4.3 Seudocódigo

El **seudocódigo** es un lenguaje informal que le ayuda a desarrollar algoritmos sin tener que preocuparse por los estrictos detalles de la sintaxis del lenguaje Java. El seudocódigo que presentaremos es especialmente útil para desarrollar algoritmos que se convertirán en porciones estructuradas de programas en Java. El seudocódigo que usamos en este libro es similar al lenguaje cotidiano: es conveniente y amigable con el usuario, aunque en realidad no es un lenguaje de programación de computadoras. En la figura 4.7 verá un algoritmo escrito en seudocódigo. Desde luego que puede usar sus propios idiomas nativos para desarrollar su propio seudocódigo.

El seudocódigo no se ejecuta en las computadoras. En vez de ello, le ayuda a "organizar" un programa antes de que intente escribirlo en un lenguaje de programación como Java. Este capítulo presenta varios ejemplos de cómo utilizar el seudocódigo para desarrollar programas en Java.

El estilo de seudocódigo que presentaremos consiste sólo en caracteres, para que usted pueda escribirlo de una manera conveniente, utilizando cualquier programa editor de texto. Un seudocódigo preparado cuidadosamente puede convertirse sin problema en su correspondiente programa en Java.

Por lo general, el seudocódigo describe sólo las instrucciones que representan las *acciones* que ocurren después de convertir un programa de seudocódigo a Java, y de ejecutarlo en una computadora. Dichas acciones podrían incluir la *entrada, salida o cálculos*. Por lo general *no* incluimos las declaraciones de variables en nuestro seudocódigo, pero algunos programadores optan por listar las variables y mencionar sus propósitos.

4.4 Estructuras de control

Es común en un programa que las instrucciones se ejecuten una después de otra, en el orden en que están escritas. Este proceso se conoce como **ejecución secuencial**. Varias instrucciones en Java, que pronto veremos, permiten al programador especificar que la siguiente instrucción a ejecutarse tal vez *no* sea la *siguiente* en la secuencia. Esto se conoce como **transferencia de control**.

Durante la década de 1960 se descubrió que el uso indiscriminado de las transferencias de control era el origen de muchas de las dificultades que experimentaban los grupos de desarrollo de software. A quien se señaló como culpable fue a la **instrucción `goto`** (utilizada en la mayoría de los lenguajes de programación de esa época), la cual permite al programador especificar la transferencia de control a uno de los muchos posibles destinos dentro de un programa. [*Nota*: Java *no* tiene una instrucción `goto`; sin embargo, la palabra `goto` está *reservada* para Java y *no* debe usarse como identificador en los programas].

Las investigaciones de Bohm y Jacopini[1] demostraron que los programas podían escribirse *sin* instrucciones `goto`. El reto de la época para los programadores fue cambiar sus estilos a una "programación sin `goto`". El término **programación estructurada** se hizo casi un sinónimo de la "eliminación del `goto`". No fue sino hasta la década de 1970 cuando los programadores comenzaron a tomar en serio la programación estructurada. Los resultados fueron impresionantes. Los grupos de desarrollo de software reportaron reducciones en los tiempos de desarrollo, mayor incidencia de entregas a tiempo de sistemas y más proyectos de software finalizados sin salirse del presupuesto. La clave para estos logros fue que los programas estructurados eran más claros, más fáciles de depurar y de modificar, y había más probabilidad de que estuvieran libres de errores desde el principio.

El trabajo de Bohm y Jacopini demostró que todos los programas podían escribirse en términos de tres estructuras de control solamente: la **estructura de secuencia**, la **estructura de selección** y la **estructura de repetición**. Cuando presentemos las implementaciones de las estructuras de control en

1 C. Bohm y G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," *Communications of the ACM*, Vol. 9, No. 5, mayo de 1966, páginas 336-371.

Java, nos referiremos a ellas en la terminología de la *Especificación del lenguaje Java* como "instrucciones de control".

