

15

Recursão



OBJETIVOS

- Neste capítulo, você aprenderá:
- O conceito de recursão.
- Como escrever e utilizar métodos recursivos.
- Como determinar o caso básico e o passo de recursão em um algoritmo recursivo.
- Como chamadas de método recursivo são tratadas pelo sistema.
- As diferenças entre recursão e iteração e quando é apropriado utilizar cada uma.



- 15.1** **Introdução**
- 15.2** **Conceitos de recursão**
- 15.3** **Exemplo que utiliza recursão: Fatoriais**
- 15.4** **Exemplo que utiliza recursão: Série de Fibonacci**
- 15.5** **Recursão e a pilha de chamadas de método**
- 15.6** **Recursão *versus* iteração**
- 15.7** **Permutações de string**
- 15.8** **Torres de Hanói**
- 15.9** **Fractais**
- 15.10** **Retorno recursivo**
- 15.11** **Conclusão**
- 15.12** **Internet e recursos Web**

15.1 Introdução

- **Programas anteriores estruturados como métodos chamam uns aos outros de uma maneira hierárquica e disciplinada.**
- **Métodos recursivos:**
 - chamam a si mesmos,
 - úteis para alguns problemas a fim de definir uma chamada ao próprio método; e
 - podem ser chamados direta ou indiretamente por um outro método.



Capítulo Exemplos de recursão e exercícios neste livro

- 15**
- Método fatorial (figuras 15.3 e 15.4)**
 - Método de Fibonacci (figuras 15.5 e 15.6)**
 - Permutações de string (figuras 15.12 e 15.13)**
 - Torres de Hanói (figuras 15.15 e 15.16)**
 - Fractais (figuras 15.23 e 15.24)**
 - O que faz esse código? (Exercício 15.7, Exercício 15.12 e Exercício 15.13)**
 - Localize o erro no seguinte código (Exercício 15.8)**
 - Elevando um inteiro à potência de um inteiro (Exercício 15.9)**
 - Visualizando a recursão (Exercício 15.10)**
 - Máximo divisor comum (Exercício 15.11)**
 - Determine se uma string é um palíndromo (Exercício 15.14)**
 - Oito rainhas (Exercício 15.15)**
 - Imprima um array (Exercício 15.16)**
 - Imprima um array de trás para frente (Exercício 15.17)**
 - Valor mínimo em um array (Exercício 15.18)**
 - Estrela fractal (Exercício 15.19)**
 - Percorrendo um labirinto com a reversão recursiva (Exercício 15.20)**
 - Gerando labirintos aleatoriamente (Exercício 15.21)**
 - Labirintos de qualquer tamanho (Exercício 15.22)**
 - Tempo necessário para calcular um número de Fibonacci (Exercício 15.23)**

**Figura 15.1 | Resumo dos 32 exemplos de recursão e exercícios neste texto.
(Parte 1 de 2.)**



Capítulo Exemplos de recursão e exercícios neste livro	
16	Classificação por intercalação (figuras 16.10 e 16.11) Pesquisa linear (Exercício 16.8) Pesquisa binária (Exercício 16.9) Classificação rápida (Quicksort) (Exercício 16.10)
17	Inserção de árvore binária (Figura 17.17) Percorrendo uma árvore binária na pré-ordem (Figura 17.17) Percorrendo uma árvore binária na ordem (Figura 17.17) Percorrendo uma árvore binária na pós-ordem (Figura 17.17) Impressão de uma lista vinculada de trás para frente (Exercício 17.20) Pesquisa em uma lista vinculada (Exercício 17.21)

**Figura 15.1 | Resumo dos 32 exemplos de recursão e exercícios neste texto.
(Parte 2 de 2.)**



15.2 Conceitos de recursão

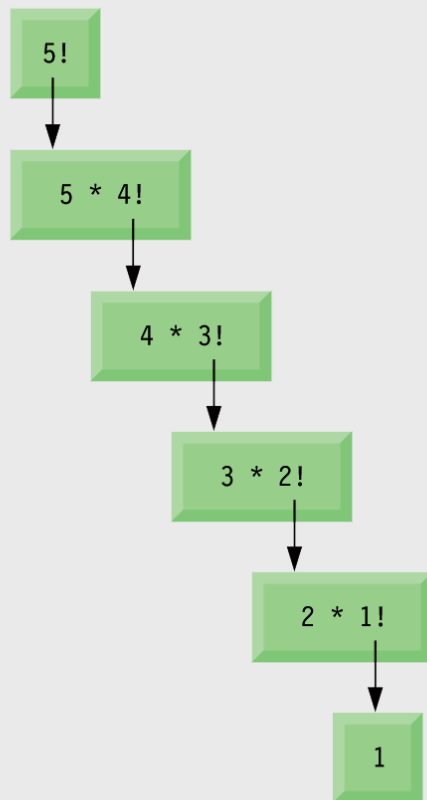
- **Elementos recursivos de solução de problemas:**
 - **Caso básico:**
 - Método recursivo só é capaz de resolver o caso mais simples — o caso básico.
 - Se o método for chamado com o caso básico, o método retorna um resultado.
 - Se o método for chamado com um problema mais complexo, o problema será dividido em duas partes — uma parte que o método sabe o que fazer e uma outra que o método não sabe o que fazer (denominada *chamada recursiva* ou *passo de recursão*).
 - **Chamada recursiva/passos de recursão:**
 - Deve assemelhar-se ao problema original, porém ser um pouco mais simples ou a menor versão.
 - O método chama uma cópia atualizada dele mesmo a fim de trabalhar em um problema menor.
 - Normalmente, inclui uma instrução `return`
- **Recursão indireta:**
 - O método recursivo chama um outro método que, conseqüentemente, faz uma chamada de volta ao método recursivo.



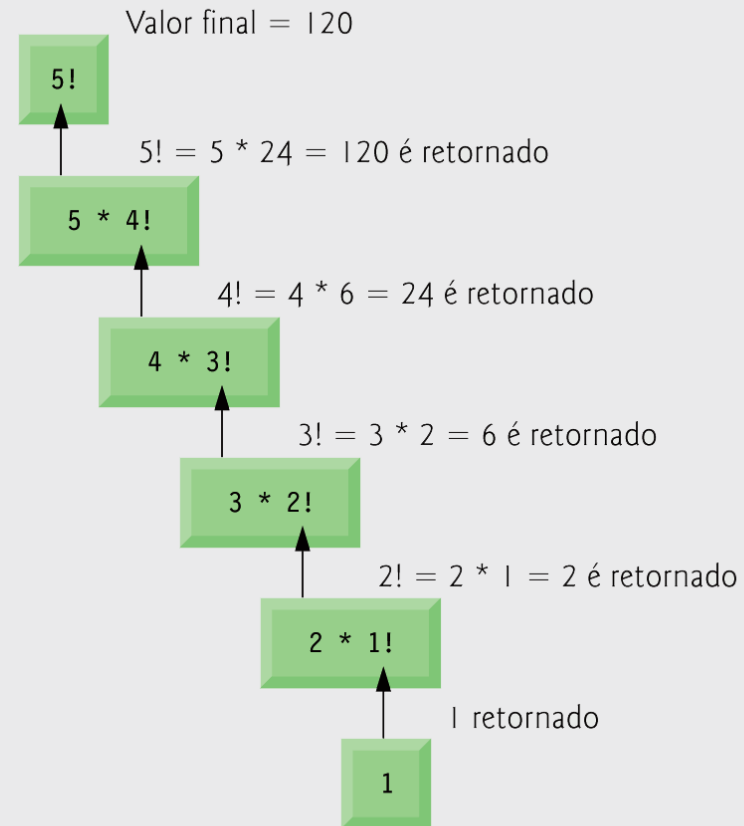
15.3 Exemplo que utiliza recursão: Fatoriais

- Fatorial de n , ou $n!$ é o produto
 - $n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1$
 - Com $1!$ igual a 1 e $0!$ definido como 1.
- Pode ser resolvido recursiva ou iterativamente (não-recursivamente).
- Solução recursiva utiliza o relacionamento a seguir:
 - $n! = n \cdot (n - 1)!$
- Recursão infinita – chamadas recursivas são feitas continuamente até que a memória tenha sido exaurida.
 - É causada omitindo o caso básico ou escrevendo um passo de recursão que não converge com o caso básico.





