

5.10 Resumo de programação estruturada

Da mesma forma como os arquitetos projetam edifícios empregando o conhecimento coletivo de sua profissão, assim também os programadores devem projetar programas. Nossso campo é muito mais jovem que a arquitetura e nossa sabedoria coletiva é consideravelmente mais esparsa. Aprendemos que a programação estruturada produz programas que são mais fáceis de entender, testar, depurar, modificar e até demonstrar como corretos em um sentido matemático do que os programas não estruturados.

Instruções de controle Java são de entrada única/saída única

A Figura 5.21 utiliza diagramas de atividade UML para resumir instruções de controle do Java. Os estados iniciais e finais indicam o *único ponto de entrada* e o *único ponto de saída* de cada instrução de controle. Conectar símbolos individuais arbitrariamente em um diagrama de atividade pode levar a programas não estruturados. Portanto, escolheu-se um conjunto limitado de instruções de controle que só pode ser combinado de duas maneiras simples para criar programas estruturados.

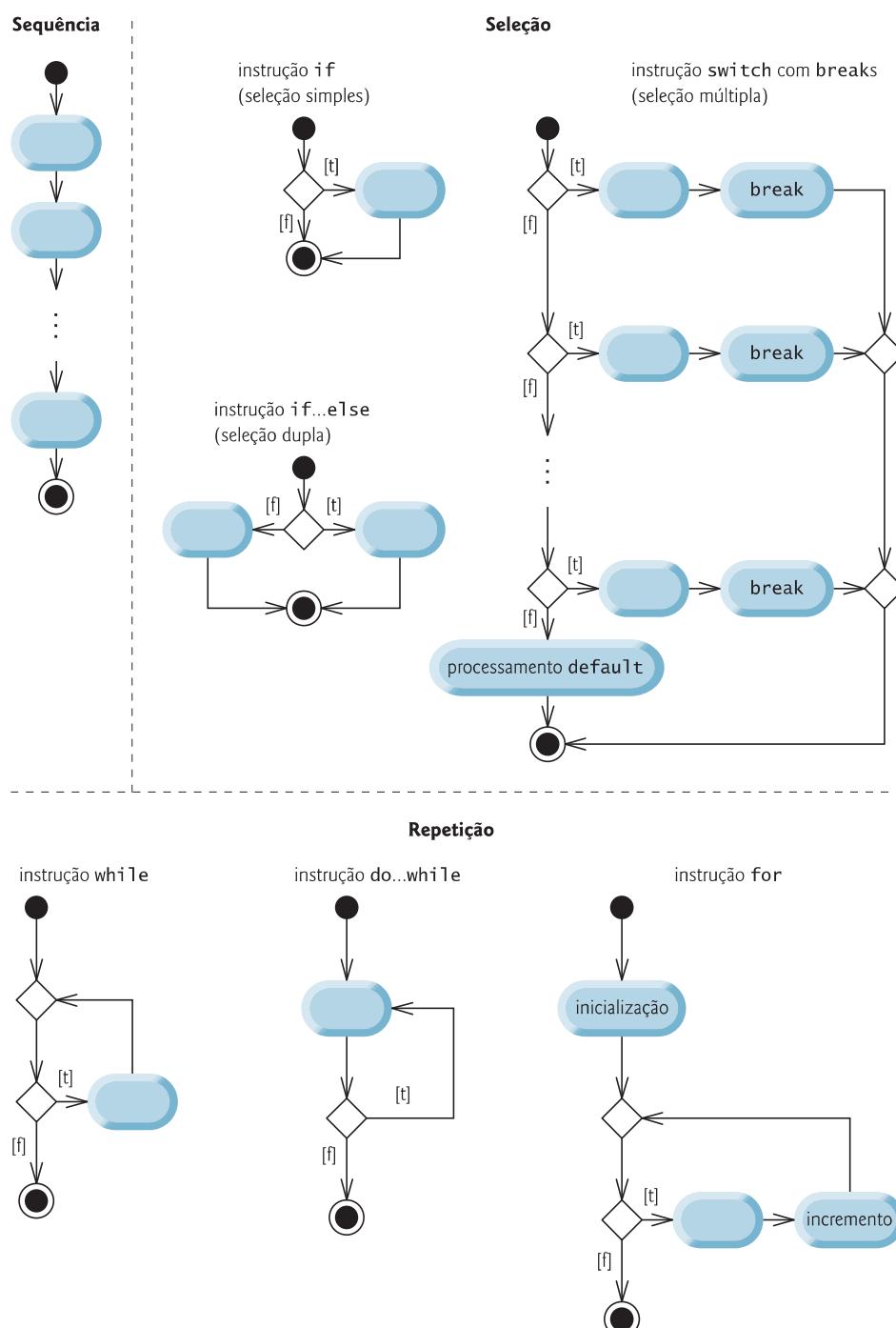


Figura 5.21 | As instruções de sequência de entrada única/saída única, seleção e repetição do Java.

Por simplicidade, o Java inclui apenas instruções de controle de *entrada única/saída única* — há somente uma maneira de entrar e uma de sair de cada instrução de controle. É simples conectar instruções de controle em sequência para formar programas estruturados. O estado final de uma instrução de controle é conectado ao estado inicial da próxima — isto é, as instruções de controle são colocadas uma depois da outra em um programa em sequência. Chamamos isso de *empilhamento de instruções de controle*. As regras para formar programas estruturados também permitem que instruções de controle sejam *aninhadas*.

Regras para formar programas estruturados

A Figura 5.22 mostra as regras para formar adequadamente programas estruturados. As regras supõem que os estados de ação podem ser utilizados para indicar *qualquer* ação. As regras também supõem que iniciamos com o diagrama de atividade simples (Figura 5.23) consistindo somente em um estado inicial, um estado de ação, um estado final e setas de transição.

Aplicar as regras mostradas na Figura 5.22 sempre resulta em um diagrama de atividade adequadamente estruturado com uma apresentação elegante de blocos de construção. Por exemplo, aplicar a regra 2 repetidamente ao diagrama de atividade mais simples resulta em um diagrama de atividade que contém muitos estados de ação em sequência (Figura 5.24). A regra 2 gera uma *pilha* de instruções de controle, então vamos chamá-la de **regra de empilhamento**. As linhas verticais tracejadas na Figura 5.24 não são parte da UML — elas são utilizadas para separar os quatro diagramas de atividades que demonstram a regra 2 da Figura 5.22 sendo aplicada.

A regra 3 é chamada **regra de aninhamento**. Se a regra 3 for aplicada repetidamente ao diagrama de atividades mais simples o resultado é com instruções de controle perfeitamente *aninhadas*. Por exemplo, na Figura 5.25, o estado de ação no diagrama de atividade mais simples é substituído por uma instrução (*if...else*) de seleção dupla. Então, a regra 3 é aplicada novamente aos estados de ação na instrução de seleção dupla, substituindo cada um por uma instrução de seleção dupla. O símbolo de estado de ação tracejado em torno de cada instrução de seleção dupla representa o estado de ação que foi substituído. [Observação: os símbolos de setas tracejadas e de estados de ação tracejados mostrados na Figura 5.25 não fazem parte da UML. Eles são utilizados aqui para ilustrar que *qualquer* estado de ação pode ser substituído por uma instrução de controle.]