Estructura de secuencia en Java

La estructura de secuencia está integrada en Java. A menos que se le indique lo contrario, la computadora ejecuta las instrucciones de Java una después de otra, en el orden en que estén escritas; es decir, en secuencia. El **diagrama de actividad** de la figura 4.1 ilustra una estructura de secuencia típica, en la que se realizan dos cálculos en orden. Java permite tantas acciones como queramos en una estructura de secuencia. Como veremos pronto, en donde quiera que se coloque una sola acción, podrán colocarse varias acciones en secuencia.

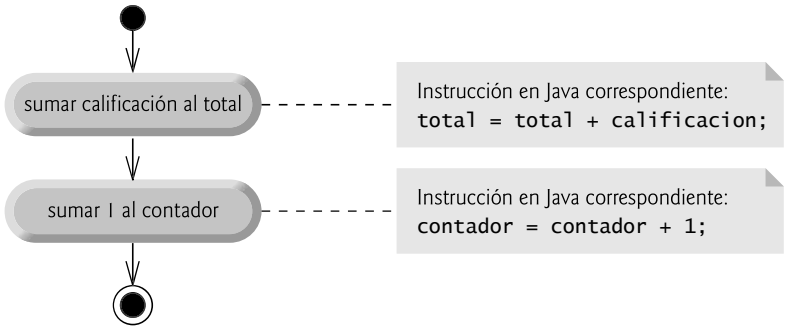

Fig. 4.1 | Diagrama de actividad de una estructura de secuencia.

Un diagrama de actividad de UML modela el **flujo de trabajo** (también conocido como la **actividad**) de una parte de un sistema de software. Dichos flujos de trabajo pueden incluir una porción de un algoritmo, como la estructura de secuencia de la figura 4.1. Los diagramas de actividad están compuestos por símbolos, como los **símbolos de estado de acción** (rectángulos cuyos lados izquierdo y derecho se reemplazan con arcos hacia fuera), **rombos** y **círculos pequeños**. Estos símbolos se conectan mediante **flechas de transición**, que representan el *flujo de la actividad*; es decir, el *orden* en el que deben ocurrir las acciones.

Al igual que el seudocódigo, los diagramas de actividad ayudan a los programadores a desarrollar y representar algoritmos. Los diagramas de actividad muestran con claridad cómo operan las estructuras de control. En este capítulo y en el 5 usaremos el UML para mostrar el flujo de control en las instrucciones de control. En los capítulos 33 y 34 en línea usaremos el UML en un ejemplo práctico verdadero de un cajero automático real.

Considere el diagrama de actividad para la estructura de secuencia de la figura 4.1. Este diagrama contiene dos **estados de acción**, cada uno de los cuales contiene una **expresión de acción** (por ejemplo, "sumar calificación a total" o "sumar 1 al contador"), que especifica una acción particular a realizar. Otras acciones podrían incluir cálculos u operaciones de entrada/salida. Las flechas en el diagrama de actividad representan **transiciones**, las cuales indican el *orden* en el que ocurren las acciones representadas por los estados de acción. El programa que implementa las actividades ilustradas por el diagrama de la figura 4.1 primero suma `calificacion` a `total`, y después suma `1` a `contador`.

El **círculo relleno** que se encuentra en la parte superior del diagrama de actividad representa el **estado inicial**; es decir, el *inicio* del flujo de trabajo *antes* de que el programa realice las actividades modeladas. El **círculo sólido rodeado por una circunferencia** que aparece en la parte inferior del diagrama representa el **estado final**; es decir, el *final* del flujo de trabajo *después* de que el programa realiza sus acciones.

La figura 4.1 también incluye rectángulos que tienen la esquina superior derecha doblada. En UML, a estos rectángulos se les llama **notas** (como los comentarios en Java), y son comentarios con explicaciones

que describen el propósito de los símbolos en el diagrama. La figura 4.1 utiliza las notas de UML para mostrar el código en Java asociado con cada estado de acción. Una **línea punteada** conecta cada nota con el elemento que ésta describe. Los diagramas de actividad generalmente *no* muestran el código en Java que implementa la actividad. En este libro utilizamos las notas para mostrar cómo se relaciona el diagrama con el código en Java. Para obtener más información sobre UML, vea nuestro ejemplo práctico opcional de diseño orientado a objetos (capítulos 33 y 34) o visite `www.uml.org`.