(a) Seqüência de chamadas recursivas



(b) Valores retornados por cada chamada recursiva

Figura 15.2 | Avaliação recursiva de 5!.

Resumo

Fatorial

FactorialCalculator.java

Caso básico retorna 1

Passo de recursão divide o problema em duas partes:
com uma o método sabe como fazer; com outra, não

Método de apresentação

Chamada recursiva: O método `portion` não sabe
como fazer; a menor versão do problema original

Chamada original ao método
recursivo

```

1 // Fig. 15.3: FactorialCalculator.java
2 // Método fatorial recursivo.
3
4 public class FactorialCalculator
5 {
6     // método fatorial recursivo
7     public long factorial( long number )
8     {
9         if ( number <= 1 ) // testa caso básico
10             return 1; // casos básicos: 0! = 1 e 1! = 1
11         else // passo de recursão
12             return number * factorial( number - 1 );
13     } // fim do método factorial
14
15     // gera saída de fatoriais
16     public void displayFactorials()
17     {
18         // calcula os fatoriais de 0 a 10
19         for ( int counter = 0; counter <= 10; counter++ )
20             System.out.printf( "%d! = %d\n", counter, factorial( counter ) );
21     } // fim do método displayFactorials
22 } // fim da classe FactorialCalculator
  
```



Erro comum de programação 15.1

Omitir o caso básico ou escrever o passo de recursão incorretamente de modo que não convirja para o caso básico pode causar um erro de lógica conhecido como recursão infinita, em que as chamadas recursivas são feitas continuamente até acabar a memória. Isso é análogo ao problema de um loop infinito em uma solução iterativa (não recursiva).



Resumo

FactorialTest.java


```
1 // Fig. 15.4: FactorialTest.java
2 // Testando método fatorial recursivo.
3
4 public class FactorialTest
5 {
6     // calcula fatoriais de 0-10
7     public static void main( String args[] )
8     {
9         FactorialCalculator factorialCalculator = new Fa
10         factorialCalculator.displayFactorials();
11     } // fim de main
12 } // fim da classe FactorialTest
```

Calcula e exibe fatoriais

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```



15.4 Exemplo que utiliza recursão: Série de Fibonacci

- A série de Fibonacci inicia com 0 e 1 e tem a propriedade de que cada número de Fibonacci subsequente é a soma dos dois números de Fibonacci anteriores.
- A série ocorre na natureza; a taxa de números de Fibonacci sucessivos converge de acordo com a taxa ou a média áurea.
- Fibonacci, série definida recursivamente como:
 - $\text{fibonacci}(0) = 0$
 - $\text{fibonacci}(1) = 1$
 - $\text{fibonacci}(n) = \text{fibonacci}(n - 1) + \text{fibonacci}(n - 2)$
- Solução recursiva para cálculo de valores de Fibonacci resulta na explosão das chamadas de métodos recursivos. 

Resumo

```

1 // Fig. 15.5: FibonacciCalculator.java
2 // Método fibonacci recursivo.
3
4 public class FibonacciCalculator
5 {
6     // declaração recursiva do método fibonacci
7     public long fibonacci( long number )
8     {
9         if ( ( number == 0 ) || ( number == 1 ) ) // casos básicos
10             return number;
11         else // passo de recursão
12             return fibonacci( number - 1 ) + fibonacci( number - 2 );
13     } // fim do método fibonacci
14
15     public void displayFibonacci()
16     {
17         for ( int counter = 0; counter <= 10; counter++ )
18             System.out.printf( "Fibonacci of %d is: %d\n", counter,
19                               fibonacci( counter ) );
20     } // fim do método displayFibonacci
21 } // fim da classe FibonacciCalculator

```

Dois casos básicos

Duas chamadas recursivas

Chamada original ao método recursivo



Resumo

FibonacciTest.java

```
1 // Fig. 15.6: FibonacciTest.java
2 // Testando o método fibonacci recursivo.
3
4 public class FibonacciTest
5 {
6     public static void main( String args[] )
7     {
8         FibonacciCalculator fibonacciCalculator = new FibonacciCalculator();
9         fibonacciCalculator.displayFibonacci();
10    } // fim de main
11 } // fim da classe FibonacciTest
```

Calcula e exibe os valores de Fibonacci

```
Fibonacci of 0 is: 0
Fibonacci of 1 is: 1
Fibonacci of 2 is: 1
Fibonacci of 3 is: 2
Fibonacci of 4 is: 3
Fibonacci of 5 is: 5
Fibonacci of 6 is: 8
Fibonacci of 7 is: 13
Fibonacci of 8 is: 21
Fibonacci of 9 is: 34
Fibonacci of 10 is: 55
```



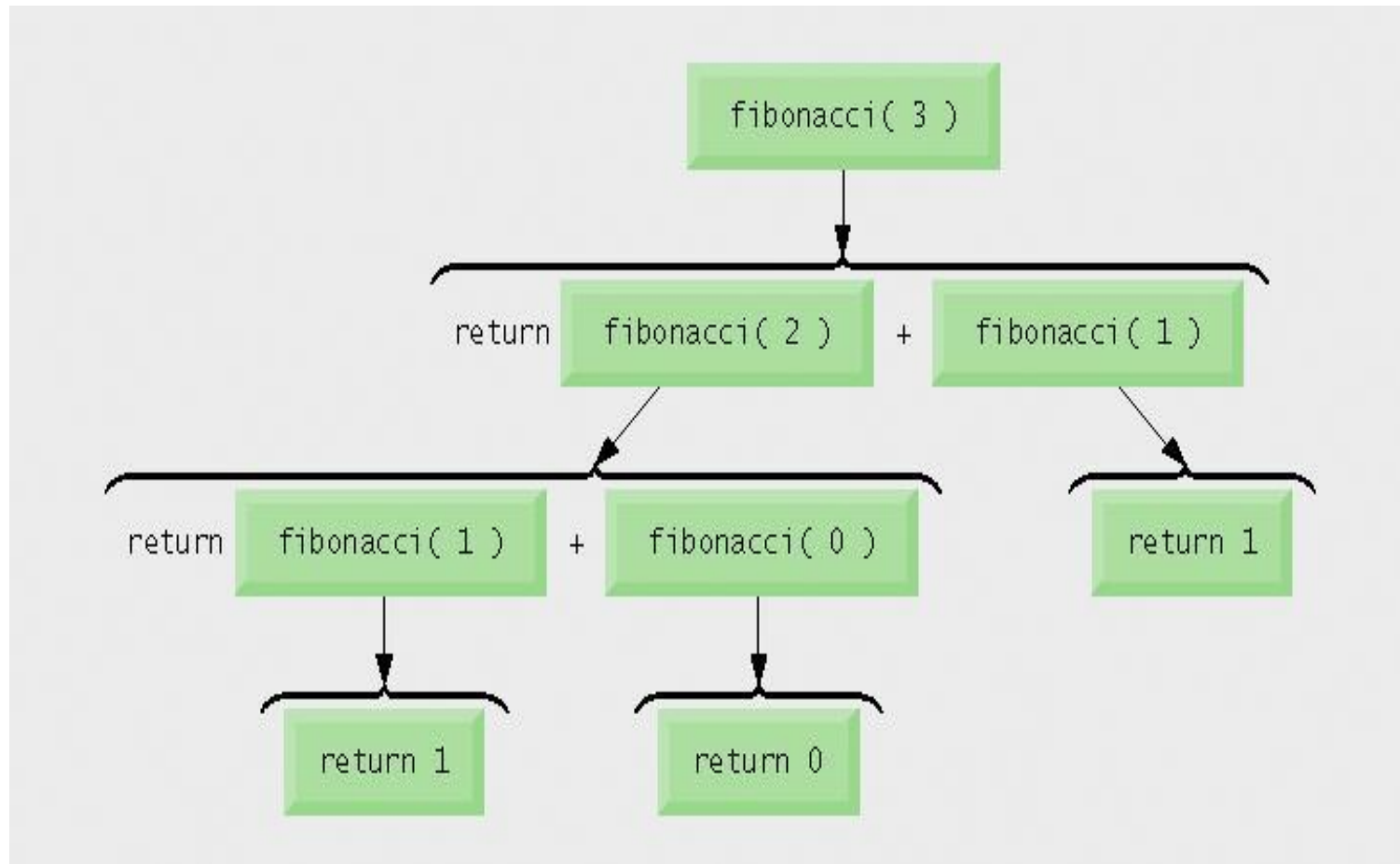


Figura 15.7 | Conjunto de chamadas recursivas para `fibonacci(3)`.

