‌های دیگر مرتبط می‌گردند.

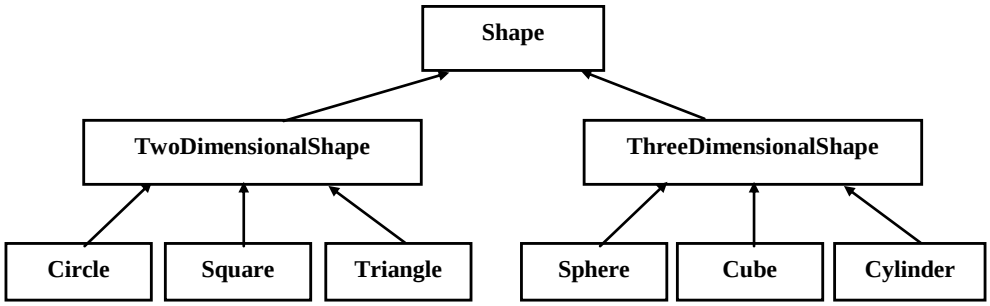
اجازه دهید تا به بررسی و ایجاد یک سلسله مراتب توارث ساده در پنج سطح بپردازیم (عرضه شده با دیاگرام کلاس UML در شکل ۲-۱۲). یک جامعه دانشگاهی را با صدها عضوی که دارد در نظر بگیرید. این اعضا متشکل از کارمندان، فارغ‌التحصیلان و دانشجویان هستند. کارمندان می‌توانند اعضای هیت علمی باشند یا کارمند ساده. اعضای هیت علمی می‌توانند، مدیر یا استاد باشند. با این وجود، برخی از مدیران می‌توانند در کلاس‌ها تدریس کنند. دقت کنید که از توارث مضاعف به فرم **AdministratorTeacher** استفاده کرده‌ایم. این ساختار سازماندهی، نمایانگر یا سلسله مراتب توارث است و در شکل ۲-۱۲ دیده می‌شود. دقت کنید که سلسله مراتب توارث می‌تواند حاوی کلاس‌های دیگری نیز باشد. برای مثال، دانشجویان می‌توانند، در زمرهٔ دانشجویان فارغ‌التحصیل یا دانشجویان فارغ‌التحصیل نشده قرار گیرند.



۳۱۸فصل دوازدهم ـــــــ

نمی‌باشند، اما هنوز هم این اعضای **private** از کلاس مبنا به ارث برده می‌شوند. تمام دیگر اعضای کلاس مبنا، عضو دسترسی اصلی خود را به هنگام تبدیل شدن به اعضای کلاس مشتق شده، حفظ و نگهداری می‌کنند (برای مثال، اعضای **public** در کلاس مبنا، تبدیل به اعضای **public** در کلاس مشتق شده می‌شوند و همانطوری که بزودی خواهید دید، اعضای **protected** در کلاس مبنا، تبدیل به اعضای **protected** در کلاس مشتق شده خواهند شد). در میان این اعضای کلاس مبنای به ارث برده شده، کلاس مشتق شده می‌تواند در اعضای **private** کلاس مبنا دستکاری نماید (اگر این اعضای به ارث برده شده چنین قابلیتی در کلاس مبنا داشته باشند). امکان تلقی کردن شی‌های کلاس مبنا و شی‌های کلاس مشتق شده بطریقه مشابه وجود دارد.

در فصل دهم بطور اجمال در مورد رابطه وجود داشتن بحث کردیم که در آن کلاس‌ها اعضای داشتند کهشی‌های از کلاس‌های دیگر بودند. چنین روابطی با بکارگیری ترکیب از کلاس‌های موجود، اقدام به ایجاد کلاس‌ها می‌کنند. برای مثال، گفتن اینکه کلاس **Employee** از کلاس **BirthDate** یا از **TelephoneNumber** است، کاملاً اشتباه می‌باشد. با این وجود، مناسب خواهد بود که بگوییم **Employee** دارای یک **BirthDate** است و اینکه **Employee** دارای **TelephoneNumber** می‌باشد.



شکل ۳-۱۲ | بخشی از سلسله مراتب کلاس Shape.

۳-۱۲ اعضای protected

در فصل سوم به توضیح اصلاح کننده‌های دسترسی **public** و **private** پرداختیم. اعضای یک کلاس **public** از هر کجای برنامه که دارای مراجعه‌ای به شی از کلاس مبنا یا یکی از کلاس‌های مشتق شده از آن است در دسترس هستند. اعضای یک کلاس مبنای **private** فقط در درون بدنه کلاس مبنا و دوستان

(friends) آن در دسترس می‌باشند. در این بخش، به توضیح عضو اصلاح کننده دسترسی دیگری بنام **protected** می‌پردازیم.

دسترسی **protected** عرضه کننده یک سطح حفاظتی میانی مابین دسترسی‌های **public** و **private** است. اعضای یک کلاس مبنای **protected** می‌توانند فقط در کلاس مبنا یا در هر کلاس مشتق شده از کلاس یا دوستان آن کلاس در دسترس قرار گیرند.

معمولاً متدهای کلاس مشتق شده می‌توانند بسادگی به اعضای **public** و **protected** کلاس مبنا با استفاده از اسامی اعضا مراجعه داشته باشند. هنگامی که یک تابع عضو از کلاس مشتق شده می‌خواهد به یک عضو کلاس مبنا دسترسی یابد می‌تواند با قرار دادن نام عضو کلاس مبنا به همراه نام کلاس مبنا و عملگر باینری تفکیک قلمرو (::) اینکار را انجام دهد. در بخش ۴-۱۲ در ارتباط با دسترسی به اعضای مجدد تعریف شده از کلاس مبنا می‌پردازیم و در بخش ۴-۴-۱۲ از داده **protected** شده استفاده می‌کنیم.

٤-۱۲ ارتباط مابین کلاس‌های مبنا و کلاس‌های مشتق شده

در این بخش، از یک سلسله مراتب توارث که حاوی انواع کارمندان در برنامه پرداخت دستمزد یک شرکت است استفاده می‌کنیم تا به توضیح رابطه موجود مابین یک کلاس مبنا و یک کلاس مشتق شده بپردازیم. کارمندان کمیسیون (یا کارمندان حق‌العمل‌کار) که بعنوان شی‌های از کلاس مبنا عرضه خواهند شد، حقوق خود را بصورت درصدی از فروش دریافت می‌کنند، در حالیکه کارمندان کمیسیون مبتنی بر پایه حقوق (که بعنوان شی‌های از کلاس مشتق شده عرضه خواهند شد) یک حقوق پایه به همراه درصدی از فروش را دریافت می‌کنند. بحث خود را که در ارتباط با رابطه موجود مابین این دو نوع کارمند است به دقت و به کمک پنج مثال مطرح می‌کنیم:

۱- در اولین مثال، یک کلاس **CommissionEnployee** ایجاد می‌کنیم که حاوی اعضا داده **private** بعنوان نام، نام خانوادگی، شماره تامین اجتماعی، نرخ کمیسیون (درصد) و مبلغ ناخالص (یعنی مجموع) فروش است.

۲- در مثال دوم اقدام به تعریف کلاس **BasePlusCommissionEmployee** می‌کنیم که حاوی اعضای داده **private** بعنوان نام، نام خانوادگی، شماره تامین اجتماعی، نرخ کمیسیون، مبلغ ناخالص فروش و حقوق پایه است. این کلاس را با نوشتن خط به خط کدهای مورد نیاز کلاس ایجاد می‌کنیم. بزودی خواهید دید که ایجاد این کلاس به آسانی از طریق ارث‌بری از کلاس **CommissionEmployee** امکان‌پذیر است.

۳- در مثال سوم یک نسخه جدید از کلاس **BasePlusComissionEmployee** تعریف می‌کنیم که مستقیماً از کلاس **CommissionEmployee** ارث‌بری دارد و مبادرت به دسترسی به اعضای **private** این کلاس می‌کند، که نتیجه اینکار خطای کامپایل خواهد بود، چرا که کلاس مشتق شده نمی‌تواند به داده **private** (خصوصی) کلاس مبنا دسترسی پیدا کند.

۴- مثال چهارم نشان می‌دهد که اگر داده **CommissionEmployee** بصورت **protected** (حفاظت شده) اعلان شود، نسخه جدید کلاس **BasePlusCommissionEmployee** که از کلاس **CommissionEmpolyee** ارث‌بری دارد می‌تواند مستقیماً به داده آن دسترسی پیدا کند. به همین منظور، نسخه جدیدی از کلاس **CommssionEmployee** را با داده **protected** تعریف می‌کنیم. هر دو نسخه ارث‌بر و غیر ارث‌بر کلاس‌های **BasePlusCommissionEmployee** دارای قابلیت‌های یکسان هستند، اما نشان خواهیم داد که ایجاد و مدیریت نسخه ارث‌بر بسیار آسانتر است.

۵- پس از بحث در مورد قواعد استفاده از داده **protected**، مثال پنجمی ایجاد می‌کنیم که اقدام به تنظیم اعضای داده **CommissionEmployee** برای برگشت به حالت **private** می‌کند تا مهندسی نرم‌افزار مناسبی داشته باشیم. در این مثال نشان داده می‌شود که کلاس مشتق شده **BasePlusCommissionEmpolyee** می‌تواند توسط توابع **public** کلاس مبنا به منظور دستکاری کردن داده خصوصی **CommissionEmployee** بکار گرفته شود.