Instrucciones de selección en Java

Java tiene tres tipos de **instrucciones de selección** (las cuales se describen en este capítulo y en el 5). La *instrucción* `if` realiza (selecciona) una acción si la condición es *verdadera*, o evita la acción si la condición es *falsa*. La *instrucción* `if...else` realiza una acción si la condición es *verdadera*, o realiza una acción distinta si la condición es *falsa*. La *instrucción* `switch` (capítulo 5) realiza una de entre *varias* acciones distintas, dependiendo del valor de una expresión.

La instrucción `if` es una **instrucción de selección simple**, ya que selecciona o ignora una *sola* acción (o, como pronto veremos, un *solo grupo de acciones*). La instrucción `if...else` se conoce como **instrucción de selección doble**, ya que selecciona entre *dos acciones distintas* (o *dos grupos de acciones*). La instrucción `switch` es una **estructura de selección múltiple**, ya que selecciona entre *diversas acciones distintas* (o *grupos de acciones*).

Instrucciones de repetición en Java

Java cuenta con tres **instrucciones de repetición** (también llamadas **instrucciones de iteración** o **instrucciones de ciclo**) que permiten a los programas ejecutar instrucciones en forma repetida, siempre y cuando una condición (llamada la **condición de continuación del ciclo**) siga siendo *verdadera*. Las instrucciones de repetición son `while`, `do...while`, `for` y las instrucciones `for` mejoradas (en el capítulo 5 presentamos las instrucciones `do...while` y `for`, y en el capítulo 7 presentaremos la instrucción `for` mejorada). Las instrucciones `while` y `for` realizan la acción (o grupo de acciones) en sus cuerpos, cero o más veces; si en un principio la condición de continuación del ciclo es *falsa*, *no* se ejecutará la acción (o grupo de acciones). La instrucción `do...while` realiza la acción (o grupo de acciones) en su cuerpo, *una o más* veces. Las palabras `if`, `else`, `switch`, `while`, `do` y `for` son palabras clave en Java. En el apéndice C aparece una lista completa de las palabras clave en Java.

Resumen de las instrucciones de control en Java

Java sólo tiene tres tipos de estructuras de control, a las cuales nos referiremos de aquí en adelante como *instrucciones de control*: la *instrucción de secuencia*, las *instrucciones de selección* (tres tipos) y las *instrucciones de repetición* (tres tipos). Cada programa se forma combinando tantas de estas instrucciones como sea apropiado para el algoritmo que implemente el programa. Podemos modelar cada una de las instrucciones de control como un diagrama de actividad. Al igual que la figura 4.1, cada diagrama contiene un estado inicial y un final, los cuales representan el punto de entrada y salida de la instrucción de control, respectivamente. Las **instrucciones de control de una sola entrada/una sola salida** facilitan la creación de programas; sólo tenemos que conectar el punto de salida de una al punto de entrada de la siguiente. A esto le llamamos **apilamiento de instrucciones de control**. Más adelante aprenderemos que sólo hay una manera alternativa de conectar las instrucciones de control: el **anidamiento de instrucciones de control**, en el cual una instrucción de control aparece *dentro* de otra. Por lo tanto, los algoritmos en los programas en Java se crean a partir de sólo tres principales tipos de instrucciones de control, que se combinan sólo de dos formas. Ésta es la esencia de la simpleza.

4.5 Instrucción `if` de selección simple

Los programas utilizan instrucciones de selección para elegir entre los cursos alternativos de acción. Por ejemplo, suponga que la calificación para aprobar un examen es 60. La instrucción en *seudocódigo*

```
Si la calificación del estudiante es mayor o igual que 60
    Imprimir "Aprobado"
```

determina si la *condición* "la calificación del estudiante es mayor o igual que 60" es *verdadera*. En caso de que sea así se imprime "Aprobado", y se "ejecuta" en orden la siguiente instrucción en seudocódigo (recuerde que el seudocódigo no es un verdadero lenguaje de programación). Si la condición es *falsa* se ignora la instrucción *Imprimir*, y se ejecuta en orden la siguiente instrucción en seudocódigo. La sangría de la segunda línea de esta instrucción de selección es opcional, pero se recomienda colocarla ya que enfatiza la estructura inherente de los programas estructurados.