Dica de desempenho 15.1

Evite programas recursivos no estilo Fibonacci, porque resultam em uma ‘explosão’ exponencial de chamadas de método.



15.5 Recursão e a pilha de chamadas do método

- **Pilha de chamadas de método utilizadas para monitorar chamadas ao método e variáveis locais dentro de uma chamada de método.**
- **Assim como ocorre com a programação não-recursiva, chamadas de métodos recursivos são colocadas na parte superior da pilha das chamadas de método.**
- **À medida que retornam as chamadas ao método recursivo, seus registros de ativação são retirados da pilha e as chamadas recursivas prévias continuam a executar.**
- **Método atual em execução sempre é o método cujo registro de ativação está na parte superior da pilha.**



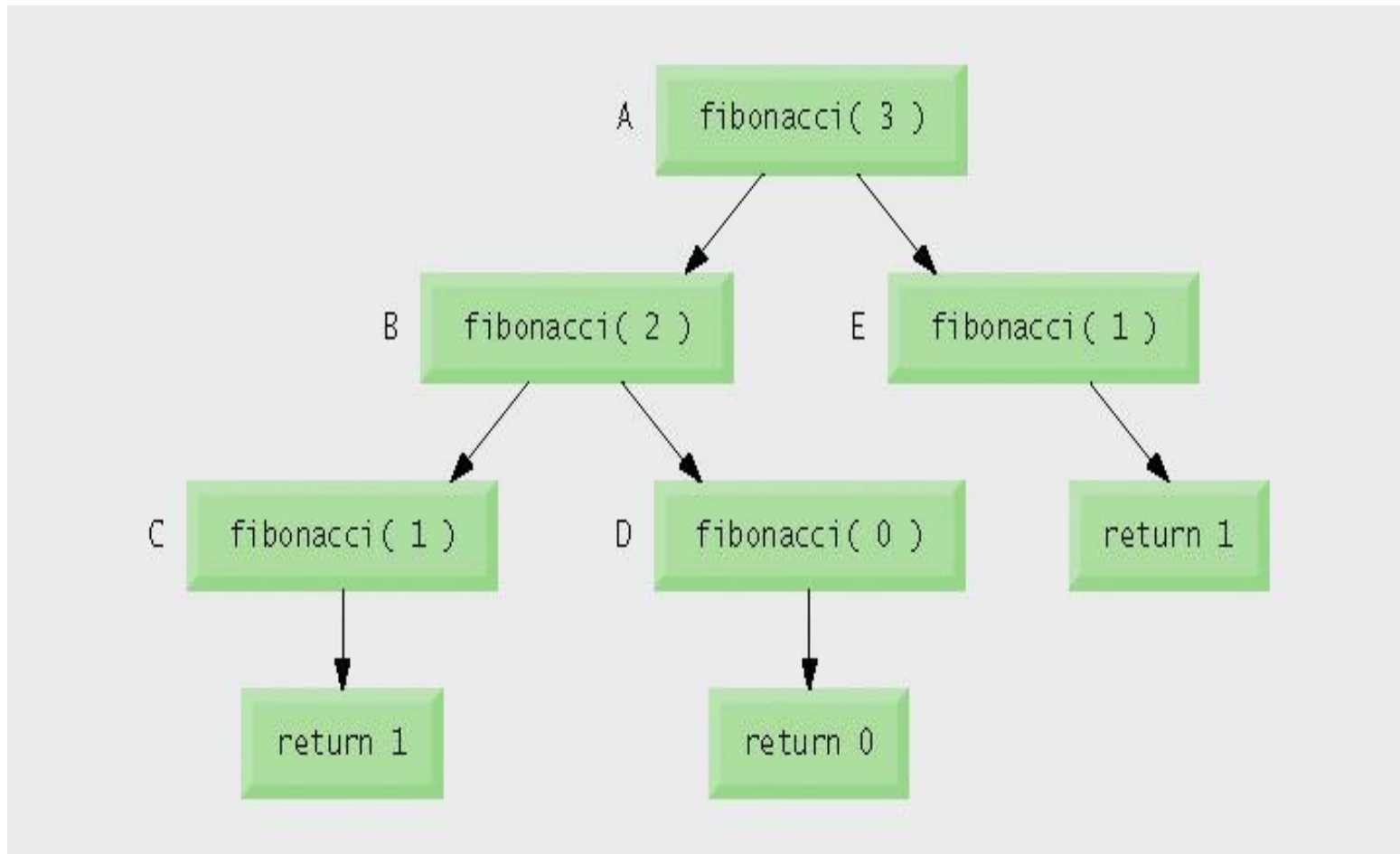


Figura 15.8 | Chamadas do método feitas dentro da chamada `fibonacci(3)`.

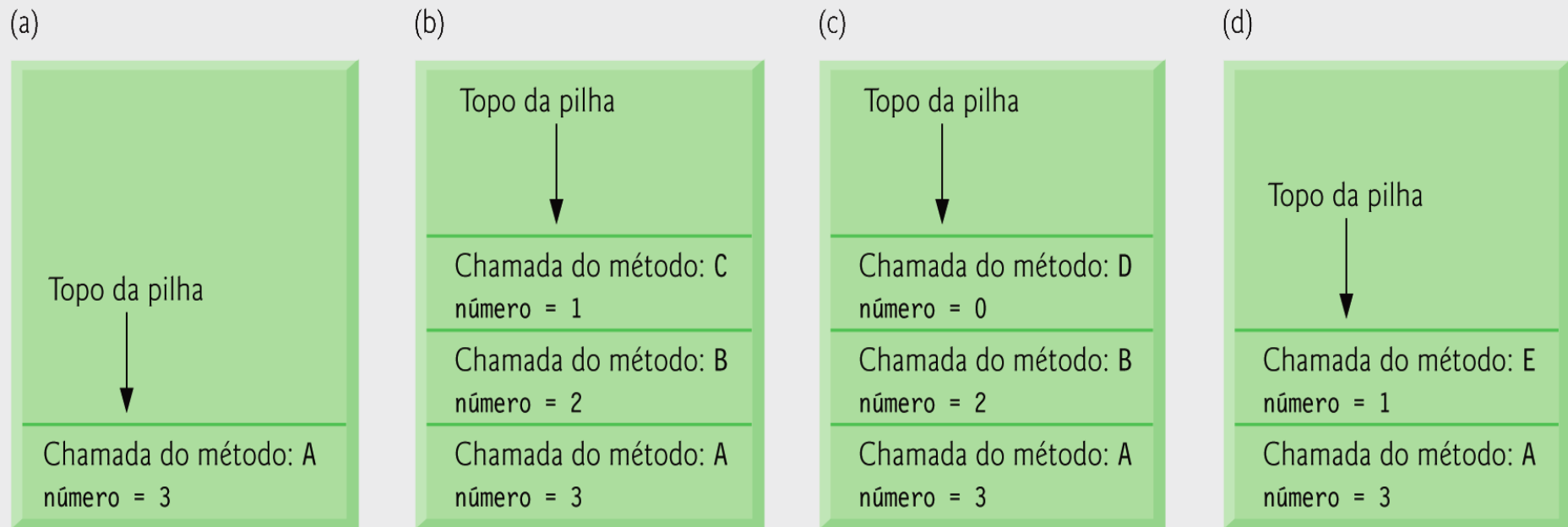


Figura 15.9 | Chamadas do método na pilha de execução do programa.