۱-٤-۱۲ ایجاد و استفاده از کلاس ComissionEmployee

اجازه دهید تا ابتدا به بررسی تعریف کلاس **ComissionEmployee** بپردازیم (شکل‌های ۴-۱۲ و ۵-۱۲) فایل سرآیند **CommissionEmolpyee** (شکل ۴-۱۲) مشخص کننده سرویس‌های سراسری کلاس **CommssionEmployee** است که شامل یک سازنده (خطوط 13-12) و توابع عضو **earnings** (خط 30) و **print** (خط 31) است.

در خطوط 28-15 توابع سراسری *get* و *set* برای کار با اعضای داده کلاس بنام **firstName**، **lastName**, **socialSecurityNumber** (شماره تامین اجتماعی)، **grossSales** (ناخالص فروش) و **commissonRate** (نرخ کمیسیون) است (اعلان شده در خطوط 77-33). فایل سرآیند **CommissionEmployee** مشخص می‌کند که هر یک از این اعضای داده حالت **private** (خصوصی) دارند، از اینرو شی‌های سایر کلاس‌ها نمی‌توانند مستقیماً به این داده دسترسی پیدا کنند. اعلان اعضای داده خصوصی و تدارک دیدن توابع *get* و *set* غیر خصوصی برای دستکاری کردن و اعتبارسنجی اعضا داده و به داشتن مهندسی نرم‌افزار مناسب کمک می‌کند. توابع عضو **setGrossSales** (تعریف شده در خطوط

57-60 از شکل ۵-۱۲) و **setCommissonRate** (تعریف شده در خطوط 69-72 از شکل ۵-۱۲)، قبل از اینکه مبادرت به تخصیص مقادیر به اعضای داده **grossSales** و **commissionRate** کنند، اعتبارسنجی آرگومان را انجام می‌دهند.

تعریف سازنده **CommissionEmployee** عمداً از گرامر مقداردهی کننده اولیه عضو در چند مثال اول این بخش استفاده نکرده است، از اینروست که می‌توانیم توضیح دهیم که چگونه تصریح کننده‌های **private** و **protected** در دسترسی به اعضا در کلاس‌های مشتق شده تاثیر می‌گذارند. همانطوری که در شکل ۵-۱۲، خطوط 13-15 مشاهده می‌کنید، اقدام به تخصیص مقادیری به اعضای داده **firstName**، **lastName** و **socialSecurityNumber** در بدنه سازنده کرده‌ایم. در انتهای این بخش به سراغ استفاده از لیست‌های مقداردهی کننده اولیه در سازنده‌ها خواهیم رفت.

```
1  // Fig. 12.4: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
26
27    void setCommissionRate( double ); // set commission rate (percentage)
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

شکل ٤-۱۲ | فایل سرآیند کلاس CommssionEmployee.

```
1  // Fig. 12.5: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
```

```
using std::cout;

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

// constructor
CommissionEmployee::CommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate )
{
   firstName = first; // should validate
   lastName = last; // should validate
   socialSecurityNumber = ssn; // should validate
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate
} // end CommissionEmployee constructor

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

```

۳۲٤فصل دوازدهم ـــــــ

earings را با استفاده از مقادیر به روز شده اعضای داده **grossSales** و **commissionRate** بنمایش در می‌آورد.

```
// Fig. 12.6: fig12_06.cpp
// Testing class CommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

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

int main()
{
   // instantiate a CommissionEmployee object
   CommissionEmployee employee(
      "Sue", "Jones", "222-22-2222", 10000, .06 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
     << "\nCommission rate is " << employee.getCommissionRate() << endl;

   employee.setGrossSales( 8000 ); // set gross sales
   employee.setCommissionRate( .1 ); // set commission rate

   cout << "\nUpdated employee information output by print function: \n"
      << endl;
   employee.print(); // display the new employee information

   // display the employee's earnings
   cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

   return 0;
} // end main
```

```
Employee information obtained by get functions:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Update employee information output by print function:

commission employee: Sue Jones
social security number: 222-22-2222
commission rate: 0.10

Employee's earnings: $800.00
```

شکل ٦-١٢ | برنامه تست کننده کلاس CommssionEmployee.

٢-٤-١٢ ایجاد کلاس BasePlusCommssionEmployee بعنوان ارث‌بری

در این بخش به سراغ قسمت دوم از مقدمه و معرفی ارث‌بری می‌رویم و آنرا با ایجاد دو تست کلاس **BasePlusCommssionEmployee** (شکل‌های ۷-۱۲ و ۸-۱۲) انجام می‌دهیم که حاوی نام، نام خانوادگی، شماره تامین اجتماعی، میزان فروش ناخالص، نرخ کمیسیون و حقوق پایه است. (این کلاس را بصورت مستقل و کاملاً جدید ایجاد می‌کنیم).

تعریف کلاس BasePlusCommissionEmploye

فایل سرآیند **BasePlusCommssionEmployee** در شکل ۷-۱۲ تصریح کننده سرویس‌های سراسری (public) کلاس است که شامل سازنده این کلاس (خطوط 14-13) و توابع عضو **earings** (خط 34) و **print** (خط 35) است.

```
1  // Fig. 12.7: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class definition represents an employee
3  // that receives a base salary in addition to commission.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 class BasePlusCommissionEmployee
11 {
12 public:
13    BasePlusCommissionEmployee( const string &, const string &,
14       const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16    void setFirstName( const string & ); // set first name
17    string getFirstName() const; // return first name
18
19    void setLastName( const string & ); // set last name
20    string getLastName() const; // return last name
21
22    void setSocialSecurityNumber( const string & ); // set SSN
23    string getSocialSecurityNumber() const; // return SSN
24
25    void setGrossSales( double ); // set gross sales amount
26    double getGrossSales() const; // return gross sales amount
27
28    void setCommissionRate( double ); // set commission rate
29    double getCommissionRate() const; // return commission rate
30
31    void setBaseSalary( double ); // set base salary
32    double getBaseSalary() const; // return base salary
33
34    double earnings() const; // calculate earnings
35    void print() const; // print BasePlusCommissionEmployee object
36 private:
37    string firstName;
38    string lastName;
39    string socialSecurityNumber;
40    double grossSales; // gross weekly sales
41    double commissionRate; // commission percentage
42    double baseSalary; // base salary
43 }; // end class BasePlusCommissionEmployee
44
45 #endif
```

شکل ۷-۱۲ | فایل سرآیند کلاس BasePlusCommssionEmployee.

خطوط 32-16 توابع سراسری *get* و *set* را برای اعضای داده خصوصی کلاس (اعلان شده در خطوط 42-37) بنام‌های **firstName**، **lastName**، **socialSecurityNumber**، **grossSales** (فروش ناخالص)، **commissionRate** و (نرخ کمیسیون) **baseSalary** (حقوق پایه) اعلان کرده‌اند. این متغیرها و توابع عضو تمام ویژگیهای ضروری یک کارمند که دارای حقوق پایه و کمیسیون دریافتی است را کپسوله می‌کند. به شباهت موجود مابین این کلاس و کلاس **CommssionEmployee** (شکل‌های ۴-۱۲ و ۵-۱۲) توجه کنید. در این مثال، هنوز قصد توضیح شباهت‌ها را نداریم.

تابع عضو **earnings** (تعریف شده در خطوط 99-96 از شکل ۸-۱۲) مبادرت محاسبه حقوق این نوع کارمند می‌کند. خط 98 نتیجه افزودن حقوق پایه کارمند به حاصلضرب نرخ کمیسیون و فروش ناخالص را برگشت می‌دهد.

تست کلاس BasePlusCommssionEmployee

شکل ۹-۱۲ تست کننده کلاس **BasePlusCommssionEmployee** است. خطوط 18-17 مبادرت به ایجاد یک شی **Employee** از این کلاس کرده و "Bob" و "Lewis"، "333-33-3333"، "5000"، ".04" و 300 را بترتیب بعنوان نام، نام خانوادگی، شماره تامین اجتماعی، فروش ناخالص، نرخ کمیسیون و حقوق پایه به سازنده ارسال می‌کنند. خطوط 31-24 از توابع *get* این کلاس برای بازیابی مقادیر اعضای داده شی در خروجی استفاده می‌کنند. خط 33 تابع عضو **setBaseSalary** را برای تغییر دادن حقوق پایه احضار می‌کند.

تابع عضو **setBaseSalary** (شکل ۸-۱۲، خطوط 87-84) ما را مطمئن می‌سازد که داده عضو **baseSalary** (حقوق پایه) هرگز یک مقدار منفی نباشد، چرا که حقوق پایه یک کارمند نمی‌تواند منفی باشد. خط 37 از شکل ۹-۱۲ تابع عضو **print** را برای چاپ (نمایش) اطلاعات به روز شده کلاس و خط 40 تابع عضو **earnings** را برای نمایش حقوق کارمند فراخوانی می‌کند.