A regra 4 gera estruturas maiores, mais complexas e mais profundamente aninhadas. Os diagramas que emergem da aplicação das regras na Figura 5.22 constituem o conjunto de todos os possíveis diagramas de atividade estruturados e, portanto, o conjunto de todos os programas estruturados possíveis. A beleza da abordagem estruturada é que utilizamos *apenas sete* instruções simples de entrada única/saída única e os montamos de *apenas duas* maneiras simples.

Regras para formar programas estruturados

1. Comece com o diagrama de atividades mais simples (Figura 5.23).
2. Qualquer estado da ação pode ser substituído por dois estados da ação na sequência.
3. Qualquer estado de ação pode ser substituído por qualquer instrução de controle (sequência de estados de ação, *if*, *if...else*, *switch*, *while*, *do...while* ou *for*).
4. As regras 2 e 3 podem ser aplicadas com a frequência que você quiser e em qualquer ordem.

Figura 5.22 | Regras para formar programas estruturados.



Figura 5.23 | Diagrama de atividade mais simples.

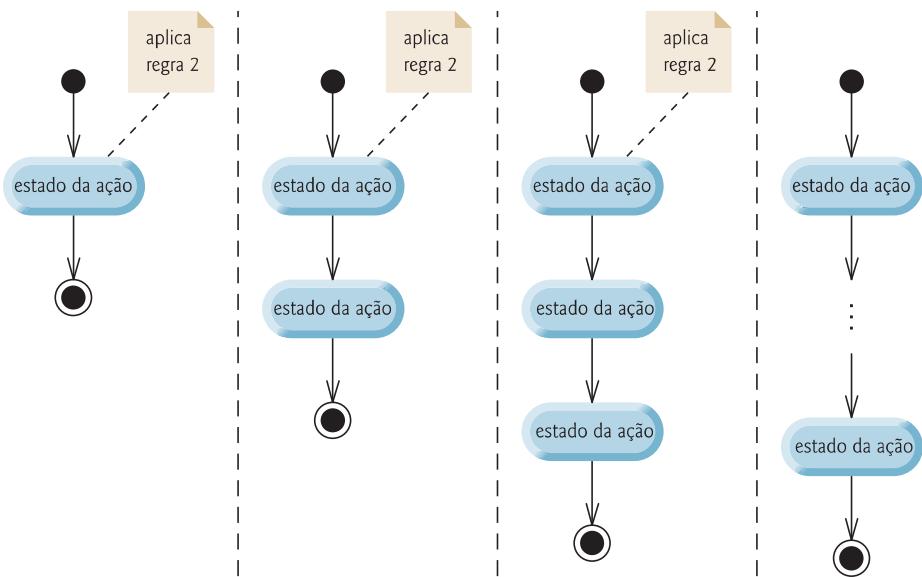


Figura 5.24 | Aplicando repetidamente a regra 2 da Figura 5.22 ao diagrama mais simples.

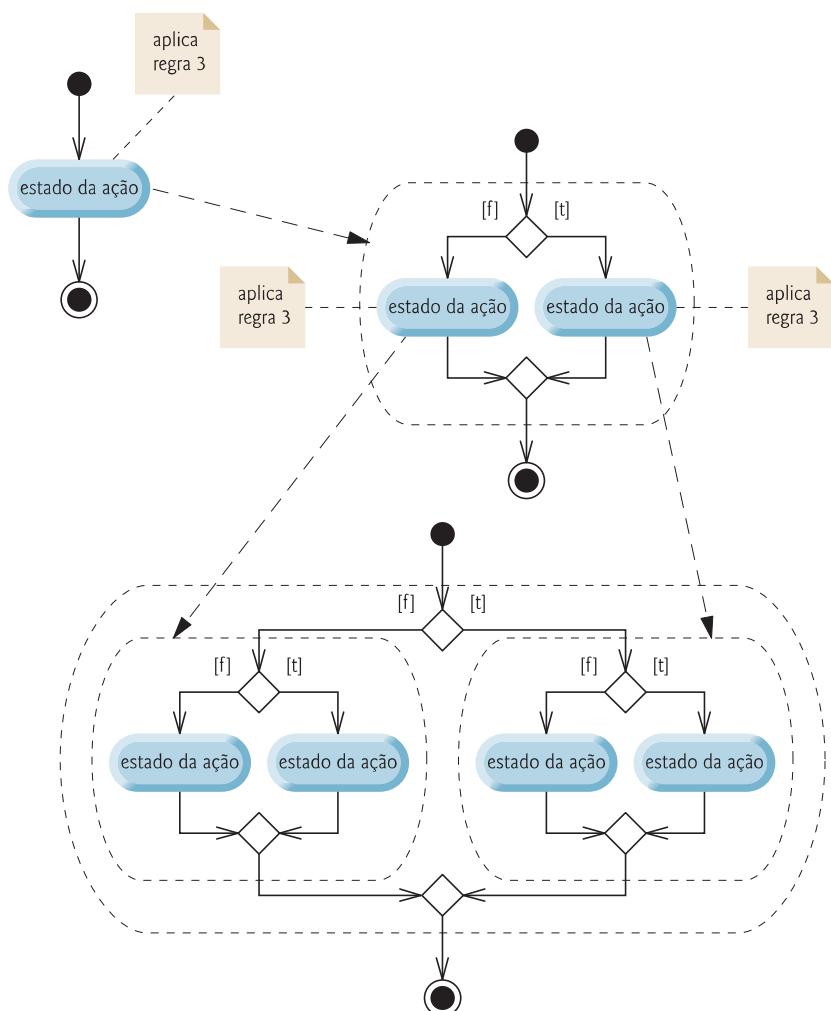


Figura 5.25 | Aplicando repetidamente a regra 3 da Figura 5.22 ao diagrama mais simples.

Se as regras na Figura 5.22 forem seguidas, um diagrama de atividade “não estruturado” (como o da Figura 5.26) não pode ser criado. Se você não tiver certeza se um diagrama particular é estruturado, aplique as regras da Figura 5.21 na ordem inversa para reduzir o diagrama ao diagrama de atividade mais simples. Se puder reduzi-lo, o diagrama original é estruturado; caso contrário, não.

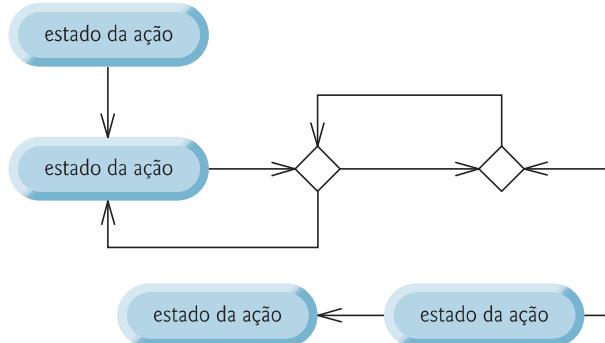


Figura 5.26 | Diagrama de atividade “não estruturado”.