15.6 Recursão *versus* iteração

- Qualquer problema que possa ser resolvido de modo recursivo também pode ser resolvido iterativamente.
- Tanto a iteração como a recursão utilizam uma instrução de controle.
 - A iteração utiliza uma instrução de repetição.
 - A recursão utiliza uma instrução de seleção.
- Iteração e recursão envolvem um teste de terminação.
 - A iteração termina quando a condição de continuação do loop falha.
 - A recursão termina quando um caso básico é alcançado.
- A recursão pode demandar muito tempo de processador e espaço de memória, mas normalmente fornece uma solução mais intuitiva.



Observação de engenharia de software 15.1

Qualquer problema que pode ser resolvido de modo recursivo também pode ser resolvido iterativamente (não recursivamente). Uma abordagem recursiva em geral é preferida sobre uma abordagem iterativa quando a abordagem recursiva espelha mais naturalmente o problema e resulta em um programa mais fácil de entender e depurar. Uma abordagem recursiva pode ser freqüentemente implementada com menos linhas de código. Outra razão de escolher uma abordagem recursiva é que uma iterativa talvez não seja aparente.



Resumo

Fatorial

calculator.java

```
1 // Fig. 15.10: FactorialCalculator.java
2 // Método fatorial iterativo.
3
4 public class FactorialCalculator
5 {
6     // declaração recursiva de método fatorial
7     public long factorial( long number )
8     {
9         long result = 1;
10
11         // declaração iterativa de método fatorial
12         for ( long i = number; i >= 1; i-- )
13             result *= i;
14
15         return result;
16     } // fim do método factorial
17
18     // gera saída de fatoriais para valores 0-10
19     public void displayFactorials()
20     {
21         // calcula os fatoriais de 0 a 10
22         for ( int counter = 0; counter <= 10; counter++ )
23             System.out.printf( "%d! = %d\n", counter, factorial( counter ) );
24     } // fim do método displayFactorials
25 } // fim da classe FactorialCalculator
```

Solução iterativa utiliza a repetição controlada por contador



Resumo

FactorialTest.java

```
1 // Fig. 15.11: FactorialTest.java
2 // Testando método factorial iterativo.
3
4 public class FactorialTest
5 {
6     // calcula fatoriais de 0-10
7     public static void main( String args[] )
8     {
9         FactorialCalculator factorialCalculator = new FactorialCalculator();
10        factorialCalculator.displayFactorials();
11    } // fim de main
12 } // fim da classe FactorialTest
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```



Dica de desempenho 15.2

Evite utilizar a recursão em situações que requerem alto desempenho. Chamadas recursivas levam tempo e consomem memória adicional.



Erro comum de programação 15.2

Ter acidentalmente um método não recursivo chamando a si próprio seja direta ou indiretamente por outro método pode causar recursão infinita.



15.7 Permutações de string

- **Permutações de uma string de texto** – todas as diferentes strings que podem ser criadas reorganizando os caracteres da string original.
- **As palavras criadas a partir das permutações são conhecidas como anagramas.**
- **Solução recursiva:** Remover um dos caracteres, localizar permutações dos caracteres remanescentes (caso recursivo), combina permutações com o caractere que foi removido.
- **Caso básico:** Localizar permutações para apenas um caractere – o próprio caractere é a única permutação.
- **Qualquer string fornece $n!$ permutações para n caracteres.**



Resumo

Permutation.java

```

1 // Fig. 15.12: Permutation.java
2 // Método recursivo para localizar todas as permutações de uma String.
3
4 public class Permutation
5 {
6     // declaração recursiva do método permuteString
7     private void permuteString(
8         String beginningString, String endingString )
9     {
10         // caso básico: se string a permutar tiver com
11         // 1, exibe apenas essa string concatenada com
12         if ( endingString.length() <= 1 )
13             System.out.println( beginningString + endingString );
14         else // passo de recursão: permuta endingString
15         {
16             // para cada caractere em endingString
17             for ( int i = 0; i < endingString.length(); i++ )
18             {
19                 try
20                 {
21                     // cria nova string para permutar eliminando
22                     // caractere no índice i
23                     String newString = endingString.substring(
24                         0, i ) + endingString.substring(
25                         i + 1, endingString.length() );
26                 }
27                 catch ( StringIndexOutOfBoundsException e )
28                 {
29                     // não deve ocorrer, mas...
30                 }
31             }
32         }
33     }
34 }

```

Caso básico: Combinar caracteres removidos (beginningString) com endingString, que é somente um caractere

Remove um caractere; localizaremos permutações para os caracteres remanescentes



```
26 // chamada recursiva com uma nova string a ser permutada
27 // e uma string inicial a ser concatenada,
28 // inclui o caractere no índice i
29 permuteString( beginningString +
30     endingString.charAt( i ), newString );
31 } // fim do try
32 catch ( StringIndexOutOfBoundsException exception )
33 {
34     exception.printStackTrace();
35 } // fim do catch
36 } // fim do for
37 } // fim do else
38 } // fim do método permuteString
39 } // fim da classe Permutation
```

Chamada recursiva: localizar permutações para os caracteres remanescentes e, então, reanexar os caracteres removidos

.java

(2 de 2)



Resumo

PermutationTest
.java

(1 de 2)