```
1  // Fig. 12.8: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13 {
14    firstName = first; // should validate
15    lastName = last; // should validate
16    socialSecurityNumber = ssn; // should validate
```

```
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set first name
void BasePlusCommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string BasePlusCommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void BasePlusCommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string BasePlusCommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void BasePlusCommissionEmployee::setSocialSecurityNumber(
   const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string BasePlusCommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void BasePlusCommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double BasePlusCommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void BasePlusCommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double BasePlusCommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
```

```
87 } // end function setBaseSalary
88
89 // return base salary
90 double BasePlusCommissionEmployee::getBaseSalary() const
91 {
92    return baseSalary;
93 } // end function getBaseSalary
94
95 // calculate earnings
96 double BasePlusCommissionEmployee::earnings() const
97 {
98    return baseSalary + ( commissionRate * grossSales );
99 } // end function earnings
100
101  // print BasePlusCommissionEmployee object
102  void BasePlusCommissionEmployee::print() const
103  {
104     cout << "base-salaried commission employee: " << firstName << ' '
105        << lastName << "\nsocial security number: " << socialSecurityNumber
106        << "\ngross sales: " << grossSales
107        << "\ncommission rate: " << commissionRate
108        << "\nbase salary: " << baseSalary;
109  } // end function print
```

شکل ۸-۱۲ | کلاس BasePlusCommssionEmployee نشاندهنده کارمندی است که علاوه بر حقوق پایه، کمیسیونی هم دریافت می‌کند.

```
1  // Fig. 12.9: fig12_09.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16    // instantiate BasePlusCommissionEmployee object
17    BasePlusCommissionEmployee
18       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20    // set floating-point output formatting
21    cout << fixed << setprecision( 2 );
22
23    // get commission employee data
24    cout << "Employee information obtained by get functions: \n"
25       << "\nFirst name is " << employee.getFirstName()
26       << "\nLast name is " << employee.getLastName()
27       << "\nSocial security number is "
28       << employee.getSocialSecurityNumber()
29       << "\nGross sales is " << employee.getGrossSales()
30       << "\nCommission rate is " << employee.getCommissionRate()
31       << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33    employee.setBaseSalary( 1000 ); // set base salary
34
35    cout << "\nUpdated employee information output by print function: \n"
36       << endl;
37    employee.print(); // display the new employee information
38
39    // display the employee's earnings
40    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42    return 0;
```

```
43 } // end main
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Update employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

شکل ۹-۱۲ | برنامه تست کننده کلاس BasePlusCommssionEmployee.

بررسی شباهت‌های مابین کلاس BasePlusCommssionEmployee و کلاس CommssionEmployee

به میزان کد بکار رفته برای کلاس **BasePlusCommssionEmployee** (شکل‌های ۷-۱۲ و ۸-۱۲) دقت کنید که تقریباً برابر با کد بکار رفته برای کلاس **CommssionEmployee** است (شکل‌های ۴-۱۲ و ۵-۱۲) برای مثال، در کلاس **BasePlusCommssionEmployee** اعضای داده خصوصی عبارتند از **firstName** و **lastName** و توابع عضو **setFirstName**، **getFirstName**، **setLastName** و **getLastName** که یکسان با موارد مشابه در کلاس **CommssionEmployee** هستند. همچنین هر دو کلاس **CommssionEmployee** و **BasePlusCommssionEmployee** حاوی اعضای داده خصوصی **socialSecurityNumber**، **commissionRate** و **grossSales** به همراه توابع *get* و *set* بمنظور کار با این اعضا هستند. علاوه بر این سازنده **BasePlusCommssionEmpolyee** تقریباً یکسان با سازنده کلاس **BasePlusCommssionEmployee** است،بجزاینکه سازنده **BasePlusCommssionEmployee** مبادرت به تست **baseSalary** هم می‌کند. موارد دیگر در کلاس **BasePlusCommssionEmployee** عبارتند از اعضای داده خصوصی **baseSalary** و توابع عضو **setBaseSalary** و **getBaseSalary**. تابع **print** این کلاس شبیه تابع **print** موجود در کلاس **CommissionEmployee** است، بجز اینکه تابع **print** در **BasePlusCommssionEmployee** مقدار عضو داده **baseSalary** را هم چاپ می‌کند.

می‌توانیم کلاس **BasePlusCommssionEmployee** را با کپی کدها از کلاس **CommissionEmplyee** و سپس اصلاح کلاس **BasePlusCommssionEmployee** برای در برداشتن یک حقوق پایه و توابع عضو که برای کار با حقوق پایه لازم هستند، ایجاد کنیم. غالباً این روش کپی کردن، زمینه‌ساز خطا بوده و زمانبر است. بدتر از آن می‌تواند بصورت کپی‌های متعدد از کد یکسان در کل سیستم پخش شود، ایجاد و

نگهداری چنین کدی کابوس است. بهترین راه حل استفاده از توارث است که اعضای داده و توابع عضو از یک کلاس را بعنوان بخش‌های از کلاس‌های دیگر جذب می‌کند، بدون اینکه کد تکثیر شده باشد.

۱۲-٤-۳ ایجاد سلسله مراتب توارث BasePlusCommssionEmployee

در این بخش یک نسخه جدید از کلاس **BasePlusCommssionEmployee** ایجاد و تست می‌کنیم (شکل‌های ۱۰-۱۲ و ۱۱-۱۲) که از کلاس **CommissionEmployee** (شکل‌های ۴-۱۲ و ۵-۱۲) مشتق شده است. در این مثال شی **BasePlusCommssionEmployee** یک، **CommisionEmploye** است (چرا که قابلیت‌های کلاس **CommissionEmployee** ارث‌برده می‌شود)، اما کلاس **BasePlusCommssionEmployee** دارای عضو داده **baseSalary** متعلق ب‌خود است (شکل ۱۰-۱۲، خط 20).

```
1  // Fig. 12.10: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

شکل ۱۰-۱۲ | تعریف کلاس BasePlusCommssionEmployee شامل رابطه توارث از کلاس CommissionEmployee.

نماد کولن (:) در خط 12 از تعریف کلاس بر این نکته دلالت دارد که کلاس حالت ارث‌بری دارد. کلمه‌کلیدی **public** نشاندهنده نوع توارث است. بعنوان یک کلاس مشتق شده (شکل یافته با توارث **public**)، **BasePlusCommssionEmployee** تمام اعضای کلاس **CommissionEmployee** را بجز سازنده به ارث می‌برد [توجه کنید که نابود کننده‌ها به ارث برده نمی‌شوند]. از اینرو، سرویس‌های سراسری **BasePlusCommssionEmployee** شامل سازنده خود بوده (خطوط 16-15) و توابع عضو

سراسری از کلاس **CommissionEmloyee** به ارث برده می‌شوند. اگرچه نمی‌توانیم در کد منبع **BasePlusCommssionEmployee** این توابع به ارث رفته را مشاهده کنیم، با اینحال آنها بخشی از کلاس مشتق شده **BasePlusCommssionEmployee** هستند. همچنین سرویس‌های سراسری کلاس مشتق شده شامل توابع عضو **setBaseSalary**، **getBaseSalary**، **earnngs** و **print** می‌باشند (خطوط 18-22).

در شکل ۱۱-۱۲ پیاده‌سازی تابع عضو متعلق **BasePlusCommssionEmployee** نشان داده شده است. سازنده در خطوط 17-10 به معرفی گرامر مقداردهی کننده اولیه کلاس مبنا (خط 14) پرداخته است که از یک مقداردهی کننده اولیه عضو برای ارسال آرگومان‌ها به سازنده کلاس مبنا استفاده می‌کند.

برای فراخوانی سازنده کلاس مبنا به منظور مقداردهی اولیه، اعضای داده کلاس مبنا که توسط کلاس مشتق شده به ارث برده می‌شوند، C++ نیازمند یک سازنده کلاس مشتق شده است. خط 14 این وظیفه را با احضار سازنده **CommissionEmployee** با نام، ارسال پارامترهای سازنده **first**، **last**، **ssn**، **sales** و **rate** بعنوان آرگومان‌های برای مقداردهی اولیه اعضای داده کلاس **firstName**، **lastName**، **socialSecurityNumber**، **grossSales** و **CommissinRate** بکار گرفته می‌شوند، انجام می‌دهد. اگر سازنده **BasePlusCommssionEmployee** نتواند صریحاً سازنده کلاس **CommssionEmployee** را فراخوانی کند. **C++** مبادرت به احضار سازنده پیش‌فرض **CommissionEmploye** خواهد کرد، اما این کلاس فاقد چنین سازنده‌ای است و از اینرو کامپایلر یک خطا صادر می‌کند. از فصل سوم بخاطر دارید که کامپایلر یک سازنده پیش‌فرض بدون پارامتر برای هر کلاسی که بطور صریح سازنده‌ای را در نظر نگرفته است، فراخوانی می‌کند. با این وجود، **CommissionEmploye** بطور صریح دارای یک سازنده بوده و نیازی به سازنده پیش‌فرض نیست و هر عملی که بخواهد سازنده پیش‌فرض را برای این کلاس فراخوانی کند با خطای کامپایل مواجه می‌شود.