Três formas de controle

A programação estruturada promove a simplicidade. São necessárias apenas três formas de controle para implementar um algoritmo:

- sequência
- seleção
- repetição

A estrutura de sequência é trivial. Liste simplesmente as instruções para executar na ordem em que elas devem executar. A seleção é implementada de uma destas três maneiras:

- instrução `if` (seleção única)
- instrução `if...else` (seleção dupla)
- instrução `switch` (seleção múltipla)

De fato, é simples provar que a instrução simples `if` é suficiente para fornecer *qualquer* forma de seleção — tudo o que pode ser feito com a instrução `if...else` e a instrução `switch` pode ser implementado combinando-se instruções `if` (embora talvez não de modo tão claro e eficiente).

A repetição é implementada de uma destas três maneiras:

- instrução `while`
- instrução `do...while`
- instrução `for`

[*Observação:* há uma quarta instrução de repetição — a instrução `for aprimorada` — que discutiremos na Seção 7.7.] É simples provar que a instrução `while` é suficiente para fornecer *qualquer* forma de repetição. Tudo o que pode ser feito com `do...while` e `for` pode ser feito com a instrução `while` (embora talvez não de uma maneira igualmente conveniente).

A combinação desses resultados ilustra que *qualquer* forma de controle que possa ser necessária um dia em um programa Java pode ser expressa em termos de

- sequência
- instrução `se` (seleção)
- instrução `while` (repetição)

e que podem ser combinadas apenas de duas maneiras — *empilhamento* e *aninhamento*. De fato, a programação estruturada é a essência da simplicidade.

5.11 (Opcional) Estudo de caso de GUIs e imagens gráficas: desenhando retângulos e ovais

Esta seção demonstra como desenhar retângulos e ovais utilizando os métodos `Graphics drawRect` e `drawOval`, respectivamente. Esses métodos são demonstrados na Figura 5.27.

A linha 6 começa a declaração de classe para `Shapes` que estende `JPanel`. A variável de instância `choice` determina se `paintComponent` deve desenhar retângulos ou ovais. O construtor `Shapes` inicializa `choice` com o valor passado no parâmetro `userChoice`.

O método `paintComponent` realiza o desenho real. Lembre-se de que a primeira instrução em cada método `paintComponent` deve ser uma chamada a `super.paintComponent`, como na linha 19. As linhas 21 a 35 repetem-se 10 vezes para desenhar 10 formas. A instrução `switch aninhada` (linhas 24 a 34) escolhe entre desenhar retângulos e desenhar ovais.

Se `choice` for 1, então o programa desenha retângulos. As linhas 27 e 28 chamam o método `Graphics drawRect`. O método `drawRect` requer quatro argumentos. As duas primeiras representam as coordenadas x e y do canto superior esquerdo do retângulo; as duas seguintes representam a largura e altura do retângulo. Nesse exemplo, iniciamos em uma posição de 10 pixels para baixo e 10 pixels à direita do canto superior esquerdo e cada iteração do loop move o canto superior esquerdo outros 10 pixels para baixo e para a direita. A largura e a altura do retângulo iniciam a 50 pixels e aumentam 10 pixels a cada iteração.

Se `choice` for 2, o programa desenha ovais. Ele cria um retângulo imaginário chamado **retângulo delimitador** e posiciona dentro dele uma oval que toca os pontos centrais dos quatro lados. O método `drawOval` (linhas 31 e 32) requer os mesmos quatro argumentos como método `drawRect`. Os argumentos especificam a posição e tamanho do retângulo para a oval. Os valores passados para `drawOval` nesse exemplo são exatamente os mesmos que aqueles passados para `drawRect` nas linhas 27 e 28. Visto que a largura e a altura do retângulo delimitador são idênticas nesse exemplo, as linhas 27 e 28 desenham um *círculo*. Como exercício, tente modificar o programa para desenhar tanto retângulos como ovais para ver como `drawOval` e `drawRect` estão relacionados.

```

1 // Figura 5.27: Shapes.java
2 // Desenhando uma cascata de formas com base na escolha do usuário.
3 import java.awt.Graphics;
4 import javax.swing.JPanel;
5
6 public class Shapes extends JPanel
7 {
8     private int choice; // escolha do usuário de qual forma desenhar
9
10    // construtor configura a escolha do usuário
11    public Shapes(int userChoice)
12    {
13        choice = userChoice;
14    }
15
16    // desenha uma cascata de formas que iniciam do canto superior esquerdo
17    public void paintComponent(Graphics g)
18    {
19        super.paintComponent(g);
20
21        for (int i = 0; i < 10; i++)
22        {
23            // seleciona a forma com base na escolha do usuário
24            switch (choice)
25            {
26                case 1: // desenha retângulos
27                    g.drawRect(10 + i * 10, 10 + i * 10,
28                                50 + i * 10, 50 + i * 10);
29                    break;
30                case 2: // desenha ovais
31                    g.drawOval(10 + i * 10, 10 + i * 10,
32                                50 + i * 10, 50 + i * 10);
33                    break;
34            }
35        }
36    }
37 } // fim da classe Shapes

```

Figura 5.27 | Desenhando uma cascata de formas com base na escolha do usuário.

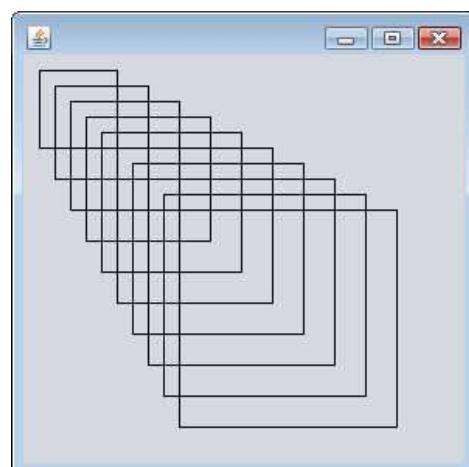
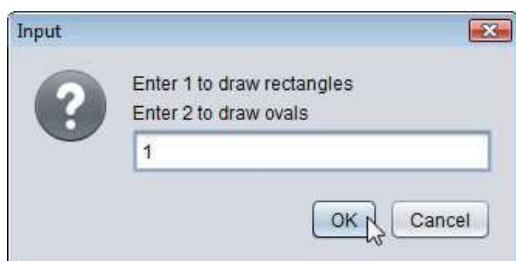
Classe ShapesTest