```
1 // Fig. 15.13: PermutationTest.java
2 // Testando o método recursivo para permutar strings.
3 import java.util.Scanner;
4
5 public class PermutationTest
6 {
7     public static void main( String args[] )
8     {
9         Scanner scanner = new Scanner( System.in );
10        Permutation permutationObject = new Permutation();
11
12        System.out.print( "Enter a string: " );
13        String input = scanner.nextLine(); // recupera String a permutar
14
15        // permuta String
16        permutationObject.permuteString( "", input );
17    } // fim de main
18 } // fim da classe PermutationTest
```

Chamada inicial ao método recursivo;
ainda não há caracteres removidos,
assim o primeiro argumento é ""



Resumo

PermutationTest

.java

(2 de 2)



15.8 Torres de Hanói

- **Problema clássico:** Sacerdotes no Extremo Oriente estão tentando mover uma pilha de discos de um pino para outro. Um dos discos deve ser movido em um determinado momento; em nenhum momento um disco maior pode ser posicionado acima de um disco menor.
- **Solução recursiva:**
 - Mova os $n - 1$ discos do pino 1 para o pino 2, utilizando o pino 3 como área de armazenamento temporário.
 - Mova o último disco (o maior) do pino 1 para o pino 3.
 - Mova os $n - 1$ discos do pino 2 para o pino 3, utilizando o pino 1 como área de armazenamento temporário.
- **Caso básico:** Quando somente um disco precisa ser movido, nenhuma área de armazenamento temporário é necessária; o disco é simplesmente movido.



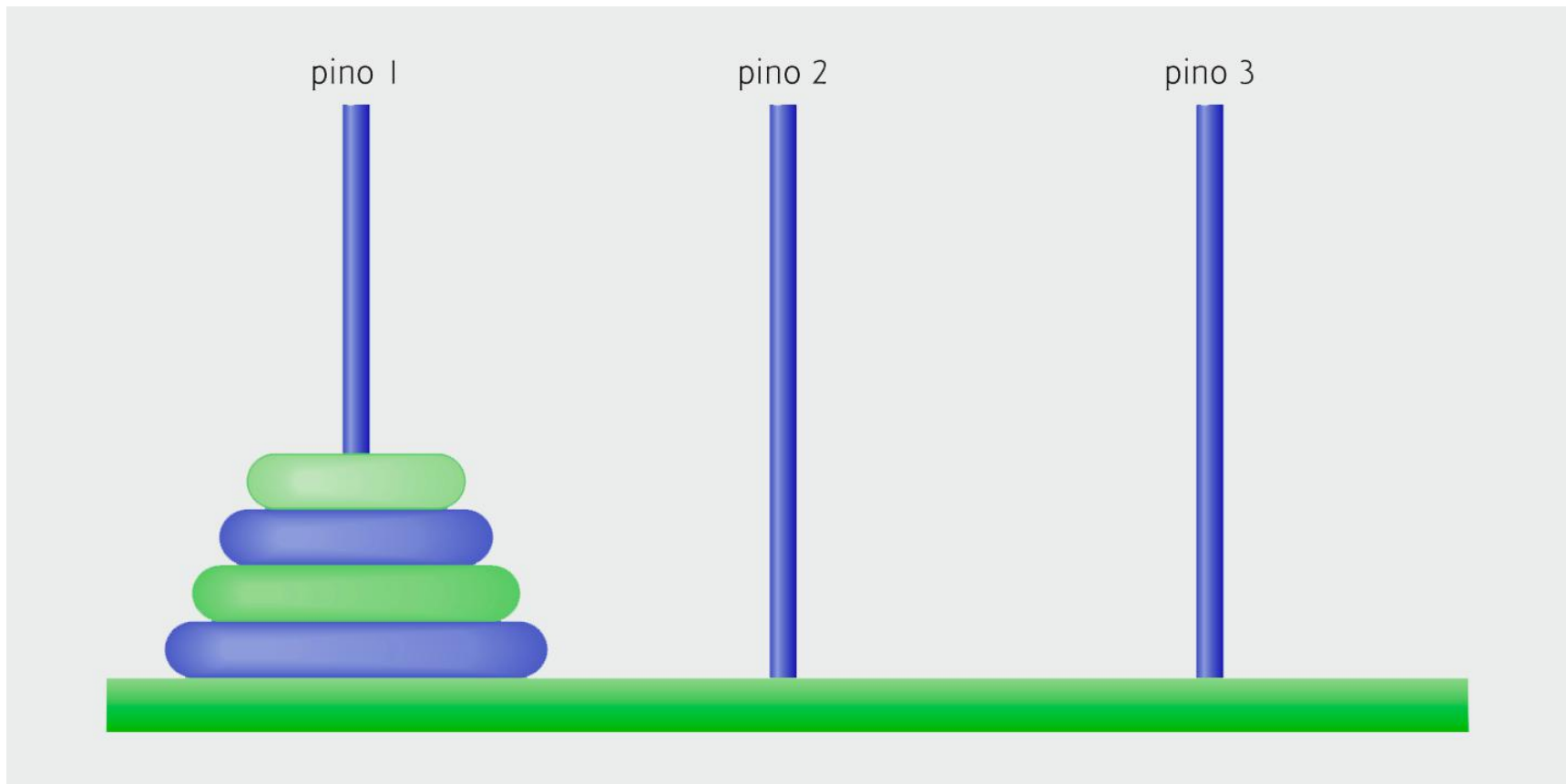


Figura 15.14 | Torres de Hanói para o caso com quatro discos.

Resumo

TowersOfHanoi.java

(1 de 2)

```
1 // Fig. 15.15: TowersOfHanoi.java
2 // O programa resolve o problema torres de Hanói, e
3 // demonstrata a recursão.
4
5 public class TowersOfHanoi
6 {
7     int numDisks; // número de discos a serem movidos
8
9     public TowersOfHanoi( int disks )
10    {
11        numDisks = disks;
12    } // fim do construtor TowersOfHanoi
13
14    // move recursivamente os discos pelas torres
15    public void solveTowers( int disks, int sourcePeg, int destinationPeg,
16        int tempPeg )
17    {
18        // caso básico - apenas um disco a ser movido
19        if ( disks == 1 )
20        {
21            System.out.printf( "\n%d --> %d", sourcePeg, destinationPeg );
22            return;
23        } // fim do if
24
```

Caso básico: simplesmente
exibe o movimento



Resumo

```
25 // passo de recursão -- move o disco para tempPeg, e depois para destinationPeg
26 // move ( disks - 1 ) discos de sourcePeg para tempPeg recursivamente
27 solveTowers( disks - 1, sourcePeg, tempPeg, destinationPeg );
28
29 // move o último disco de sourcePeg para destinationPeg
30 System.out.printf( "\n%d --> %d", sourcePeg, destinationPeg );
31
32 // move ( disks - 1 ) discos de tempPeg para destinationPeg
33 solveTowers( disks - 1, tempPeg, destinationPeg, sourcePeg );
34 } // fim do método solveTowers
35 } // fim da classe TowersOfHanoi
```

Usa o compartimento 3 como a área de armazenamento temporário

Usa o compartimento 1 como a área de armazenamento temporário

Move $n-1$ discos do compartimento 1 para o compartimento 2

Move o último disco no compartimento 1 para o compartimento 3

Move $n-1$ discos do compartimento 2 para o compartimento 3



Resumo

TowersOfHanoiTest
 .java

```

1 // Fig. 15.16: TowersOfHanoiTest.java
2 // Testa a solução do problema Torres de Hanói.
3
4 public class TowersOfHanoiTest
5 {
6     public static void main( String args[] )
7     {
8         int startPeg = 1;    // valor 1 utilizado para indicar startPeg na saída
9         int endPeg = 3;      // valor 3 utilizado para indicar endPeg na saída
10        int tempPeg = 2;     // valor 2 utilizado para indicar tempPeg na saída
11        int totalDisks = 3;  // número de discos
12        TowersOfHanoi towersOfHanoi = new TowersOfHanoi( totalDisks );
13
14        // chamada não-recursiva inicial: move todos os discos.
15        towersOfHanoi.solveTowers( totalDisks, startPeg, endPeg,
16    } // fim de main
17 } // fim da classe TowersOfHanoiTest
  
```

Faz a chamada inicial ao método
 recursivo