```
1  // Fig. 12.11: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
```

```
// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // derived class cannot access the base class's private data
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // derived class cannot access the base class's private data
  cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

```
C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35):
   error C2248:'CommissionEmployee::commissionRate':
   cannot access private member declared in class 'CommissionEmployee'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(37):
   see declaration of 'CommissionEmployee::commissionRate'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10):
   see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35):
   error C2248:'CommissionEmployee::grossSales':
   cannot access private member declared in class 'CommissionEmployee'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(36):
   see declaration of 'CommissionEmployee::grossSales'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10):
   see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(42):
   error C2248:'CommissionEmployee::firstName':
   cannot access private member declared in class 'CommissionEmployee'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(33):
   see declaration of 'CommissionEmployee::firstName'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10):
   see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(43):
   error C2248:'CommissionEmployee::lastName':
   cannot access private member declared in class 'CommissionEmployee'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(34):
   see declaration of 'CommissionEmployee::lastName'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10):
   see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(43):
   error C2248:'CommissionEmployee::socialSecurity-Number':
   cannot access private member declared in class 'CommissionEmployee'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(35):
   see declaration of 'CommissionEmployee::socialSecurityNumber'
   C:\cpphtp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10):
   see declaration of 'CommissionEmployee'

C:\cpphtp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(44):
```

۳۳٤فصل دوازدهم ـــــــ

استفاده از نام کلاس در خط 12 نیاز دارد تا به کامپایلر اعلان کند که کلاس مبنا موجود است. تعریف کلاس دقیقاً در **CommissionEmpolyee.h** است.

دلیل دوم این است که کامپایلر از تعریف کلاس برای تعیین سایز شی از آن کلاس استفاده می‌کند (در بخش ۸-۳ در این مورد صحبت کرده‌ایم). یک برنامه سرویس‌گیرنده که یک شی از کلاس ایجاد می‌کند بایستی تعریف کلاس را **#include** نماید تا کامپایلر بتواند به میزان مناسب برای آن شی حافظه رزرو نماید.

به هنگام استفاده از توارث، سایز یک شی از کلاس مشتق شده بستگی به اعضای داده اعلان شده در تعریف کلاس داشته و اعضای داده آنرا مستقیماً و غیرمستقیم از کلاس مبنا به ارث می‌برند. با وارد کردن تعریف کلاس در خط 10 به کامپایلر اجازه داده می‌شود تا حافظه مورد نیاز برای اعضای داده کلاس مبنا که بخشی از شی از کلاس مشتق شده می‌باشند تامین شده و از اینرو کل سایز تخصیصی شامل این موارد نیز می‌شود.

دلیل آخر برای خط 10 امکان دادن به کامپایلر برای تعیین اینکه آیا کلاس مشتق شده از اعضای به ارث برده شده کلاس مبنا بدرستی استفاده می‌کند یا خیر. برای مثال در برنامه شکل‌های ۱۰-۱۲ و ۱۱-۱۲، کامپایلر از فایل سرآیند کلاس مبنا برای تعیین اینکه اعضای داده در دسترس کلاس مشتق شده از نوع **private** در کلاس مبنا هستند یا خیر، استفاده کرده است. از آنجا که این نوع داده‌ها در دسترس کلاس مشتق شده قرار داده نمی‌شوند، کامپایلر خطا تولید می کند.

فرآیند لینک در سلسله مراتب توارث

در بخش ۹-۳، در ارتباط با فرآیند لینک در ایجاد یک برنامه کاربردی بنام **GradeBook** صحبت کردیم. در آن مثال، مشاهده کردید که شی سرویس‌گیرنده با کد شی کلاس **GradeBook** به همراه هر کلاس بکار رفته از کتابخانه استاندارد C++ لینک شد.

فرآیند لینک در برنامه‌ای که از کلاس‌های به ارث رفته استفاده می‌کند، مشابه است. فرآیند مستلزم کد شی برای تمام کلاس‌های بکار رفته در برنامه و کد شی بکار رفته چه بصورت مستقیم و غیرمستقیم از کلاس‌های مبنا در هر کلاس مشتق شده‌ای در برنامه است. فرض کنید سرویس گیرنده‌ای می‌خواهد برنامه‌ای ایجاد کند که از کلاس **BasePlusCommssionEmployee** استفاده کند که خود از کلاس **CommissionEmployee** مشتق شده است. در زمان کامپایل برنامه سرویس‌گیرنده کد شی سرویس‌گیرنده بایستی با کد شی کلاس‌های **BasePlusCommssionEmployee** و

CommissionEmployee لینک شده باشد، چرا که **BasePlusCommssionEmployee** توابع عضو را از کلاس مبنا **CommissionEmployee** ارث می‌برد. همچنین کد با کد شی هر کلاسی از کتابخانه استاندارد C++ که در کلاس **CommssionEmployee** و کلاس **BasePlusCommssionEmployee** یا کد سرویس‌گیرنده بکار رفته لینک می‌شود. در اینحالت برنامه قادر به دسترسی به پیاده‌سازی تمام توابع در برنامه خواهد بود.

۱۲-٤-٤ ایجاد سلسله مراتب توارث CommssionEmployee-BasePlusCommssionEmployee با استفاده از داده protected

برای اینکه کلاس **BasePlusCommssionEmployee** بتواند بطور مستقیم به اعضای داده **firstName**، **lastName**، **socialSecurityNumber**، **grossSales** و **commssionRate** از کلاس **CommissionEmployee** دسترسی داشته باشد، می‌توانیم این اعضا را بصورت **protected** (حفاظت شده) در کلاس مبنا اعلان کنیم. همانطوری که در بخش ۲-۱۲ توضیح داده شد، اعضای **protected** کلاس مبنا می‌توانند توسط اعضا و دوستان (friend) کلاس مبنا و اعضا و دوستان هر کلاس مشتق شده از آن کلاس مبنا در دسترس قرار گیرند.

تعریف کلاس مبنا CommissionEmployee با داده protected

کلاس **CommiesionEmployee** در برنامه شکل‌های ۱۲-۱۲ و ۱۳-۱۲ مبادرت به اعلان اعضای داده **firstName**، **lastName**، **socialSecurityNumber**، **grossSales** و **commissionRate** بصورت **protected** (شکل ۱۲-۱۲، خطوط 37-33) بجای **private** کرده است. پیاده‌سازی تابع عضو در شکل ۱۳-۱۲ همانند شکل ۵-۱۲ است.

```
1  // Fig. 12.12: CommissionEmployee.h
2  // CommissionEmployee class definition with protected data.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
```

```
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 protected:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

شکل ۱۲-۱۲ | تعریف کلاس CommssionEmployee که به داده protected اعلان شده اجازه دسترسی توسط کلاس‌های مشتق شده را می‌دهد.

```
1  // Fig. 12.13: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12 {
13    firstName = first; // should validate
14    lastName = last; // should validate
15    socialSecurityNumber = ssn; // should validate
16    setGrossSales( sales ); // validate and store gross sales
17    setCommissionRate( rate ); // validate and store commission rate
18 } // end CommissionEmployee constructor
19
20 // set first name
21 void CommissionEmployee::setFirstName( const string &first )
22 {
23    firstName = first; // should validate
24 } // end function setFirstName
25
26 // return first name
27 string CommissionEmployee::getFirstName() const
28 {
29    return firstName;
30 } // end function getFirstName
31
32 // set last name
33 void CommissionEmployee::setLastName( const string &last )
34 {
35    lastName = last; // should validate
36 } // end function setLastName
37
38 // return last name
39 string CommissionEmployee::getLastName() const
40 {
41    return lastName;
42 } // end function getLastName
43
44 // set social security number
45 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
46 {
47    socialSecurityNumber = ssn; // should validate
48 } // end function setSocialSecurityNumber
```

```
49
50 // return social security number
51 string CommissionEmployee::getSocialSecurityNumber() const
52 {
53    return socialSecurityNumber;
54 } // end function getSocialSecurityNumber
55
56 // set gross sales amount
57 void CommissionEmployee::setGrossSales( double sales )
58 {
59    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
60 } // end function setGrossSales
61
62 // return gross sales amount
63 double CommissionEmployee::getGrossSales() const
64 {
65    return grossSales;
66 } // end function getGrossSales
67
68 // set commission rate
69 void CommissionEmployee::setCommissionRate( double rate )
70 {
71    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
72 } // end function setCommissionRate
73
74 // return commission rate
75 double CommissionEmployee::getCommissionRate() const
76 {
77    return commissionRate;
78 } // end function getCommissionRate
79
80 // calculate earnings
81 double CommissionEmployee::earnings() const
82 {
83    return commissionRate * grossSales;
84 } // end function earnings
85
86 // print CommissionEmployee object
87 void CommissionEmployee::print() const
88 {
89    cout << "commission employee: " << firstName << ' ' << lastName
90       << "\nsocial security number: " << socialSecurityNumber
91       << "\ngross sales: " << grossSales
92       << "\ncommission rate: " << commissionRate;
93 } // end function print
```

شکل ۱۳-۱۲ | کلاس CommssionEmployee با داده protected.

اصلاح کلاس مشتق شده BasePlusCommssionEmployee

اکنون مبادرت به اصلاح کلاس **BasePlusCommssionEmployee** (شکل‌های ۱۴-۱۲ و ۱۵-۱۲) می‌کنیم تا بتواند از نسخه کلاس **CommisonEmplyee** در شکل‌های ۱۲-۱۲ و ۱۳-۱۲ ارث‌بری داشته باشد. بدلیل اینکه کلاس **BasePlusCommssionEmployee** از این نسخه از کلاس ارث‌بری دارد، شی‌های کلاس **BasePlusCommssionEmployee** می‌توانند به عضوهای داده به ارث رفته که بصورت **protected** در کلاس **BasePlusCommssionEmployee** اعلان شده‌اند، دسترسی پیدا کنند (یعنی اعضای **firstName**، **lastName**، **socialSecurityNumber**، **grossSales** و **commsionRate**). در

نتیجه، کامپایلر به هنگام کامپایل توابع عضو **earnings** و **print** که در شکل ۱۵-۱۲ تعریف شده‌اند، خطا تولید نخواهد کرد.