A Figura 5.28 é responsável por tratar a entrada do usuário e criar uma janela para exibir o desenho adequado com base na resposta do usuário. A linha 3 importa `JFrame` para tratar a exibição, e a linha 4 importa `JOptionPane` para tratar a entrada. As linhas 11 a 13 exibem um prompt para o usuário na forma de um diálogo de entrada e armazenam a resposta do usuário na variável `input`. Observe que, ao exibir múltiplas linhas de texto em um prompt em um `JOptionPane`, você deve usar `\n` para começar uma nova linha de texto, em vez de `%n`. A linha 15 utiliza o método `parseInt` de `Integer` para converter a `String` inserida pelo usuário em um `int` e armazenar o resultado na variável `choice`. A linha 18 cria um objeto `Shapes` e passa a escolha do usuário para o construtor. As linhas 20 a 25 realizam as operações padrão que criam e configuram uma janela nesse estudo de caso — criam um `quadro`, configuram-no para encerrar o aplicativo quando fechado, adicionam o desenho ao quadro, configuram o tamanho dele e o tornam visível.

```

1 // Figura 5.28: ShapesTest.java
2 // Obtendo a entrada de usuário e criando um JFrame para exibir Shapes.
3 import javax.swing.JFrame; // manipula a exibição
4 import javax.swing.JOptionPane;
5
6 public class ShapesTest
7 {
8     public static void main(String[] args)
9     {
10         // obtém a escolha do usuário
11         String input = JOptionPane.showInputDialog(
12             "Enter 1 to draw rectangles\n" +
13             "Enter 2 to draw ovals");
14
15         int choice = Integer.parseInt(input); // converte a entrada em int
16
17         // cria o painel com a entrada do usuário
18         Shapes panel = new Shapes(choice);
19
20         JFrame application = new JFrame(); // cria um novo JFrame
21
22         application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23         application.add(panel);
24         application.setSize(300, 300);
25         application.setVisible(true);
26     }
27 } // fim da classe ShapesTest

```



continua

continuação

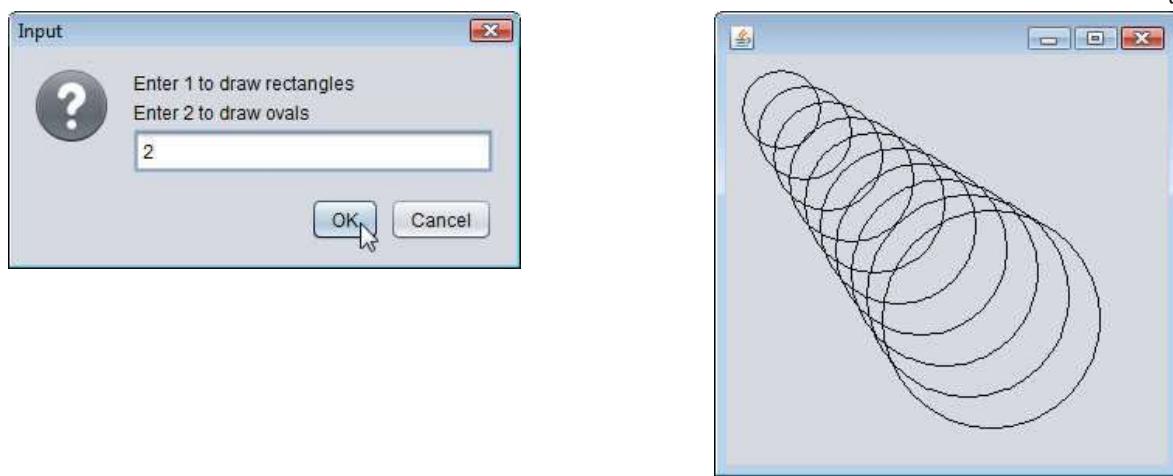


Figura 5.28 | Obtendo a entrada de usuário e criando um JFrame para exibir Shapes.

Exercícios do estudo de caso sobre GUIs e imagens gráficas

- 5.1** Desenhe 12 círculos concêntricos no centro de um JPanel (Figura 5.29). O círculo mais interno deve ter um raio de 10 pixels e cada círculo sucessivo deve ter um raio de 10 pixels maior que o anterior. Comece localizando o centro do JPanel. Para obter o canto superior esquerdo de um círculo, mova-se um raio para cima e um raio para a esquerda a partir do centro. A largura e a altura do retângulo delimitador têm o mesmo diâmetro do círculo (isto é, duas vezes o raio).
- 5.2** Modifique a Questão 5.16 no final dos exercícios do capítulo para ler a entrada utilizando diálogos e exibir o gráfico de barras usando retângulos de comprimentos variados.

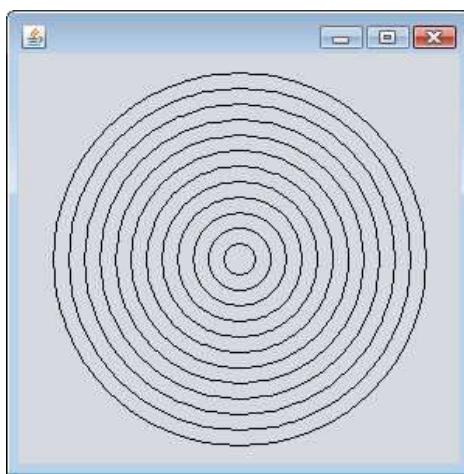


Figura 5.29 | Desenhando círculos concêntricos.

5.12 Conclusão

Neste capítulo, completamos nossa introdução às instruções de controle, que lhe permitem controlar o fluxo de execução em métodos. O Capítulo 4 discutiu `if`, `if...else` e `while`. Este capítulo demonstrou `for`, `do...while` e `switch`. Mostramos que qualquer algoritmo pode ser desenvolvido utilizando combinações da estrutura de sequência, os três tipos das instruções de seleção — `if`, `if...else` e `switch` — e os três tipos de instrução de repetição — `while`, `do...while` e `for`. Neste capítulo e no Capítulo 4, discutimos como você pode combinar esses blocos de construção para utilizar as comprovadas técnicas de construção de programa e solução de problemas. Você usou a instrução `break` para fechar uma instrução `switch` e terminar imediatamente um loop, e utilizou uma instrução `continue` para terminar a iteração atual do loop e passar para a próxima iteração do loop. Este capítulo também introduziu operadores lógicos do Java, que permitem utilizar expressões condicionais mais complexas em instruções de controle. No Capítulo 6, examinamos os métodos em maior profundidade.

Resumo

Seção 5.2 Princípios básicos de repetição controlada por contador

- A repetição controlada por contador requer uma variável de controle, o valor inicial da variável de controle, o incremento pelo qual a variável de controle é modificada a cada passagem pelo loop (também conhecido como cada iteração do loop) e a condição de continuação do loop, que determina se um loop deve continuar.
- Você pode declarar uma variável e inicializá-la na mesma instrução.

Seção 5.3 Instrução de repetição for

- A instrução `while` pode ser utilizada para implementar qualquer loop controlado por contador.
- A instrução `for` especifica todos os detalhes do contador — repetição controlada no cabeçalho.
- Quando a instrução `for` começa a ser executada, sua variável de controle é declarada e inicializada. Se inicialmente a condição de continuação do loop for verdadeira, o corpo será executado. Depois de executar o corpo do loop, a expressão de incremento é executada. Então, o teste de continuação do loop é realizado novamente para determinar se o programa deve continuar com a próxima iteração do loop.
- O formato geral da instrução `for` é

```
for (inicialização; condiçãoDeContinuaçãoDoLoop; incremento)
    instrução
```

onde a expressão *inicialização* nomeia a variável de controle do loop e fornece seu valor inicial, *condiçãoDeContinuaçãoDoLoop* é a condição que determina se o loop deve continuar executando e *incremento* modifica o valor da variável de controle, para que a condição de continuação do loop por fim se torne falsa. Os dois pontos e vírgulas no cabeçalho `for` são necessários.