```

1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
  
```



15.9 Fractais

- **Um fractal – uma figura geométrica que freqüentemente pode ser gerada a partir de um padrão repetido recursivamente por um número infinito de vezes.**
- **Padrão aplicado a cada segmento da figura original.**
- **Benoit Mandelbrot introduziu o termo ‘fractal’, juntamente com especificidades de como os fractais são criados e de suas aplicações práticas.**
 - **Ajuda a melhor entender os padrões na natureza, o corpo humano e o universo.**
 - **Forma de arte popular.**



15.9 Fractais (*Continuação*)

- **Propriedade auto-semelhante** – os fractais têm essa propriedade na eventualidade de que, quando subdivididos em partes, cada um se assemelhe a uma cópia de tamanho reduzido do todo.
- **Se a parte for uma cópia exata do original, dizemos que o fractal é estritamente auto-semelhante.**
- **Toda vez que um padrão é aplicado, dizemos que o fractal está em um novo nível ou profundidade.**
- **Exemplos de fractais: Curva de KOch, Floco de neve de Koch.**





Figura 15.17 | Fractal Curva de Koch.

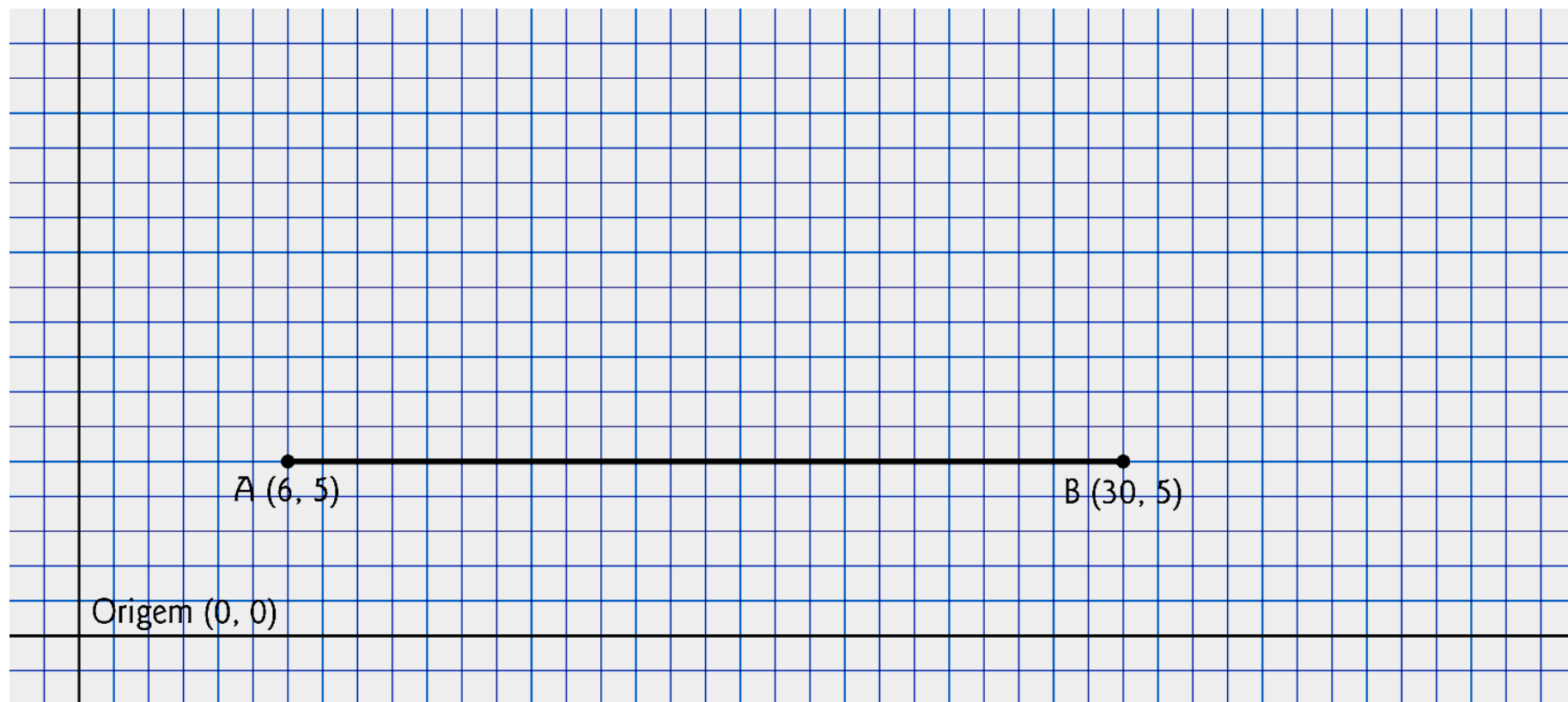


Figura 15.18 | 'Fractal de Lo' no nível 0.

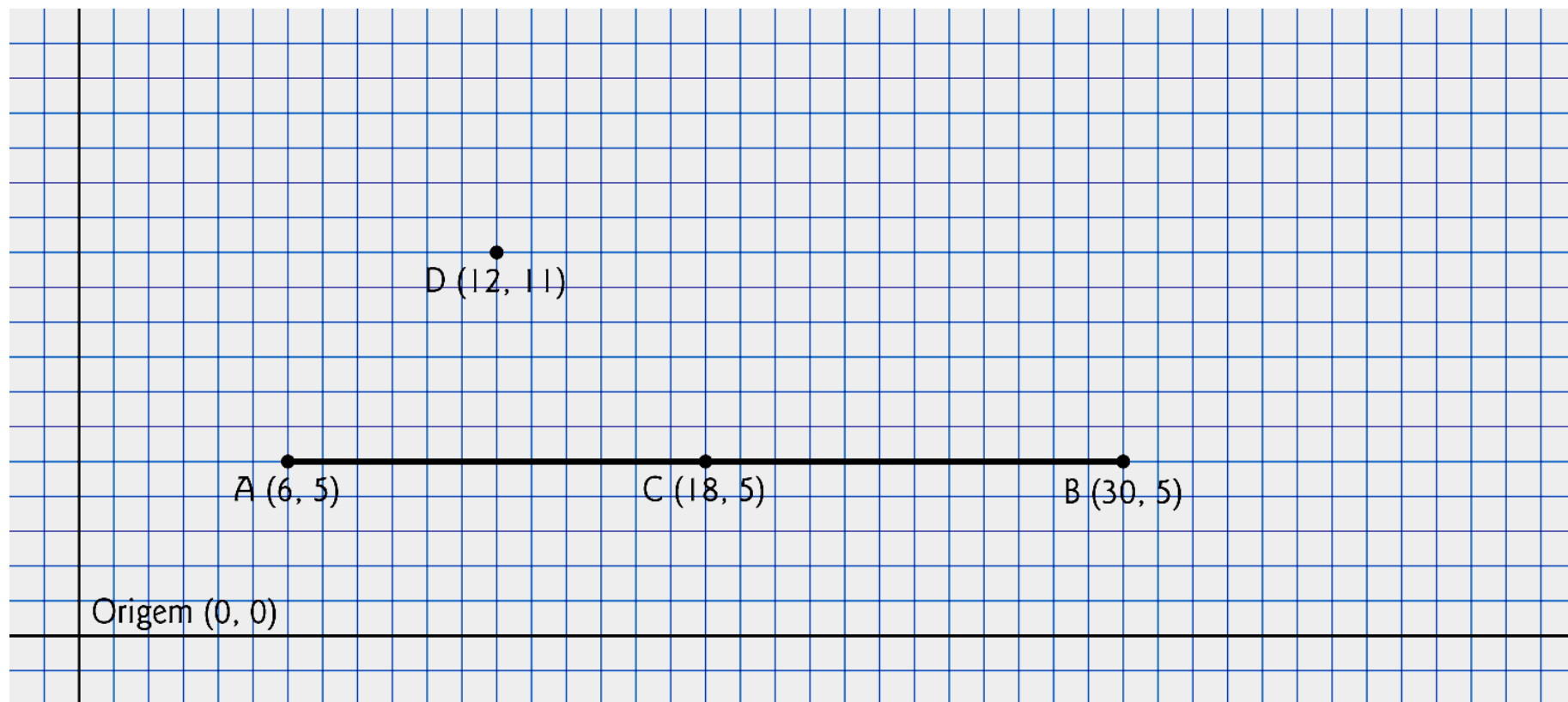


Figura 15.19 | Determinando os pontos C e D para o nível 1 do 'Fractal de Lo'.

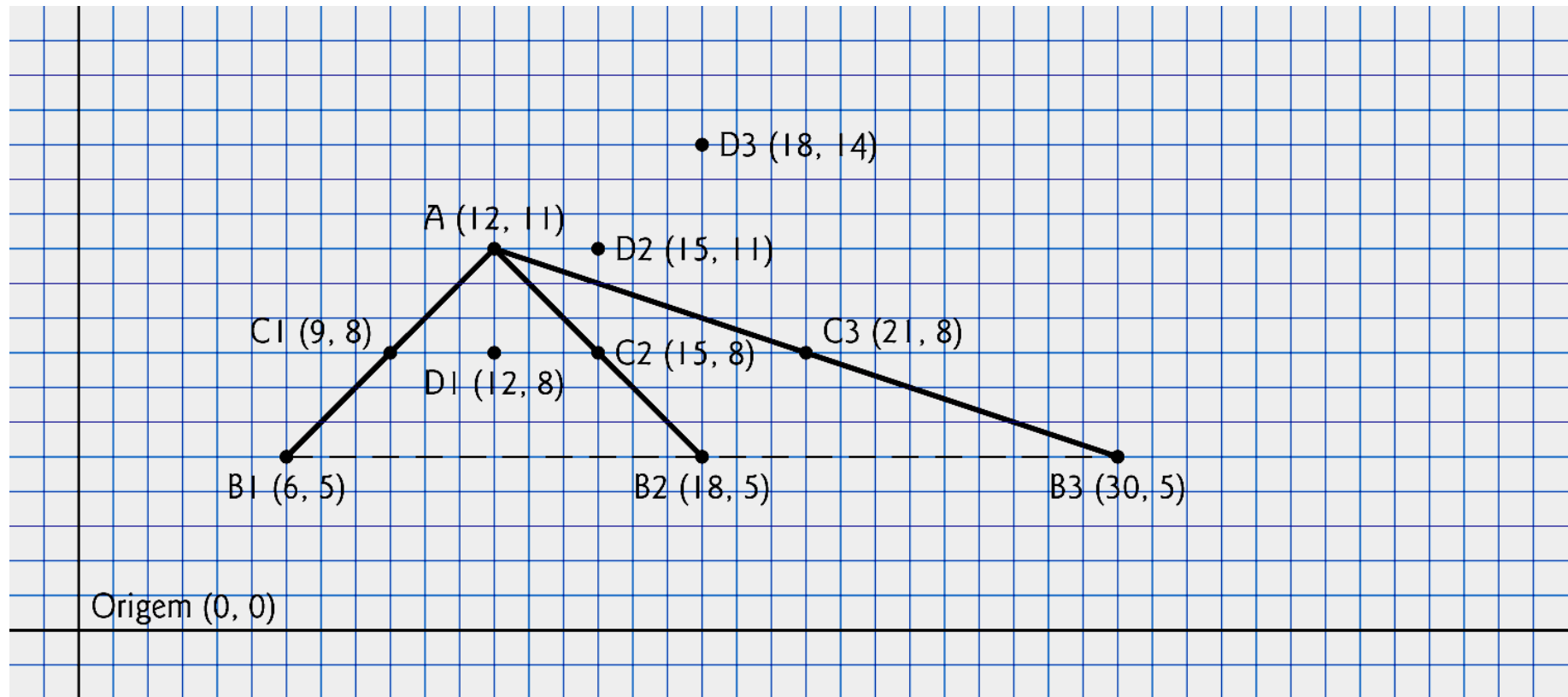


Figura 15.20 | 'Fractal de Lo' no nível 1, com pontos C e D determinados para o nível 2. [Nota: O fractal no nível 0 está incluído como uma linha tracejada como um lembrete de onde a linha foi localizada em relação ao fractal atual.]

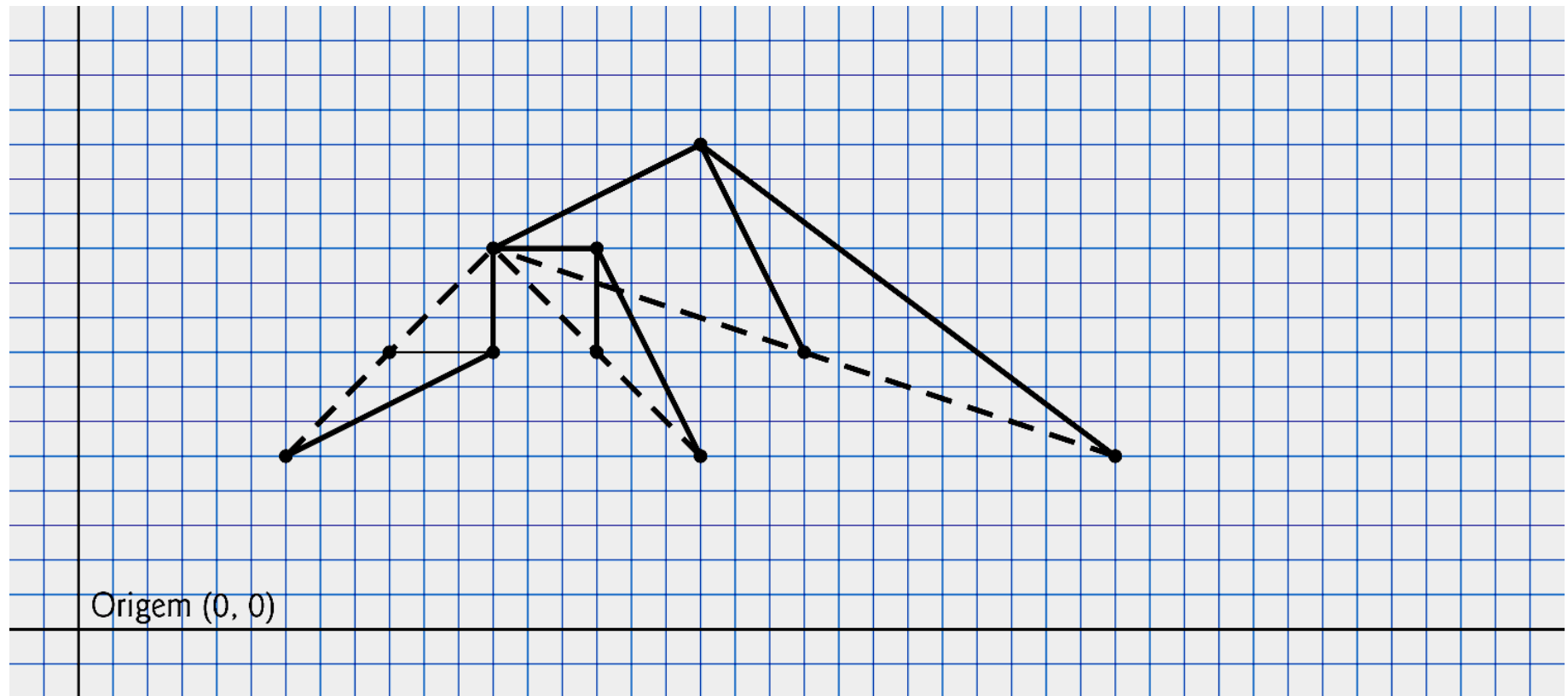


Figura 15.21 | 'Fractal de Lo' no nível 2, com linhas tracejadas do nível 1 fornecido.

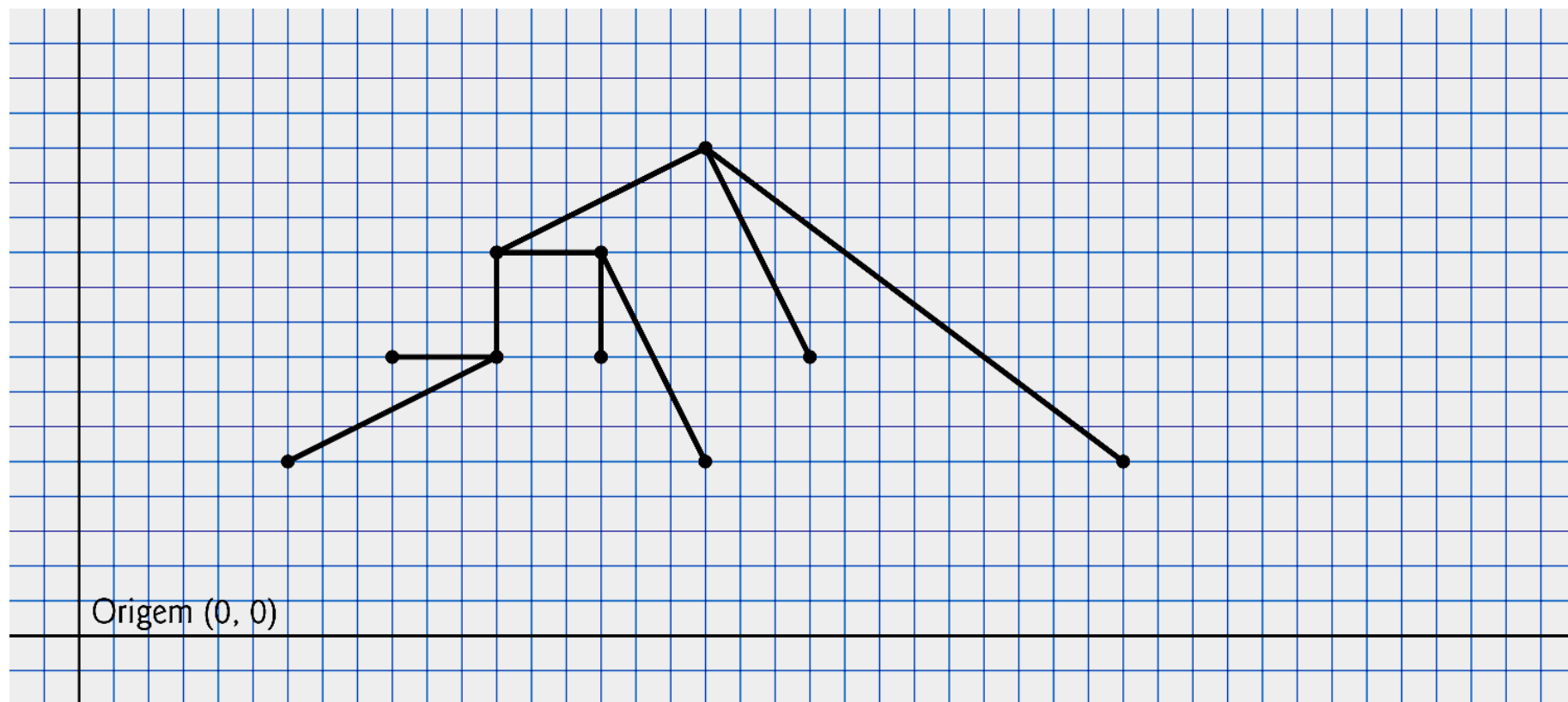


Figura 15.22 | 'Fractal de Lo' no nível 2.

Resumo

Fractal.java

(1 de 5)