```
1  // Fig. 12.14: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

شکل ۱۴-۱۲ | فایل سرآیند کلاس BasePlusCommssionEmployee.

شکل ۱۵-۱۲ فایل پیاده‌سازی **BasePlusCommssionEmployee** است که داده **protected** از **CommsionEmployee** را به ارث می‌برد. کلاس **BasePlusCommssionEmployee** سازنده کلاس **CommsionEmplyee** را به ارث نمی‌برد. با این وجود سازنده کلاس **BasePlusCommssionEmployee** اقدام به فراخوانی صریح سازنده **CommssionEmployee** می‌کند (شکل ۱۵-۱۲، خطوط 17-10)، چرا که **CommssionEmployee** حاوی یک سازنده پیش‌فرض نیست که بتواند آنرا بصورت ضمنی (غیرصریح) را فراخوانی نماید.

```
1  // Fig. 12.15: BasePlusCommissionEmployee.cpp
2  // Class BasePlusCommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  // BasePlusCommissionEmployee class definition
7  #include "BasePlusCommissionEmployee.h"
8
9  // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // explicitly call base-class constructor
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
```

```
// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // can access protected data of base class
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // can access protected data of base class
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

شکل ۱۵-۱۲ | فایل پیاده‌سازی کلاس BasePlusCommssionEmployee.

تست کلاس اصلاح شده BasePlusCommssionEmployee

در برنامه ۱۶-۱۲ از یک شی **BasePlusCommssionEmployee** برای انجام همان وظایف که برنامه ۹-۱۲ بر روی یک شی از نسخه اول کلاس **BasePlusCommssionEmployee** انجام می‌داد (شکل‌های ۷-۱۲ و ۸-۱۲) استفاده شده است. دقت کنید که خروجی هر دو برنامه یکسان هستند. ابتدا **BasePlusCommssionEmployee** را بدون استفاده از توارث ایجاد کرده و این نسخه جدید را با استفاده از ارث‌بری ایجاد کرده‌ایم. با این همه هر دو کلاس وظایف یکسانی را انجام می‌دهند. توجه کنید که کد کلاس **BasePlusCommssionEmployee** (یعنی فایلهای سرآیند و پیاده‌سازی)، که 74 خط می‌شود. بطور قابل ملاحظه کوتاهتر از کد نسخه غیر ارث‌بر این کلاس می‌باشد که از 154 خط تشکیل شده است، چرا که نسخه ارث‌بر بخشی از قابلیت‌ها و وظایف خود را از **CommssionEmployee** به ارث برده و نسخه غیر ارث‌بر چنین خاصیتی ندارد. همچنین، در اینجا فقط یک کپی از توابع کلاس **CommissionEmployee** اعلان و تعریف شده است. در اینحالت نگهداری کد منبع، اصلاح و خطایابی آن آسانتر می‌شود، چرا که کد منبع مرتبط با **CommisionEmployee** فقط در فایلهای شکل ۱۲-۱۲ و ۱۳-۱۲ قرار دارند.

```
// Fig. 12.16: fig12_16.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
using std::cout;
```

```
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16    // instantiate BasePlusCommissionEmployee object
17    BasePlusCommissionEmployee
18       employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20    // set floating-point output formatting
21    cout << fixed << setprecision( 2 );
22
23    // get commission employee data
24    cout << "Employee information obtained by get functions: \n"
25       << "\nFirst name is " << employee.getFirstName()
26       << "\nLast name is " << employee.getLastName()
27       << "\nSocial security number is "
28       << employee.getSocialSecurityNumber()
29       << "\nGross sales is " << employee.getGrossSales()
30       << "\nCommission rate is " << employee.getCommissionRate()
31       << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33    employee.setBaseSalary( 1000 ); // set base salary
34
35    cout << "\nUpdated employee information output by print function: \n"
36       << endl;
37    employee.print(); // display the new employee information
38
39    // display the employee's earnings
40    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42    return 0;
43 } // end main
```

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Update employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

شکل ۱۶-۱۲ داده کلاس مبنای protected می‌تواند از طریق کلاس مشتق شده در دسترس قرار گیرد.

نکاتی در ارتباط با استفاده از داده protected

در این مثال، اعضای داده کلاس مبنا را بصورت **protected** اعلان کردیم، از اینروست که کلاس‌های مشتق شده قادر به اصلاح داده‌ها بصورت مستقیم هستند. ارث‌بری اعضاء داده **protected**، کمی در افزایش کارایی موثر است، چرا که می‌توانیم مستقیماً به اعضا دسترسی پیدا کنیم بدون اینکه متحمل

فراخوانی‌های اضافی توابع عضو *get* یا *set* شده باشیم. با این وجود، در بسیاری از موارد، بهتر است از اعضای داده **private** استفاده کنیم تا به لحاظ مهندسی نرم‌افزار در مسیر درستی قرار داشته باشیم و وظیفه بهینه‌سازی کد را به کامپایلر واگذار کنیم. در اینحالت نگهداری، خطایابی و اصلاح برنامه آسانتر می‌شود.

استفاده از اعضای داده **protected** دو مشکل عمده دارد. اول اینکه، شی از کلاس مشتق شده نمی‌تواند از یک تابع برای تنظیم مقدار عضو داده **protected** کلاس مبنا استفاده کند. از اینرو، یک شی از کلاس مشتق شده می‌تواند یک مقدار نامعتبر به عضو داده **protected** تخصیص دهد، از اینرو شی در وضعیت غیرپایدار باقی می‌ماند. برای مثال عضو داده **grossSales** از کلاس **CommssionEmployee** که بصورت **protected** اعلان شده است، یک شی از کلاس مشتق شده (مثلاً **BasePlusCommssionEmployee**) می‌تواند یک مقدار منفی به **grossSales** تخصیص دهد. مشکل دوم در ارتباط با استفاده از اعضای داده **protected** این است که توابع عضو از کلاس مشتق شده بستگی به پیاده‌سازی کلاس مبنا دارند. در عمل، کلاس‌های مشتق شده بایستی فقط به سرویس‌های کلاس مبنا بستگی داشته باشند (یعنی توابع عضو غیر **private**) و نه به پیاده‌سازی کلاس مبنا. در صورتی که اعضا داده در کلاس مبنا بصورت **protected** باشند و اگر پیاده‌سازی کلاس مبنا دچار تغییر شود، نیاز به اصلاح تمام کلاس‌های مشتق شده از آن کلاس مبنا خواهیم داشت. برای مثال، اگر به برخی از دلایل نیاز باشد که اسامی اعضای داده **firstName** و **lastName** را به **first** و **last** تغییر دهیم، مجبور هستیم در تمام مکان‌های که یک کلاس مشتق شده بصورت مستقیم به این اعضای کلاس مبنا مراجعه می‌کند این تغییرات را اعمال کنیم.

در چنین حالتی، گفته می‌شود که نرم‌افزار شکننده یا بی‌دوام است، چرا که یک تغییر کوچک در کلاس مبنا می‌تواند پیاده‌سازی کلاس مشتق شده را در هم ریزد.

۵-٤-۱۲ ایجاد سلسله مراتب توارث CommissionEmplyee-BasePlusCommssionEmployee با استفاده از private

اکنون باز هم به سراغ سلسله مراتب قبلی می‌رویم، اما این بار از یک روش مناسب در مهندسی نرم‌افزار استفاده خواهیم کرد. اعضای داده کلاس **CommsionEmployee** را بصورت **private** اعلان می‌کنیم (شکل ۱۷-۱۲، خطوط خطوط 37-33) و توابع عضو آنرا بصورت **public** در نظر می‌گیریم تا بتوانیم این مقادیر را نگهداری کنیم. اگر تصمیم به تغییر اسامی داده عضو بگیریم، دیگر تعاریف توابع **print** و **earnings** دچار تغییر نخواهند شد و فقط تعاریف توابع عضو *set* و *get* که مستقیماً با اعضای داده کار می‌کنند نیاز به تغییر خواهند داشت.

توجه کنید که این تغییرات منحصراً در درون کلاس مبنا صورت می‌گیرد و نیاز به اعمال هیچ تغییری در کلاس مشتق شده نخواهد بود. کلاس مشتق شده **BasePlusCommssionEmployee** (شکل‌های ۱۹-۱۲ و ۲۰-۱۲) توابع عضو غیر **private** را از کلاس **CommissionEmployee** به ارث برده و می‌تواند به اعضای **private** کلاس مبنا از طریق آن توابع دسترسی پیدا کند.