- A maioria das instruções `for` pode ser representada com instruções `while` equivalentes como a seguir:

```
inicialização;
while (condiçãoDeContinuaçãoDoLoop)
{
    instrução
    incremento;
}
```

- Em geral, as instruções `for` são utilizadas para repetição controlada por contador e as instruções `while`, para repetição controlada por sentinelas.
- Se a expressão de *inicialização* no cabeçalho `for` declarar a variável de controle, esta só poderá ser utilizada nessa instrução `for` — ela não existirá fora da instrução `for`.
- As expressões em um cabeçalho `for` são opcionais. Se a *condiçãoDeContinuaçãoDoLoop* for omitida, o Java irá supor que ela sempre é verdadeira, criando assim um loop infinito. Você poderia omitir a expressão *inicialização* se a variável de controle for inicializada antes do loop. Você poderia omitir a expressão *incremento* se o incremento fosse calculado com instruções no corpo do loop ou se nenhum incremento fosse necessário.
- A expressão *incremento* em uma instrução `for` atua como se ela fosse uma instrução independente no fim do corpo de `for`.
- Uma instrução `for` pode contar para baixo utilizando um incremento negativo — isto é, um decremento. Se a condição de continuação do loop `for` inicialmente `false`, o corpo da instrução `for` não executa.

Seção 5.4 Exemplos com a estrutura for

- O Java trata as constantes de ponto flutuante como `1000.0` e `0.05` como tipo `double`. De maneira semelhante, o Java trata as constantes de número inteiro como `7` e `-22` como tipo `int`.
- O especificador de formato `%4s` gera saída para uma `String` em um tamanho de campo de 4 — isto é, `printf` exibe o valor com pelo menos 4 posições de caractere. Se o valor a ser enviado para a saída for menor do que a largura de 4 posições de caractere, o valor é alinhado à direita no campo por padrão. Se a largura tiver um valor maior do que 4 posições de caractere, o tamanho do campo é expandido para acomodar o número apropriado de caracteres. Para alinhar o valor à esquerda, utilize um número inteiro negativo para especificar o tamanho do campo.
- `Math.pow(x, y)` calcula o valor de `x` elevado à `y`-ésima potência. O método recebe dois argumentos `double` e retorna um valor `double`.
- O flag de formatação vírgula (,) em especificador de formato indica que um valor de ponto flutuante deve ser gerado com um separador de agrupamento. O separador real utilizado é específico à localidade do usuário (isto é, país). Nos Estados Unidos, o número terá vírgulas que separam cada três dígitos e um ponto decimal que separa a parte fracionária do número, como em `1,234.45`.
- O `.` em um especificador de formato indica que o número inteiro à direita é a precisão do número.

Seção 5.5 Instrução de repetição do...while

- A instrução `do...while` é semelhante à instrução `while`. No `while`, o programa testa a condição de continuação do loop no início do loop, antes de executar seu corpo; se a condição for falsa, o corpo nunca será executado. A instrução `do...while` testa a condição de continuação do loop *depois* de executar o corpo do loop; portanto, o corpo sempre executa pelo menos uma vez.

Seção 5.6 A estrutura de seleção múltipla switch

- A instrução `switch` realiza diferentes ações com base nos valores possíveis de uma expressão integral (um valor constante do tipo `byte`, `short`, `int` ou `char`, mas não `long`) ou uma `String`.
- O indicador de fim de arquivo é uma combinação de pressionamento de tecla dependente do sistema que termina a entrada de usuário. Nos sistemas UNIX/Linux/Mac OS X, o fim de arquivo é inserido digitando a sequência `<Ctrl> d` em uma linha separada. Essa notação significa pressionar simultaneamente a tecla `Ctrl` e a tecla `d`. Nos sistemas Windows, insira o fim de arquivo digitando `<Ctrl> z`.
- O método `Scanner` determina se há mais dados a inserir. Esse método retorna o valor `boolean true` se houver mais dados; do contrário, ele retorna `false`. Enquanto o indicador de fim do arquivo não tiver sido digitado, o método `hasNext` retornará `true`.
- A instrução `switch` consiste em um bloco que contém uma sequência de rótulos `case` e um caso `default` opcional. Em um `switch`, o programa avalia a expressão de controle e compara seu valor com cada rótulo `case`. Se ocorrer uma correspondência, o programa executará as instruções para esse `case`.
- Listar casos consecutivamente sem instruções entre eles permite aos casos executar o mesmo conjunto de instruções.
- Cada valor que você deseja testar em um `switch` deve ser listado em um rótulo `case` separado.
- Cada `case` pode ter múltiplas instruções, e essas não precisam ser colocadas entre chaves.
- As instruções de um `case` geralmente terminam com uma instrução `break` que termina a execução do `switch`.
- Sem as instruções `break`, toda vez que ocorre uma correspondência nas instruções `switch`, as instruções para esse caso e casos subsequentes são executadas até que uma instrução `break` ou o fim do `switch` seja encontrado.
- Se não ocorrer nenhuma correspondência entre o valor da expressão controladora e um rótulo `case`, o caso `default` opcional é executado. Se não ocorrer nenhuma correspondência e o `switch` não contiver um caso `default`, o controle de programa simplesmente continua com a primeira instrução depois do `switch`.

Seção 5.7 Estudo de caso da classe AutoPolicy: Strings em instruções switch

- Strings podem ser usadas na expressão de controle da instrução `switch` e rótulos `case`.

Seção 5.8 Instruções break e continue

- A instrução `break`, quando executada em um `while`, `for`, `do...while` ou `switch`, ocasiona a saída imediata dessa instrução.
- A instrução `continue`, quando executada em `while`, `for` ou `do...while`, pula as instruções do corpo remanescentes do loop e passa para a próxima iteração. Nas instruções `while` e `do...while`, o programa avalia o teste de continuação do loop imediatamente. Em uma instrução `for`, a expressão incremento é executada, então o programa avalia o teste de continuação do loop.