La instrucción anterior *Si* en seudocódigo puede escribirse en Java de la siguiente manera:

```
if (calificacionEstudiante >= 60)
   System.out.println("Aprobado");
```

El código en Java corresponde en gran medida con el seudocódigo. Ésta es una de las propiedades que hace del seudocódigo una herramienta de desarrollo de programas tan útil.

Diagrama de actividad en UML para una instrucción `if`

La figura 4.2 muestra la instrucción `if` de selección simple. Esta figura contiene lo que quizá sea el símbolo más importante en un diagrama de actividad: el *rombo* o **símbolo de decisión**, el cual indica que se tomará una *decisión*. El flujo de trabajo continúa a lo largo de una ruta determinada por las **condiciones de guardia** asociadas de ese símbolo, que pueden ser *verdaderas* o *falsas*. Cada flecha de transición que sale de un símbolo de decisión tiene una condición de guardia (especificada entre *corchetes*, a un lado de la flecha de transición). Si una condición de guardia es *verdadera*, el flujo de trabajo entra al estado de acción al que apunta la flecha de transición. En la figura 4.2, si la calificación es mayor o igual que 60, el programa imprime "Aprobado" y luego se dirige al estado final de esta actividad. Si la calificación es menor que 60, el programa se dirige de inmediato al estado final sin mostrar ningún mensaje.

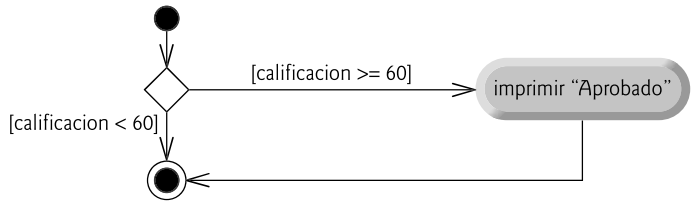

Fig. 4.2 | Diagrama de actividad en UML de la instrucción `if` de selección simple.

La instrucción `if` es una instrucción de control de una sola entrada/una sola salida. Pronto veremos que los diagramas de actividad para las instrucciones de control restantes también contienen estados iniciales, flechas de transición, estados de acción que indican las acciones a realizar, símbolos de decisión (con sus condiciones de guardia asociadas) que indican las decisiones a tomar, y estados finales.

4.6 Instrucción `if...else` de selección doble

La instrucción `if` de selección simple realiza una acción indicada sólo cuando la condición es *verdadera* (`true`); de no ser así, se evita dicha acción. La instrucción **`if...else` de selección doble** le permite especificar una acción a realizar cuando la condición es *verdadera*, y otra distinta cuando la condición es *falsa*. Por ejemplo, la instrucción en seudocódigo:

Si la calificación del estudiante es mayor o igual que 60
    Imprimir "Aprobado"
De lo contrario
    Imprimir "Reprobado"

imprime "Aprobado" si la calificación del estudiante es mayor o igual que 60, y "Reprobado" si la calificación del estudiante es menor que 60. En cualquier caso, después de la impresión se "ejecuta" la siguiente instrucción en seudocódigo en la secuencia.

La instrucción anterior *si…entonces* en seudocódigo puede escribirse en Java como

```
if (calificacion >= 60)
   System.out.println("Aprobado");
else
   System.out.println("Reprobado");
```

El cuerpo de la instrucción `else` también tiene sangría. Cualquiera que sea la convención de sangría que usted elija, debe aplicarla de manera consistente en todos sus programas.

Buena práctica de programación 4.1

Utilice sangría en ambos cuerpos de instrucciones de una estructura `if…else`. Muchos IDE hacen esto por usted.

Buena práctica de programación 4.2

Si hay varios niveles de sangría, en cada uno debe aplicarse la misma cantidad de espacio adicional.

Diagrama de actividad en UML para una instrucción `if…else`

La figura 4.3 muestra el flujo de control en la instrucción `if...else`. Una vez más (además del estado inicial, las flechas de transición y el estado final), los símbolos en el diagrama de actividad de UML representan estados de acción y decisiones.