```
1  // Fig. 12.17: CommissionEmployee.h
2  // CommissionEmployee class definition with good software engineering.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

شکل ۱۷-۱۲ | تعریف کلاس CommissionEmployee به روش مناسب مهندسی نرم‌افزار.

```
1  // Fig. 12.18: CommissionEmployee.cpp
2  // Class CommissionEmployee member-function definitions.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  // constructor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14    setGrossSales( sales ); // validate and store gross sales
15    setCommissionRate( rate ); // validate and store commission rate
16 } // end CommissionEmployee constructor
```

```
17
18 // set first name
19 void CommissionEmployee::setFirstName( const string &first )
20 {
21    firstName = first; // should validate
22 } // end function setFirstName
23
24 // return first name
25 string CommissionEmployee::getFirstName() const
26 {
27    return firstName;
28 } // end function getFirstName
29
30 // set last name
31 void CommissionEmployee::setLastName( const string &last )
32 {
33    lastName = last; // should validate
34 } // end function setLastName
35
36 // return last name
37 string CommissionEmployee::getLastName() const
38 {
39    return lastName;
40 } // end function getLastName
41
42 // set social security number
43 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
44 {
45    socialSecurityNumber = ssn; // should validate
46 } // end function setSocialSecurityNumber
47
48 // return social security number
49 string CommissionEmployee::getSocialSecurityNumber() const
50 {
51    return socialSecurityNumber;
52 } // end function getSocialSecurityNumber
53
54 // set gross sales amount
55 void CommissionEmployee::setGrossSales( double sales )
56 {
57    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
58 } // end function setGrossSales
59
60 // return gross sales amount
61 double CommissionEmployee::getGrossSales() const
62 {
63    return grossSales;
64 } // end function getGrossSales
65
66 // set commission rate
67 void CommissionEmployee::setCommissionRate( double rate )
68 {
69    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
70 } // end function setCommissionRate
71
72 // return commission rate
73 double CommissionEmployee::getCommissionRate() const
74 {
75    return commissionRate;
76 } // end function getCommissionRate
77
78 // calculate earnings
79 double CommissionEmployee::earnings() const
80 {
81    return getCommissionRate() * getGrossSales();
82 } // end function earnings
83
84 // print CommissionEmployee object
85 void CommissionEmployee::print() const
86 {
```

```
87    cout << "commission employee: "
88       << getFirstName() << ' ' << getLastName()
89       << "\nsocial security number: " << getSocialSecurityNumber()
90       << "\ngross sales: " << getGrossSales()
91       << "\ncommission rate: " << getCommissionRate();
92 } // end function print
```

شکل ۱۸-۱۲ | فایل پیاده‌سازی کلاس از CommissionEmployee.

در پیاده‌سازی **CommissionEmployee** (شکل ۱۸-۱۲، خطوط 16-9) توجه کنید که از مقداردهی کنند اولیه عضو (خط 12) برای تنظیم مقادیر عضو **firstName**، **lastName** و **socialSecurityNumber** استفاده کرده‌ایم. نشان داده‌ایم که چگونه کلاس مشتق شده **BasePlusCommssionEmployee** (شکل ۱۹-۱۲ و ۲۰-۱۲) می‌تواند توابع عضو کلاس مبنا را که غیر **private** هستند را برای کار با این اعضای داده فراخوانی کند.

کلاس **BasePlusCommssionEmployee** (شکل‌های ۱۹-۱۲ و ۲۰-۱۲) چندین تغییر در پیاده‌سازی تابع عضو خود دارد (شکل ۲۰-۱۲) که آنرا از نسخه قبلی کلاس متمایز می‌سازد (شکل‌های ۱۴-۱۲ و ۱۵-۱۲).

```
1  // Fig. 12.19: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

شکل ۱۹-۱۲ | فایل سرآیند کلاس BasePlusCommssionEmployee.

```
1   // Fig. 12.20: BasePlusCommissionEmployee.cpp
2   // Class BasePlusCommissionEmployee member-function definitions.
3   #include <iostream>
4   using std::cout;
5
6   // BasePlusCommissionEmployee class definition
7   #include "BasePlusCommissionEmployee.h"
8
9   // constructor
```

```
10  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate, double salary )
13     // explicitly call base-class constructor
14     : CommissionEmployee( first, last, ssn, sales, rate )
15  {
16     setBaseSalary( salary ); // validate and store base salary
17  } // end BasePlusCommissionEmployee constructor
18
19  // set base salary
20  void BasePlusCommissionEmployee::setBaseSalary( double salary )
21  {
22     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23  } // end function setBaseSalary
24
25  // return base salary
26  double BasePlusCommissionEmployee::getBaseSalary() const
27  {
28     return baseSalary;
29  } // end function getBaseSalary
30
31  // calculate earnings
32  double BasePlusCommissionEmployee::earnings() const
33  {
34     return getBaseSalary() + CommissionEmployee::earnings();
35  } // end function earnings
36
37  // print BasePlusCommissionEmployee object
38  void BasePlusCommissionEmployee::print() const
39  {
40     cout << "base-salaried ";
41
42     // invoke CommissionEmployee's print function
43     CommissionEmployee::print();
44
45     cout << "\nbase salary: " << getBaseSalary();
46  } // end function print
```

شکل ۲۰-۱۲ | کلاس BasePlusCommssionEmployee که از کلاس CommissionEmployee ارث‌بری دارد اما نمی‌تواند مستقیماً به داده private کلاس دسترسی پیدا کند.

توابع عضو **earnings** (شکل ۲۰-۲، خطوط 35-32) و **print** (خطوط 46-38) هر یک تابع عضو **getBaseSalary** را برای بدست آوردن مقدار حقوق پایه، بجای دسترسی مستقیم به **baseSalary** را احضار می‌کنند. این روش از تغییرات **earnings** و **print** که در صورت تغییر در پیاده‌سازی عضو داده **baseSalary** رخ می‌دهد، حفاظت می‌کند. برای مثال، اگر تصمیم به تغییر نام دادن عضو داده **baseSalary** یا تغییر نوع آن بگیریم، فقط توابع عضو **setBaseSalary** و **getBaseSalary** نیاز به تغییر خواهند داشت.

تابع **earnings** (شکل ۲۰-۱۲، خطوط 35-32) تعریف مجددی از تابع عضو **earnings** از کلاس **CommissionEmployee** (شکل ۱۸-۱۲، خطوط 82-79) برای محاسبه حقوق برای کارمندی است که حقوق و کمیسیونی از فروش دریافت می‌کند. نسخه **earnings** از کلاس **BasePlusCommssionEmployee** بخشی از حقوق کارمند را بر مبنای کمیسیون را صرفاً با فراخوانی تابع **earnings** کلاس مبنا با عبارت **commissionEmployee::earnings()** بدست می‌آورد. (شکل ۲۲-

۱۲، خط 34). سپس تابع **earnings** از کلاس **BasePlusCommssionEmployee** اقدام به افزودن حقوق پایه به این مقدار می‌کند، تا کل حقوق کارمند محاسبه شود. به گرامر بکار رفته در فراخوانی یک تابع عضو کلاس مبنا که مجدداً تعریف شده از یک کلاس مشتق شده است دقت کنید. قرار دادن نام کلاس مبنا و عملگر باینری تفکیک قلمرو (::) قبل از نام تابع عضو کلاس مبنا. با داشتن تابع **earnings** کلاس **BasePlusCommssionEmployee** که تابع **earnings** از کلاس **CommissionEmployee** را برای محاسبه بخشی از حقوق شی از **BasePlusCommssionEmployee** فراخوانی می‌کنند، از تکثیر کد اجتناب شده و مشکلات نگهداری کد کاهش می‌یابد.

همین حالت برای تابع **BasePlusCommssionEmployee** (شکل ۲۰-۱۲، خطوط 46-38) که تعریف مجددی از تابع عضو **print** از کلاس **CommissionEmployee** است، صادق می‌باشد (شکل ۱۸-۱۲، خطوط 92-85). این تابع اطلاعاتی در ارتباط با کارمندی که حقوق پایه به همراه کمیسیون را دریافت می‌کند، به نمایش در می‌آورد.

برنامه شکل ۲۱-۱۲ همان کارها را بر روی یک شی **BasePlusCommssionEmployee** را همانند شکل‌های ۹-۱۲ و ۱۶-۱۲ که بر روی شی‌های از کلاس **CommissionEmployee** و **BasePlusCommssionEmployee** انجام می‌دادند، انجام می‌دهد. با استفاده از توارث و فراخوانی توابع عضو که داده‌ها در آنها پنهان است، کلاسی خواهیم داشت که بخوبی ایجاد شده و از کارایی مناسبی نیز برخوردار است.