Seção 5.9 Operadores lógicos

- As condições simples são expressas em termos dos operadores relacionais `>`, `<`, `>=` e `<=` e os operadores de igualdade `==` e `!=`, e cada expressão testa apenas uma condição.
- Os operadores lógicos permitem-lhe formar condições complexas combinando condições simples. Os operadores lógicos são `&&` (E condicional), `||` (OU condicional), `&` (E lógico booleano), `|` (OU inclusivo lógico booleano), `^` (OU exclusivo lógico booleano) e `!` (NÃO lógico).
- Para assegurar que duas condições são verdadeiras, utilize o operador `&&` (E condicional). Se uma ou as duas condições simples forem falsas, a expressão inteira será falsa.
- Para assegurar que uma das duas *ou* ambas as condições são verdadeiras, utilize o operador `||` (OU condicional), que é avaliado como verdadeiro se uma das ou ambas as condições simples forem verdadeiras.
- Uma condição que usa os operadores `&&` ou `||` utiliza a avaliação em curto-circuito — elas só são avaliadas até que se conheça se a condição é verdadeira ou é falsa.
- Os operadores `&` e `|` funcionam de forma idêntica aos operadores `&&` e `||`, mas sempre avaliam ambos os operandos.
- Uma condição simples que contém o operador OU exclusivo lógico booleano (`^`) é `true` se *e somente se* um de seus operandos for `true` e o outro `false`. Se os dois operandos forem `true` ou ambos forem `false`, a condição inteira é `false`. Também é garantido que esse operador avaliará seus dois operandos.
- O operador unário `!` (NÃO lógico) “inverte” o valor de uma condição.

Exercícios de revisão

5.1 Preencha as lacunas em cada uma das seguintes afirmações:

- Em geral, as instruções _____ são utilizadas para repetição controlada por contador e as instruções _____ são utilizadas para repetição controlada por sentinela.
- A instrução `do...while` testa a condição de continuação do loop _____ de executar o corpo do loop; portanto, o corpo sempre executa pelo menos uma vez.
- A instrução _____ seleciona entre múltiplas ações com base nos possíveis valores de uma variável ou expressão, ou uma `String`.
- A instrução _____, quando executada em uma instrução de repetição, pula as instruções restantes no corpo do loop e prossegue com a próxima iteração do loop.
- O operador _____ pode ser utilizado para assegurar que duas condições são *ambas* verdadeiras antes de escolher certo caminho de execução.
- Se a condição de continuação do loop em um cabeçalho `for` for inicialmente _____, o programa não executará o corpo da instrução `for`.
- Os métodos que realizam as tarefas comuns e não exigem os objetos são chamados de métodos _____.

5.2 Determine se cada uma das seguintes alternativas é *verdadeira* ou *falsa*. Se *falsa*, explique por quê.

- O caso `default` é requerido na instrução de seleção `switch`.
- A instrução `break` é requerida no último caso de uma instrução de seleção `switch`.
- A expressão `((x > y) && (a < b))` é verdadeira se `x > y` for verdadeiro ou `a < b` for verdadeira.
- Uma expressão contendo o operador `||` é verdadeira se um ou ambos de seus operandos forem verdadeiros.
- O flag de formatação vírgula `(,)` em um especificador de formato (por exemplo, `%,20.2f`) indica que um valor deve ser enviado para a saída com um separador de milhares.
- Para testar para uma série de valores em uma instrução `switch`, utilize um hífen `(-)` entre os valores inicial e final da série em um rótulo `case`.
- Listar casos consecutivamente sem instruções entre eles permite aos casos executar o mesmo conjunto de instruções.

5.3 Escreva uma instrução Java ou um conjunto de instruções Java para realizar cada uma das seguintes tarefas:

- Some os inteiros ímpares entre 1 e 99 utilizando uma instrução `for`. Assuma que as variáveis de inteiro `sum` e `count` foram declaradas.
- Calcule o valor de `2.5` elevado à potência de `3`, utilizando o método `pow`.
- Imprima os inteiros de 1 a 20, utilizando um loop `while` e a variável contadora `i`. Assuma que a variável `i` foi declarada, mas não foi inicializada. Imprima apenas cinco inteiros por linha. [Dica: utilize o cálculo `i % 5`. Quando o valor dessa expressão for 0, imprima um caractere de nova linha; caso contrário, imprima um caractere de tabulação. Assuma que esse código é um aplicativo. Utilize o método `System.out.println()` para imprimir o caractere de nova linha, e utilize o método `System.out.print('\\t')` para imprimir o caractere de tabulação.]
- Repita a parte (c) utilizando uma instrução `for`.

5.4 Localize o erro em cada um dos seguintes segmentos de código e explique como corrigi-los:

- ```
i = 1;
while (i <= 10);
 ++i;
}
```
- ```
for (k = 0.1; k != 1.0; k += 0.1)
    System.out.println(k);
```
- ```
switch (n)
{
 case 1:
 System.out.println("The number is 1");
 case 2:
 System.out.println("The number is 2");
 break;
 default:
 System.out.println("The number is not 1 or 2");
 break;
}
```
- O seguinte código deve imprimir os valores 1 a 10:

```
n = 1;
while (n < 10)
 System.out.println(n++);
```

## Respostas dos exercícios de revisão

- 5.1** a) `for`, `while`. b) depois. c) `switch`. d) `continue`. e) `&&` (E condicional). f) `false`. g) `static`.
- 5.2** a) Falso. O caso `default` é opcional. Se nenhuma ação padrão for necessária, então não há necessidade de um caso `default`. b) Falso. A instrução `break` é utilizada para sair da instrução `switch`. A instrução `break` não é necessária para o último caso em uma instrução `switch`. c) Falso. Ambas as expressões relacionais devem ser verdadeiras para a expressão inteira ser verdadeira ao utilizar o operador `&&`. d) Verdadeiro. e) Verdadeiro. f) Falso. A instrução `switch` não fornece um mecanismo para testar intervalos de valores, então cada valor que deve ser testado deve ser listado em um rótulo `case` separado. g) Verdadeiro.
- 5.3** a) `sum = 0;`  
`for (count = 1; count <= 99; count += 2)`  
 `sum += count;`
- b) `double result = Math.pow(2.5, 3);`
- c) `i = 1;`
- ```

while (i <= 20)
{
    System.out.print(i);

    if (i % 5 == 0)
        System.out.println();
    else
        System.out.print('\t');

    ++i;
}
```
- d) `for (i = 1; i <= 20; i++)`
`{`
 `System.out.print(i);

 if (i % 5 == 0)
 System.out.println();
 else
 System.out.print('\t');`
- 5.4** a) Erro: o ponto e vírgula depois do cabeçalho `while` causa um loop infinito, e há uma chave esquerda ausente.
Correção: substitua o ponto e vírgula por uma { ou remova o ; e a }.
- b) Erro: utilizar um número de ponto flutuante para controlar uma instrução `for` pode não funcionar, porque os números de ponto flutuante só são representados aproximadamente pela maioria dos computadores.
Correção: utilize um inteiro e realize o cálculo adequado a fim de obter os valores que você deseja:
- ```

for (k = 1; k != 10; k++)
 System.out.println((double) k / 10);
```
- c) Erro: o código ausente é a instrução `break` nas instruções para o primeiro `case`.  
Correção: adicione uma instrução `break` ao fim das instruções para o primeiro `case`. Esta omissão não é necessariamente um erro se você quiser que a instrução `case 2`: execute cada vez que a instrução `case 1`: executar.
- d) Erro: um operador relacional inadequado é utilizado na condição de continuação de `while`.  
Correção: utilize `<=` em vez de `<` ou altere 10 para 11.