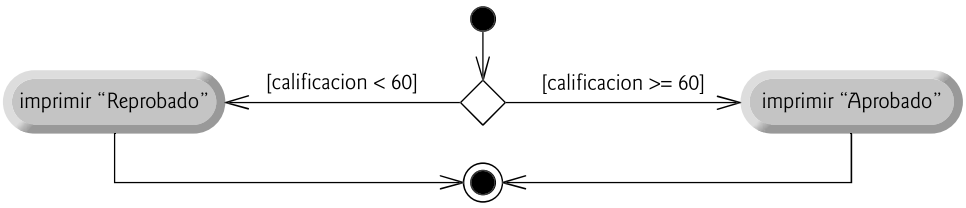

Fig. 4.3 | Diagrama de actividad de UML de la instrucción `if...else` de selección doble.

Instrucciones `if...else` anidadas

Un programa puede evaluar varios casos colocando instrucciones `if...else` dentro de otras instrucciones `if...else` para crear **instrucciones `if...else` anidadas**. Por ejemplo, el siguiente seudocódigo representa una instrucción `if...else` anidada que imprime `A` para las calificaciones de exámenes mayores o iguales a 90, `B` para las calificaciones en el rango de 80 a 89, `C` para las calificaciones en el rango de 70 a 79, `D` para las calificaciones en el rango de 60 a 69 y `F` para todas las demás calificaciones:

```
Si la calificación del estudiante es mayor o igual que 90
    Imprimir "A"
de lo contrario
    Si la calificación del estudiante es mayor o igual que 80
        Imprimir "B"
    de lo contrario
        Si la calificación del estudiante es mayor o igual que 70
            Imprimir "C"
        de lo contrario
            Si la calificación del estudiante es mayor o igual que 60
                Imprimir "D"
            de lo contrario
                Imprimir "F"
```

Este seudocódigo puede escribirse en Java como

```
if (calificacionEstudiante >= 90)
   System.out.println("A");
else
   if (calificacionEstudiante >= 80)
      System.out.println("B");
   else
      if (calificacionEstudiante >= 70)
         System.out.println("C");
      else
         if (calificacionEstudiante >= 60)
            System.out.println("D");
         else
            System.out.println("F");
```

Tip para prevenir errores 4.1

En una instrucción `if...else` *anidada, debe asegurarse de evaluar todos los posibles casos.*

Si la variable `calificacionEstudiante` es mayor o igual a 90, las primeras cuatro condiciones en la instrucción `if...else` anidada serán verdaderas, pero sólo se ejecutará la instrucción en la parte `if` de la primera instrucción `if...else`. Después de que se ejecute esa instrucción, se evita la parte `else` de la instrucción `if...else` más "externa". La mayoría de los programadores en Java prefieren escribir la instrucción `if...else` anterior así:

```
if (calificacionEstudiante>= 90)
   System.out.println("A");
else if (calificacionEstudiante>= 80)
   System.out.println("B");
else if (calificacionEstudiante >= 70)
   System.out.println("C");
else if (calificacionEstudiante>= 60)
   System.out.println("D");
else
   System.out.println("F");
```

Las dos formas son idénticas, excepto por el espaciado y la sangría, que el compilador ignora. La segunda forma es más popular ya que evita usar mucha sangría hacia la derecha en el código. Dicha sangría a menudo deja poco espacio en una línea de código, forzando a que las líneas se separen.