```
1    // Fig. 12.21: fig12_21.cpp
2    // Testing class BasePlusCommissionEmployee.
3    #include <iostream>
4    using std::cout;
5    using std::endl;
6    using std::fixed;
7
8    #include <iomanip>
9    using std::setprecision;
10
11   // BasePlusCommissionEmployee class definition
12   #include "BasePlusCommissionEmployee.h"
13
14   int main()
15   {
16      // instantiate BasePlusCommissionEmployee object
17      BasePlusCommissionEmployee
18         employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20      // set floating-point output formatting
21      cout << fixed << setprecision( 2 );
22
23      // get commission employee data
24      cout << "Employee information obtained by get functions: \n"
25         << "\nFirst name is " << employee.getFirstName()
26         << "\nLast name is " << employee.getLastName()
27         << "\nSocial security number is "
28         << employee.getSocialSecurityNumber()
```

```
29         << "\nGross sales is " << employee.getGrossSales()
30         << "\nCommission rate is " << employee.getCommissionRate()
31         << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33      employee.setBaseSalary( 1000 ); // set base salary
34
35      cout << "\nUpdated employee information output by print function: \n"
36         << endl;
37      employee.print(); // display the new employee information
38
39      // display the employee's earnings
40      cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42      return 0;
43   } // end main
```

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Update employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

شکل ۲۱-۱۲ | داده private کلاس مبنا که برای یک کلاس مشتق شده از طریق تابع عضو public یا protected ارث رفته توسط کلاس مشتق شده در دسترس می باشد.

۱۲-۵ سازنده‌ها و پایان‌دهنده‌ها در کلاس‌های مشتق شده

همانطوری که در بخش‌های قبلی گفته شد، نمونه‌سازی یک شی کلاس مشتق شده با فراخوانی سازنده‌های کلاس مبنا صورت می‌گیرد و اینکار قبل از آنکه سازنده‌های مشتق شده قادر به انجام وظایف خود باشند اعمال می‌شود. فراخوانی سازنده کلاس مبنا می‌تواند بصورت صریح و غیرصریح انجام شود. بطور مشابه اگر کلاس مبنا از کلاس دیگری مشتق شده باشد، بایستی سازنده کلاس مبنا اقدام به فراخوانی سازنده کلاس بعدی در درخت سلسله مراتب نماید و اینکار تا پایان ادامه می‌یابد. آخرین سازنده فراخوانی شده در این زنجیره سازنده کلاس در بالای سلسله مراتب است که ابتدا اجرای بدنه آن خاتمه می‌یابد. هر سازنده کلاس مبنا اعضای داده کلاس مبنا را که توسط کلاس‌های مشتق شده به ارث برده شده‌اند، مقداردهی اولیه می‌کند. برای مثال، به سلسله مراتب **CommissionEmployee/BasePlusCommissionEmployee** در شکل‌های ۱۲-۱۷ الی ۱۲-۲۰ توجه نمائید. هنگامی که برنامه اقدام به ایجاد یک شی **BasePlusCommissionEmployee** می‌کند، یکی از سازنده‌های **CommissionEmployee** فراخوانی می‌شود.

از آنجایی که کلاس **CommissionEmployee** در بالای سلسله مراتب قرار دارد، سازنده آن اجرا شده، اعضای داده **private** آن که بخشی از شی **BasePlusCommssionEmployee** می‌باشند مقداردهی اولیه می‌شوند. زمانیکه اجرای سازنده **CommissionEmployee** کامل شده، کنترل را به سازنده **BasePlusCommssionEmployee** برگشت می‌دهد، که آن هم **baseSalary** را مقداردهی اولیه می‌نماید.

زمانیکه یک شی از کلاس مشتق شده نابود می‌شود، برنامه، نابود کننده آن شی را فراخوانی می‌کند. اینکار با فراخوانی زنجیره‌وار نابود کننده‌ها شروع می‌شود که در آن نابود کننده کلاس مشتق شده و نابود کننده‌های مستقیم و غیرمستقیم کلاس‌های مبنا و اعضای کلاس‌ها به ترتیب معکوس از اجرای سازنده‌ها، اجرا می‌شوند. زمانیکه نابود کننده یک شی کلاس مشتق شده فراخوانی می‌گردد، نابود کننده وظیفه خود را انجام می‌دهد، سپس نابود کننده‌ای را که در یک سطح بالاتر از سلسله مراتب قرار دارد، احضار می‌کند. این فرآیند تا فراخوانی نابود کننده قرار گرفته در بالاترین سطح سلسله مراتب ادامه می‌یابد. سپس شی از حافظه حذف می‌گردد.

سازنده‌ها، نابود کننده‌ها و عملگرهای سربارگذاری شده تخصیص کلاس مبنا توسط کلاس‌های مشتق شده، ارث‌بری نمی‌شوند. با این وجود، سازنده‌ها، نابود کننده‌ها و عملگرهای تخصیص سربارگذاری شده کلاس مشتق شده می‌توانند سازنده‌ها، نابود کننده‌ها و عملگرهای تخصیص سربارگذاری شده کلاس مبنا را فراخوانی کنند.

مثال بعدی نگاهی مجدد به سلسله مراتب کارمند کمیسیون بگیر است که توسط کلاس **CommissionEmployee** (شکل‌های۲۲-۱۲ و ۲۳-۱۲) و کلاس **BasePlusCommssionEmployee** (شکل‌های ۲۴-۱۲ و ۲۵-۱۲) تعریف شده و حاوی سازنده‌ها و نابود کننده‌های است که هر یک به هنگام فراخوانی پیغامی چاپ می‌کنند. همانطوری که در خروجی شکل ۲۶-۱۲ مشاهده می‌کنید، این پیغام‌ها ترتیب فراخوانی سازنده‌ها و نابود کننده‌ها را در سلسله مراتب توارث نشان می‌دهند.

```
// Fig. 12.22: CommissionEmployee.h
// CommissionEmployee class definition represents a commission employee.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );
   ~CommissionEmployee(); // destructor

```

```
16     void setFirstName( const string & ); // set first name
17     string getFirstName() const; // return first name
18
19     void setLastName( const string & ); // set last name
20     string getLastName() const; // return last name
21
22     void setSocialSecurityNumber( const string & ); // set SSN
23     string getSocialSecurityNumber() const; // return SSN
24
25     void setGrossSales( double ); // set gross sales amount
26     double getGrossSales() const; // return gross sales amount
27
28     void setCommissionRate( double ); // set commission rate
29     double getCommissionRate() const; // return commission rate
30
31     double earnings() const; // calculate earnings
32     void print() const; // print CommissionEmployee object
33  private:
34     string firstName;
35     string lastName;
36     string socialSecurityNumber;
37     double grossSales; // gross weekly sales
38     double commissionRate; // commission percentage
39  }; // end class CommissionEmployee
40
41  #endif
```

شکل ۲۲-۱۲ | فایل سرآیند کلاس CommissionEmployee.

در این مثال، سازنده **CommissionEmployee** را اصلاح کرده (خطوط 21-10 از شکل ۲۳-۱۲) و یک نابود کننده **CommissionEmployee** (خطوط 29-24) به آن افزوده‌ایم، که هر کدام به هنگام فراخوانی یک پیغام مناسب در خروجی قرار می‌دهند.همچنین سازنده **BasePlusCommssionEmployee** را اصلاح کرده (خطوط 22-11 از شکل ۲۵-۱۲) و یک نابود کننده به آن افزوده‌ایم (خطوط 30-25) که هر کدام به هنگام فراخوانی یک پیغام مناسب در خروجی قرار می‌دهند.

```
1   // Fig. 12.23: CommissionEmployee.cpp
2   // Class CommissionEmployee member-function definitions.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   #include "CommissionEmployee.h" // CommissionEmployee class definition
8
9   // constructor
10  CommissionEmployee::CommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate )
13     : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
14  {
15     setGrossSales( sales ); // validate and store gross sales
16     setCommissionRate( rate ); // validate and store commission rate
17
18     cout << "CommissionEmployee constructor: " << endl;
19     print();
20     cout << "\n\n";
21  } // end CommissionEmployee constructor
22
23  // destructor
24  CommissionEmployee::~CommissionEmployee()
25  {
26     cout << "CommissionEmployee destructor: " << endl;
27     print();
28     cout << "\n\n";
```

```
} // end CommissionEmployee destructor

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

// calculate earnings
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings

// print CommissionEmployee object
void CommissionEmployee::print() const
```

```
99  {
100    cout << "commission employee: "
101       << getFirstName() << ' ' << getLastName()
102       << "\nsocial security number: " << getSocialSecurityNumber()
103       << "\ngross sales: " << getGrossSales()
104       << "\ncommission rate: " << getCommissionRate();
105 } // end function print
```

شکل ۲۳-۱۲ | سازنده CommissionEmployee که متنی در خروجی قرار می‌دهد.

```
1   // Fig. 12.24: BasePlusCommissionEmployee.h
2   // BasePlusCommissionEmployee class derived from class
3   // CommissionEmployee.
4   #ifndef BASEPLUS_H
5   #define BASEPLUS_H
6
7   #include <string> // C++ standard string class
8   using std::string;
9
10  #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12  class BasePlusCommissionEmployee : public CommissionEmployee
13  {
14  public:
15     BasePlusCommissionEmployee( const string &, const string &,
16        const string &, double = 0.0, double = 0.0, double = 0.0 );
17     ~BasePlusCommissionEmployee(); // destructor
18
19     void setBaseSalary( double ); // set base salary
20     double getBaseSalary() const; // return base salary
21
22     double earnings() const; // calculate earnings
23     void print() const; // print BasePlusCommissionEmployee object
24  private:
25     double baseSalary; // base salary
26  }; // end class BasePlusCommissionEmployee
27
28  #endif
```

شکل ۲٤-۱۲ | فایل سرآیند کلاس BasePlusCommssionEmployee.