## Questões

- 5.5** Descreva os quatro elementos básicos de repetição controlada por contador.
- 5.6** Compare e contraste as instruções de repetição `while` e `for`.
- 5.7** Discuta uma situação em que seria mais adequado utilizar uma instrução `do...while` do que uma instrução `while`. Explique por quê.
- 5.8** Compare e contraste as instruções `break` e `continue`.
- 5.9** Localize e corrija o(s) erro(s) em cada um dos seguintes segmentos de código:
- a) `For (i = 100, i >= 1, i++)`  
 `System.out.println(i);`
- b) O seguinte código deve imprimir se o inteiro `value` for par ou ímpar:
- ```

switch (value % 2)
{
    case 0:
        System.out.println("Even integer");
```

```
case 1:  
    System.out.println("Odd integer");  
}
```

- c) O código a seguir deve dar saída dos inteiros ímpares de 19 a 1:

```
for (i = 19; i >= 1; i += 2)  
    System.out.println(i);
```

- d) O código seguinte deve dar saída dos inteiros pares de 2 a 100:

```
counter = 2;
```

```
do
{
    System.out.println(counter);
    counter += 2;
} While (counter < 100);
```

5.10 O que o seguinte programa faz?

```
1 // Exercício 5.10: Printing.java
2 public class Printing
3 {
4     public static void main(String[] args)
5     {
6         for (int i = 1; i <= 10; i++)
7         {
8             for (int j = 1; j <= 5; j++)
9                 System.out.print('@');
10
11             System.out.println();
12         }
13     }
14 } // fim da classe Printing
```

- 5.11** (*Localize o menor valor*) Escreva um aplicativo que localiza o menor de vários números inteiros. Suponha que o primeiro valor lido especifica o número de valores a serem inseridos pelo usuário.

5.12 (*Calculando o produto de números inteiros ímpares*) Escreva um aplicativo que calcula o produto dos números inteiros ímpares de 1 a 15.

5.13 (*Fatoriais*) Fatoriais costumam ser utilizados em problemas de probabilidade. O fatorial de um inteiro positivo n (escrito como $n!$ e pronunciado como “fatorial de n ”) é igual ao produto dos números inteiros positivos de 1 a n . Escreva um aplicativo que calcula os fatoriais de 1 a 20. Utilize o tipo `long`. Exiba os resultados em formato tabular. Que dificuldade poderia impedir você de calcular o fatorial de 100?

5.14 (*Programa de juros compostos modificado*) Modifique o aplicativo de juros compostos da Figura 5.6 para repetir os passos para taxas de juros de 5%, 6%, 7%, 8%, 9% e 10%. Utilize um loop `for` para variar a taxa de juros.

5.15 (*Programa para impressão de triângulos*) Escreva um aplicativo que exibe os seguintes padrões separadamente, um embaixo do outro. Utilize loops `for` para gerar os padrões. Todos os asteriscos (*) devem ser impressos por uma única instrução na forma `System.out.print('*');` o que faz com que os asteriscos sejam impressos lado a lado. Uma instrução na forma `System.out.println();` pode ser utilizada para mover-se para a próxima linha. Uma instrução na forma `System.out.print(' ');` pode ser utilizada para exibir um espaço para os últimos dois padrões. Não deve haver outras instruções de saída no programa. [Dica: os dois últimos padrões requerem que cada linha inicie com um número adequado de espaços em branco.]

- | (a) | (b) | (c) | (d) |
|-------|-------|-------|-------|
| * | ***** | ***** | * |
| ** | ***** | ***** | ** |
| *** | ***** | ***** | *** |
| **** | ***** | ***** | **** |
| ***** | ***** | ***** | ***** |
| ***** | ***** | ***** | ***** |
| ***** | *** | *** | ***** |
| ***** | ** | ** | ***** |
| ***** | | * | ***** |

- 5.16** (*Gráfico de barras do programa de impressão*) Uma aplicação interessante dos computadores é exibir diagramas e gráficos de barras. Escreva um aplicativo que leia cinco números entre 1 e 30. Para cada número que é lido, seu programa deve exibir o mesmo número de asteriscos adjacentes. Por exemplo, se seu programa lê o número 7, ele deve exibir *****. Exiba as barras dos asteriscos depois de ler os cinco números.

5.17 (*Calculando vendas*) Um varejista on-line vende cinco produtos cujos preços no varejo são como a seguir: produto 1, US\$ 2,98; produto 2, US\$ 4,50; produto 3, US\$ 9,98; produto 4, US\$ 4,49 e produto 5, US\$ 6,87. Escreva um aplicativo que leia uma série de pares de números como segue:

 - a) número de produto
 - b) quantidade vendida

Seu programa deve utilizar uma instrução `switch` para determinar o preço de varejo de cada produto. Você deve calcular e exibir o valor de varejo total de todos os produtos vendidos. Utilize um loop controlado por sentinela para determinar quando o programa deve parar o loop e exibir os resultados finais.

5.18 (*Programa de juros compostos modificado*) Modifique o aplicativo na Figura 5.6 para utilizar apenas inteiros para calcular os juros compostos. [Dica: trate todas as quantidades monetárias como números inteiros em centavos. Então, divida o resultado em suas partes dólar e centavos utilizando as operações divisão e resto, respectivamente. Insira uma vírgula entre as partes dólar e centavos.]

5.19 Suponha que $i = 1$, $j = 2$, $k = 3$ e $m = 2$. O que cada uma das seguintes instruções imprime?

 - a) `System.out.println(i == 1);`
 - b) `System.out.println(j == 3);`
 - c) `System.out.println((i >= 1) && (j < 4));`
 - d) `System.out.println((m <= 99) & (k < m));`
 - e) `System.out.println((j >= i) || (k == m));`
 - f) `System.out.println((k + m < j) | (3 - j >= k));`
 - g) `System.out.println(!(k > m));`

5.20 (*Calculando o valor de π*) Calcule o valor de π a partir de uma série infinita

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Imprima uma tabela que mostre o valor aproximado de π calculando os 200.000 primeiros termos dessa série. Quantos termos você tem de utilizar antes de primeiro obter um valor que começa com 3,14159?

5.21 (*Triplos de Pitágoras*) Um triângulo retângulo pode ter lados cujos comprimentos são todos inteiros. O conjunto de três valores inteiros para os comprimentos dos lados de um triângulo retângulo é chamado de triplo de Pitágoras. Os comprimentos dos três lados devem satisfazer a relação de que a soma dos quadrados de dois dos lados é igual ao quadrado da hipotenusa. Escreva um aplicativo para exibir uma tabela dos triplos de Pitágoras para `side1`, `side2` e `hypotenuse`, todos não maiores que 500. Utilize um loop `for` triplamente aninhado que tenta todas as possibilidades. Esse é um método de computação de “força bruta”. Você aprenderá nos cursos de ciência da computação mais avançados que para muitos problemas interessantes não há uma abordagem algorítmica conhecida além do uso de força bruta absoluta.

5.22 (*Programa de impressão de triângulos modificado*) Modifique a Questão 5.15 para combinar seu código dos quatro triângulos de asteriscos separados, de modo que todos os quatro padrões sejam impressos lado a lado. [Dica: faça uso inteligente de loops `for` aninhados.]