Problema del `else` suelto

El compilador de Java siempre asocia un `else` con el `if` que le precede inmediatamente, a menos que se le indique otra cosa mediante la colocación de llaves (`{` y `}`). Este comportamiento puede ocasionar lo que se conoce como el **problema del `else` suelto**. Por ejemplo,

```
if (x > 5)
   if (y > 5)
      System.out.println("x y y son > 5");
else
   System.out.println("x es <= 5");
```

parece indicar que si `x` es mayor que `5`, la instrucción `if` anidada determina si `y` es también mayor que `5`. De ser así, se produce como resultado la cadena "`x y y son > 5`". De lo contrario, parece ser que si `x` no es mayor que `5`, la instrucción `else` que es parte del `if...else` produce como resultado la cadena "`x es <= 5`". ¡Cuidado! Esta instrucción `if...else` anidada *no* se ejecuta como parece ser. El compilador en realidad interpreta la instrucción así:

```
if (x > 5)
   if (y > 5)
      System.out.println("x y y son > 5");
   else
      System.out.println("x es <= 5");
```

en donde el cuerpo del primer `if` es un `if...else` *anidado*. La instrucción `if` más externa evalúa si `x` es mayor que `5`. De ser así, la ejecución continúa evaluando si `y` es también mayor que `5`. Si la segunda condición es *verdadera*, se muestra la cadena apropiada ("`x y y son > 5`"). No obstante, si la segunda condición es *falsa* se muestra la cadena "`x es <= 5`", aun cuando sabemos que `x` es mayor que `5`. Peor aún, si la condición de la instrucción `if` exterior es falsa, se omite la instrucción `if...else` interior y no se muestra nada en pantalla.

Para forzar a que la instrucción `if...else` anidada se ejecute como se tenía pensado originalmente, debe escribirse de la siguiente manera:

```
if (x > 5)
{
   if (y > 5)
      System.out.println("x y y son > 5");
}
else
   System.out.println("x es <= 5");
```

Las llaves indican que el segundo `if` se encuentra en el cuerpo del primero y que el `else` está asociado con el *primer* `if`. Los ejercicios 4.27 y 4.28 analizan con más detalle el problema del `else` suelto.

Bloques

Por lo general, la instrucción `if` espera sólo *una* instrucción en su cuerpo. Para incluir *varias* instrucciones en el cuerpo de un `if` (o en el cuerpo del `else` en una instrucción `if...else`), encierre las instrucciones entre llaves. Las instrucciones contenidas dentro de un par de llaves (como el cuerpo de un

método) forman un **bloque**. Un bloque puede colocarse en cualquier parte de un método en donde pueda colocarse una sola instrucción.

El siguiente ejemplo incluye un bloque en la parte `else` de una instrucción `if...else`:

```
if (calificacion >= 60)
   System.out.println("Aprobado");
else
{
   System.out.println("Reprobado");
   System.out.println("Debe tomar este curso otra vez.");
}
```

En este caso, si `calificacion` es menor que 60, el programa ejecuta *ambas* instrucciones en el cuerpo del `else` e imprime

```
Reprobado.
Debe tomar este curso otra vez.
```

Observe las llaves que rodean a las dos instrucciones en la cláusula `else`. Estas llaves son importantes. Sin ellas, la instrucción

```
System.out.println("Debe tomar este curso otra vez.");
```

estaría fuera del cuerpo de la parte `else` de la instrucción `if...else` y se ejecutaría *sin importar* que la calificación fuera menor a 60.

Los *errores de sintaxis* (como cuando se omite una llave en un bloque del programa) los detecta el compilador. Un **error lógico** (como cuando se omiten ambas llaves en un bloque del programa) tiene su efecto en tiempo de ejecución. Un **error lógico fatal** hace que un programa falle y termine antes de tiempo. Un **error lógico no fatal** permite que un programa siga ejecutándose, pero éste produce resultados incorrectos.

Así como un bloque puede colocarse en cualquier parte donde pueda colocarse una instrucción individual, también es posible no tener instrucción alguna. En la sección 2.8 vimos que la instrucción vacía se representa colocando un punto y coma (`;`) en donde normalmente iría una instrucción.

Error común de programación 4.1

Colocar un punto y coma después de la condición en una instrucción `if` o `if...else` produce un error lógico en las instrucciones `if` de selección simple, y un error de sintaxis en las instrucciones `if...else` de selección doble (cuando la parte del `if` contiene una instrucción en el cuerpo).