```
1   // Fig. 12.25: BasePlusCommissionEmployee.cpp
2   // Class BasePlusCommissionEmployee member-function definitions.
3   #include <iostream>
4   using std::cout;
5   using std::endl;
6
7   // BasePlusCommissionEmployee class definition
8   #include "BasePlusCommissionEmployee.h"
9
10  // constructor
11  BasePlusCommissionEmployee::BasePlusCommissionEmployee(
12     const string &first, const string &last, const string &ssn,
13     double sales, double rate, double salary )
14     // explicitly call base-class constructor
15     : CommissionEmployee( first, last, ssn, sales, rate )
16  {
17     setBaseSalary( salary ); // validate and store base salary
18
19     cout << "BasePlusCommissionEmployee constructor: " << endl;
20     print();
21     cout << "\n\n";
22  } // end BasePlusCommissionEmployee constructor
23
24  // destructor
25  BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
26  {
27     cout << "BasePlusCommissionEmployee destructor: " << endl;
28     print();
```

```
29      cout << "\n\n";
30   } // end BasePlusCommissionEmployee destructor
31
32   // set base salary
33   void BasePlusCommissionEmployee::setBaseSalary( double salary )
34   {
35      baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
36   } // end function setBaseSalary
37
38   // return base salary
39   double BasePlusCommissionEmployee::getBaseSalary() const
40   {
41      return baseSalary;
42   } // end function getBaseSalary
43
44   // calculate earnings
45   double BasePlusCommissionEmployee::earnings() const
46   {
47      return getBaseSalary() + CommissionEmployee::earnings();
48   } // end function earnings
49
50   // print BasePlusCommissionEmployee object
51   void BasePlusCommissionEmployee::print() const
52   {
53      cout << "base-salaried ";
54
55      // invoke CommissionEmployee's print function
56      CommissionEmployee::print();
57
58      cout << "\nbase salary: " << getBaseSalary();
59   } // end function print
```

شکل ۲۵-۱۲ | سازنده BasePlusCommssionEmployee که متنی در خروجی قرار می‌دهد.

برنامه شکل ۲۶-۱۲ به توصیف ترتیب فراخوانی سازنده‌ها و نابود کننده‌ها برای شی‌هایی می‌پردازد که بخشی از یک سلسله مراتب هستند. تابع **main** (خطوط 34-15) با نمونه‌سازی شی **employee1** از **CommissionEmployee** (خطوط 22-21) در یک بلوک مجزا در درون **main** شروع می‌شود (خطوط 23-20).شی بلافاصله به خارج از قلمرو خود می‌رود،از اینرو سازنده و نابود کننده **CommissionEmployee** فراخوانی می‌شوند. سپس، خطوط 27-26 شی **Employee2** از **BasePlusCommssion** را ایجاد می‌کنند. با اینکار سازنده **CommissionEmployee** برای نمایش خروجی با مقادیر ارسالی از سازنده **BasePlusCommssionEmployee** فراخوانی شده، سپس خروجی تعیین شده در سازنده **BasePlusCommssionEmployee** به کار می‌افتد. سپس خطوط 31-30 مبادرت به ایجاد شی **employee3** از **BasePlusCommssionEmployee** می‌کنند. مجدداً هر دو سازنده **CommissionEmployee** و **BasePlusCommssionEmployee** فراخوانی می‌شوند. توجه کنید که در هر دو مورد، بدنه سازنده **CommissionEmployee** قبل از بدنه سازنده **BasePlusCommssionEmployee** اجرا می‌شود. زمانیکه به انتهای **main** می‌رسیم، نابود کننده‌ها برای شی‌های **employee2** و **employee3** فراخوانی می‌شوند.

۳۵٤فصل دوازدهم ـــــــ

```
CommissionEmployee constructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

BasePluCommissionEmployee constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00

BasePluCommissionEmployee destructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00

CommissionEmployee destructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

BasePluCommissionEmployee destructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00

CommissionEmployee destructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
```

شکل ۲۶-۱۲ | ترتیب فراخوانی سازنده و نابود کننده.

۶-۱۲ توارث public، proteted و private

زمانیکه کلاسی از یک کلاس مبنا، مشتق می‌شود، کلاس مبنا می‌تواند از طریق ارث‌بری **public**، **protected** و **private** به ارث برود. استفاده از روش ارث‌بری **protected** و **private** بندرت اتفاق می‌افتد و در استفاده از آنها باید دقت کرد. در این کتاب ما از روش توارث **public** استفاده می‌کنیم. جدول شکل ۲۷-۱۲ هر یک از انواع توارث و میزان ارث‌بری از طریق یک کلاس مشتق شده را بطور خلاصه عرضه کرده است. ستون اول حاوی تصریح کننده‌های دسترسی کلاس مبنا است.

به هنگام مشتق کردن یک کلاس از یک کلاس مبنای **public**، اعضای **public** کلاس مبنا، تبدیل به اعضای **public** کلاس مشتق شده گردیده و اعضای **protected** از کلاس مبنا، تبدیل به اعضای **protected** کلاس مشتق شده می‌شوند. اعضای **private** کلاس مبنا هرگز بطور مستقیم از طریق یک کلاس مشتق شده در دسترس نمی‌باشند، اما می‌توان آنها را از طریق فراخوانی اعضای **public** و **protected** کلاس مبنا دسترسی پیدا کرد.

به هنگام مشتق کردن یک کلاس از یک کلاس مبنای **protected**، اعضای **public** و **protected** کلاس مبنا، تبدیل به اعضای **protected** کلاس مشتق شده می‌شوند. به هنگام مشتق کردن یک کلاس از یک کلاس مبنای **private**، اعضای **public** و **protected** از کلاس مبنا، تبدیل به اعضای **private** کلاس مشتق می‌شوند (یعنی توابع تبدیل به توابع یوتیلیتی می‌شوند) توارث **private** و **protected** یک رابط is-a (است–یک) نیست.

	نوع توارث		
تصریح‌کننده دسترسی عضو کلاس مبنا	توارث **public**	توارث **protected**	توارث **private**
public	**public** در کلاس مشتق شده. می‌تواند مستقیماً توسط توابع عضو، توابع **friend** و توابع غیر عضو در دسترس قرار گیرد	**protected** در کلاس مشتق شده. می‌تواند مستقیماً توسط توابع عضو و توابع **friend** در دسترس قرار گیرد.	**private** در کلاس مشتق شده می‌تواند مستقیماً توسط توابع عضو و توابع **friend** در دسترس قرار گیرد.
Protected	**protected** در کلاس مشتق شده. می‌تواند مستقیماً توسط عضو و توابع **friend** در دسترس قرار گیرد.	**protected** در کلاس مشتق شده. می‌تواند مستقیماً توسط توابع عضو و توابع **friend** در دسترس قرار گیرد.	**private** در کلاس مشتق شدند می‌تواند مستقیماً توسط توابع عضو و توابع **firend** در دسترس قرار گیرد.
private	پنهان در کلاس مشتق شده. می‌تواند توسط توابع عضو و توابع **friend** از طریق توابع عضو **public** یا **protected** کلاس مبنا در دسترس قرار گیرد.	پنهان در کلاس مشتق شده. می تواند توسط توابع عضو و توابع **friend** از طریق توابع عضو **public** یا **protected** کلاس مبنا در دسترس قرار گیرد.	پنهان در کلاس مشتق شده. می‌تواند توسط توابع عضو و توابع **friend** از طریق توابع عضو **public** یا **protected** کلاس مبنا در دسترس قرار گیرد.

شکل ۲۷-۱۲ | خلاصه‌ای از دسترسی به اعضای کلاس مبنا در یک کلاس مشتق شده.

۷-۱۲ مهندسی نرم‌افزار بکمک توارث

در این بخش به نقش بکارگیری توارث در بهینه‌سازی نرم‌افزار موجود می‌پردازیم. هنگامی که از توارث برای ایجاد یک کلاس از روی یک کلاس موجود استفاده می‌کنیم، کلاس جدید تعدادی از اعضای داده، توابع عضو کلاس موجود را به ارث می‌برد، همانطوری که در جدول شکل ۲۷-۱۲ توضیح داده شده

است. زمانیکه کلاس ایجاد شد، می‌توانیم با توجه به نیازهای خود اقدام به افزودن اعضا برای بهینه‌سازی کلاس جدید کنیم.

گاهی اوقات، درک مشکلاتی که طراحان مشغول بکار در پروژه‌های بزرگ و صنایع با آنها مواجه هستند، برای دانشجویان سخت است. اشخاصی که تجربه کار در چنین پروژه‌های را دارند معتقد هستند که بکارگیری مجدد نرم‌افزار می‌تواند نقش بسیار موثری در فرآیند توسعه نرم‌افزار داشته باشد. برنامه‌نویسی شی‌گرا امر بکارگیری مجدد نرم‌افزار را تسهیل بخشیده و از اینرو زمان توسعه کاهش می‌یابد.