5.23 (*Leis de De Morgan*) Neste capítulo, discutimos os operadores lógicos `&&`, `&`, `||`, `|`, `^` e `!`. As leis de De Morgan às vezes podem tornar mais convenientes para expressar uma expressão lógica. Essas leis afirmam que a expressão `!(condição1 && condição2)` é logicamente equivalente à expressão `(!condição1 || !condição2)`. Além disso, a expressão `!(condição1 || condição2)` é logicamente equivalente à expressão `(!condição1&& !condição2)`. Utilize as leis de De Morgan para escrever expressões equivalentes para cada uma das expressões a seguir, então escreva um aplicativo para mostrar que tanto a expressão original como a nova expressão em cada caso produzem o mesmo valor:

 - a) `!(x < 5) && !(y >= 7)`
 - b) `!(a == b) || !(g != 5)`
 - c) `!((x <= 8) && (y > 4))`
 - d) `!((i > 4) || (j <= 6))`

5.24 (*Programa de impressão de losangos*) Escreva um aplicativo que imprime a seguinte forma de um losango. Você pode utilizar instruções de saída que imprimem um único asterisco (*), um único espaço ou um único caractere de nova linha. Maximize sua utilização de repetição (com instruções `for` aninhadas) e minimize o número de instruções de saída.

- 5.25** (*Programa de impressão de losangos modificado*) Modifique o aplicativo que você escreveu na Questão 5.24 para ler um número ímpar no intervalo 1 a 19 para especificar o número de linhas no losango. Seu programa então deve exibir um losango do tamanho apropriado.
- 5.26** Uma crítica à instrução `break` e à instrução `continue` é que cada uma é desestruturada. Na verdade, essas instruções sempre podem ser substituídas por instruções estruturadas, embora fazer isso possa ser difícil. Descreva de maneira geral como você removeria qualquer instrução `break` de um loop em um programa e a substituiria por alguma equivalente estruturada. [Dica: a instrução `break` sai de um loop do corpo do loop. A outra maneira de sair de um loop é falhando no teste de continuação do loop. Considere a possibilidade de utilizar no teste de continuação do loop um segundo teste que indica “saída prévia por causa de uma condição ‘break’.”] Utilize a técnica que você desenvolve aqui para remover a instrução `break` do aplicativo na Figura 5.13.
- 5.27** O que o seguinte segmento de programa faz?
- ```

for (i = 1; i <= 5; i++)
{
 for (j = 1; j <= 3; j++)
 {
 for (k = 1; k <= 4; k++)
 System.out.print('*');

 System.out.println();
 } // fim do for interno

 System.out.println();
} // fim do for externo

```
- 5.28** Descreva de maneira geral como você removeria qualquer instrução `continue` de um loop em um programa e a substituiria por alguma equivalente estruturada. Utilize a técnica que você desenvolve aqui para remover a instrução `continue` do programa na Figura 5.14.
- 5.29** (*A canção “The Twelve Days of Christmas”*) Escreva um aplicativo que utiliza instruções de repetição e `switch` para imprimir a canção “The Twelve Days of Christmas”. Uma instrução `switch` deve ser utilizada para imprimir o dia (“primeiro”, “segundo” etc.). Uma instrução `switch` separada deve ser utilizada para imprimir o restante de cada verso. Visite o site [en.wikipedia.org/wiki/The\\_Twelve\\_Days\\_of\\_Christmas\\_\(song\)](http://en.wikipedia.org/wiki/The_Twelve_Days_of_Christmas_(song)) para obter a letra da música.
- 5.30** (*Classe AutoPolicy modificada*) Modifique a classe `AutoPolicy` na Figura 5.11 para validar os códigos de estado de duas letras para os estados do nordeste dos EUA. Os códigos são: CT para Connecticut, MA para Massachusetts, ME para Maine, NH para New Hampshire, NJ para New Jersey, NY para Nova York, PA para Pensilvânia e VT para Vermont. No método `AutoPolicy setState` use o operador lógico OU (`||`) (Seção 5.9) para criar uma condição composta em uma instrução `if...else` que compara o argumento do método com cada código de duas letras. Se o código estiver incorreto, a parte `else` da instrução `if...else` deve exibir uma mensagem de erro. Nos próximos capítulos, você aprenderá a usar o tratamento de exceções para indicar que um método recebeu um valor inválido.

## Fazendo a diferença

- 5.31** (*Perguntas sobre fatos do aquecimento global*) A controversa questão do aquecimento global foi amplamente divulgada no filme “Uma verdade inconveniente,” apresentando o ex-vice-presidente Al Gore. Gore e uma rede de cientistas da ONU, o Grupo Intergovernamental sobre Alterações Climáticas, dividiu o Prêmio Nobel da Paz de 2007 em reconhecimento aos “seus esforços para fomentar e disseminar melhor conhecimento sobre as mudanças climáticas feitas pelo homem”. Pesquise on-line os dois lados da questão em relação ao aquecimento global (é recomendável pesquisar frases como “global warming skeptics” [“céticos do aquecimento global”]). Crie um questionário de múltipla escolha com cinco perguntas sobre o aquecimento global, cada uma tendo quatro possíveis respostas (numeradas 1 a 4). Seja objetivo e tente representar de uma maneira justa ambos os lados da questão. Em seguida, escreva um aplicativo que administre o questionário, calcule o número de respostas corretas (zero a cinco) e retorne uma mensagem ao usuário. Se o usuário responder corretamente cinco perguntas, imprima “Excelente”; se responder quatro, imprima “Muito bom”; se responder três ou menos, imprima “É o momento de aprimorar seu conhecimento sobre o aquecimento global” e inclua uma lista de alguns sites onde você encontrou os fatos.
- 5.32** (*Alternativas para o planejamento tributário; o “Imposto justo”*) Há muitas propostas para tornar a tributação mais justa. Verifique a iniciativa FairTax norte-americana em [www.fairtax.org](http://www.fairtax.org). Pesquise como o FairTax proposto funciona. Uma sugestão é eliminar impostos de renda e a maioria dos outros impostos a favor de um imposto de consumo de 23% sobre todos os produtos e serviços que você compra. Alguns oponentes do FairTax questionam o percentual de 23% e afirmam que, por causa da maneira como o imposto é calculado, seria mais exato dizer que a taxa é 30% — verifique isso cuidadosamente. Escreva um programa que peça ao usuário para inserir despesas nas várias categorias de despesas que ele tem (por exemplo, moradia, alimentação, vestuário, transporte, educação, assistência médica e férias) e então imprima o FairTax estimado que a pessoa pagaria.
- 5.33** (*Crescimento da base de usuários do Facebook*) De acordo com o CNNMoney.com, o Facebook alcançou um bilhão de usuários em outubro de 2012. Usando a técnica de crescimento composto que você aprendeu na Figura 5.6 e supondo que a base de usuários cresça a uma taxa de 4% ao mês, quantos meses levarão para que o Facebook aumente sua base de usuários para 1,5 bilhão? Quantos meses serão necessários para que o Facebook expanda sua base de usuários para dois bilhões?