Operador condicional (`?:`)

Java cuenta con el **operador condicional** (`?:`), que en ocasiones puede utilizarse en lugar de una instrucción `if...else`. Este operador puede hacer su código más corto y claro. Es el único **operador ternario** en Java (un operador que utiliza *tres* operandos). En conjunto, los operandos y el símbolo `?:` forman una **expresión condicional**. El primer operando (a la izquierda del `?`) es una **expresión booleana** (`boolean`) (es decir, una *condición* que se evalúa a un valor `boolean`: `true` o `false`), el segundo operando (entre el `?` y `:`) es el valor de la expresión condicional si la expresión booleana es *verdadera* (`true`), y el tercer operando (a la derecha del `:`) es el valor de la expresión condicional si la expresión booleana se evalúa como `false`. Por ejemplo, la instrucción

```
System.out.println(calificacionEstudiante >= 60 ? "Aprobado" : "Reprobado");
```

imprime el valor del argumento de la expresión condicional de `println`. La expresión condicional en esta instrucción produce como resultado la cadena "`Aprobado`" si la expresión booleana `calificacionEstudiante >= 60` es verdadera, y la cadena "`Reprobado`" si la expresión booleana es falsa. Por lo tanto, esta instrucción con el operador condicional realiza en esencia la misma función que la instrucción `if...else` que se mostró anteriormente en esta sección. Puesto que la precedencia del operador condicional es baja, es común que toda la expresión condicional se coloque entre paréntesis. Pronto veremos que las expresiones condicionales pueden usarse en algunas situaciones en las que no se pueden utilizar instrucciones `if...else`.

Tip para prevenir errores 4.2

Para evitar algunos errores sutiles, use expresiones del mismo tipo para el segundo y tercer operandos del operador `?:`.

4.7 Clase `Estudiante`: instrucciones `if...else` anidadas

El ejemplo de las figuras 4.4 y 4.5 demuestra una instrucción `if...else` anidada que determina la calificación en letras de un estudiante con base en el promedio del estudiante en un curso.

La clase `Estudiante`

La clase `Estudiante` (figura 4.4) tiene características similares a las de la clase `Cuenta` (que vimos en el capítulo 3). La clase `Estudiante` almacena el nombre y promedio de un estudiante, además de proporcionar métodos para manipular esos valores. La clase contiene:

- la variable de instancia `nombre` de tipo `String` (línea 5) para almacenar el nombre de un `Estudiante`
- la variable de instancia `promedio` de tipo `double` (línea 6) para almacenar el promedio de un `Estudiante` en un curso
- un constructor (líneas 9 a 18) que inicializa el `nombre` y el `promedio`; en la sección 5.9 aprenderá a expresar las líneas 15-16 y 37-38 de manera más concisa con operadores lógicos que puedan evaluar múltiples condiciones
- los métodos `establecerNombre` y `obtenerNombre` (líneas 21 a 30) para *establecer* y *obtener* el `nombre` del `Estudiante`
- los métodos `establecerPromedio` y `obtenerPromedio` (líneas 33 a 46) para *establecer* y *obtener* el `promedio` del `Estudiante`
- el método `obtenerCalificacionEstudiante` (líneas 49 a 65) que usa *instrucciones* `if...else` *anidadas* para determinar la *calificación en letras* del `Estudiante`, con base en su promedio.

El constructor y el método `establecerPromedio` usan *instrucciones* `if` *anidadas* (líneas 15 a 17 y 37 a 39) para *validar* el valor que se utiliza para establecer el `promedio`. Estas instrucciones aseguran que el valor sea mayor que `0.0` *y* menor o igual a `100.0`; de lo contrario, el valor de `promedio` permanece *sin cambios*. Cada instrucción `if` contiene una condición *simple*. Si la condición en la línea 15 es *verdadera*, sólo entonces se evaluará la condición en la línea 16 y *sólo* si las condiciones en las líneas 15 *y* 16 son *verdaderas* se ejecutará la instrucción en la línea 17.

Observación de ingeniería de software 4.1

En el capítulo 3 vimos que no debe hacer llamadas a métodos desde los constructores (en el capítulo 10, Programación orientada a objetos: polimorfismo e interfaces le explicaremos por qué). Por esta razón, hay código de validación duplicado en las líneas 15 a 17 y 37 a 39 de la figura 4.4 y en los ejemplos subsiguientes.