```
1 // Fig. 15.23: Fractal.java
2 // Demonstra interface com o usuário para desenhar um fractal.
3 import java.awt.Color;
4 import java.awt.FlowLayout;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import javax.swing.JFrame;
8 import javax.swing.JButton;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11 import javax.swing.JColorChooser;
12
13 public class Fractal extends JFrame
14 {
15     private final int WIDTH = 400; // define a largura da GUI
16     private final int HEIGHT = 480; // define a altura da GUI
17     private final int MIN_LEVEL = 0;
18     private Color color = Color.BLUE;
19
20     private JButton changeColorJButton, increaseLevelJButton,
21         decreaseLevelJButton;
22     private JLabel levelJLabel;
23     private FractalJPanel drawSpace;
24     private JPanel mainJPanel, controlJPanel;
25
26     // configura a GUI
27     public Fractal()
28     {
29         super( "Fractal" );
30     }
```



Resumo

Fractal.java

(2 de 5)

```

31 // configura o painel de controle
32 controlJPanel = new JPanel();
33 controlJPanel.setLayout( new FlowLayout() );
34
35 // configura o botão de cor e registra o ouvinte
36 changeColorJButton = new JButton( "Color" );
37 controlJPanel.add( changeColorJButton );
38 changeColorJButton.addActionListener(
39     new ActionListener() // classe interna anônima
40     {
41         // processa o evento changeColorJButton
42         public void actionPerformed((ActionEvent event) )
43         {
44             color = JColorChooser.showDialog(
45                 Fractal.this, "Choose a color", color );
46
47             // configura a cor padrão, se nenhuma cor for retornada
48             if ( color == null )
49                 color = Color.BLUE;
50
51             drawSpace.setColor( color );
52         } // fim do método actionPerformed
53     } // fim da classe interna anônima
54 ); // fim de addActionListener
55

```



Resumo

Fractal.java

```

56 // configura o botão decrease level para adicionar painel de controle
57 // registra o ouvinte
58 decreaseLevelJButton = new JButton( "Decrease Level" );
59 controlJPanel.add( decreaseLevelJButton );
60 decreaseLevelJButton.addActionListener(
61     new ActionListener() // classe interna anônima
62     {
63         // processa o evento decreaseLevelJButton
64         public void actionPerformed((ActionEvent event)
65         {
66             int level = drawSpace.getLevel();
67             level--; // diminui o nível por um
68
69             // modifica o nível se possível
70             if ( level >= MIN_LEVEL )
71             {
72                 levelJLabel.setText( "Level: " + level );
73                 drawSpace.setLevel( level );
74                 repaint();
75             } // fim de if
76         } // fim do método actionPerformed
77     } // fim da classe interna anônima
78 ); // fim do addActionListener
79

```

Recupera o nível atual

Diminui o nível

Configura um novo nível

Redesenha um fractal até o novo nível



Resumo

Fractal.java

```

80 // configura o botão increase level para adicionar painel de controle
81 // e registra o ouvinte
82 increaseLevelJButton = new JButton( "Increase Level" );
83 controlJPanel.add( increaseLevelJButton );
84 increaseLevelJButton.addActionListener(
85     new ActionListener() // classe interna anônima
86     {
87         // processa o ouvinte increaseLevelJButton
88         public void actionPerformed((ActionEvent event)
89         {
90             int level = drawSpace.getLevel();
91             level++; // aumenta o nível por um
92
93             // modifica o nível se possível
94             if ( level >= MIN_LEVEL )
95             {
96                 levelJLabel.setText( "Level: " + level );
97                 drawSpace.setLevel( level );
98                 repaint();
99             } // fim do if
100         } // fim do método actionPerformed
101     } // fim da classe interna anônima
102 ); // fim do addActionListener
103
104 // configura levelJLabel para adicionar ao controlJPanel
105 levelJLabel = new JLabel( "Level: 0" );
106 controlJPanel.add( levelJLabel );
107

```

Recupera o nível atual

Aumenta o nível

Configura um novo nível

Redesenha um fractal até o novo nível



Resumo

Fractal.java

(5 de 5)

```
108 drawSpace = new FractalJPanel( 0 );
109
110 // cria mainJPanel para conter controlJPanel e drawSpace
111 mainJPanel = new JPanel();
112 mainJPanel.add( controlJPanel );
113 mainJPanel.add( drawSpace );
114
115 add( mainJPanel ); // adiciona JPanel ao JFrame
116
117 setSize( WIDTH, HEIGHT ); // configura tamanho do JFrame
118 setVisible( true ); // exibe o JFrame
119 } // fim do construtor Fractal
120
121 public static void main( String args[] )
122 {
123     Fractal demo = new Fractal();
124     demo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
125 } // fim de main
126} // fim da classe Fractal
```



Resumo

FractalJPanel.java

(1 de 6)

```
1 // Fig. 15.24: FractalJPanel.java
2 // FractalJPanel demonstrata desenho recursivo de um fractal.
3 import java.awt.Graphics;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import javax.swing.JPanel;
7
8 public class FractalJPanel extends JPanel
9 {
10     private Color color; // armazena cor utilizada para desenhar o fractal
11     private int level;   // armazena o nível atual do fractal
12
13     private final int WIDTH = 400; // define largura do JPanel
14     private final int HEIGHT = 400; // define altura do JPanel
15
16     // configura o nível do fractal inicial com o valor especificado
17     // e configura as especificações do JPanel
18     public FractalJPanel( int currentLevel )
19     {
20         color = Color.BLUE; // inicializa cor do desenho como azul
21         level = currentLevel; // configura o nível do fractal inicial
22         setBackground( Color.WHITE );
23         setPreferredSize( new Dimension( WIDTH, HEIGHT ) );
24     } // fim do construtor FractalJPanel
25
```



Resumo

```

26 // desenha o fractal recursivamente
27 public void drawFractal( int level, int xA, int yA, int xB,
28 int yB, Graphics g )
29 {
30 // caso básico: desenha uma linha conectando dois pontos
31 if ( level == 0 )
32 g.drawLine( xA, yA, xB, yB );
33 else // passo de recursão: determina novos pontos
34 {
35 // calcula ponto intermediário entre (xA, yA) e (xB, yB)
36 int xC = ( xA + xB ) / 2;
37 int yC = ( yA + yB ) / 2;
38
39 // calcula o quarto ponto (xD, yD) que forma um
40 // triângulo isósceles reto entre (xA, yA) e (xC, yC)
41 // onde o ângulo reto está em (xD, yD)
42 int xD = xA + ( xC - xA ) / 2 - ( yC - yA ) / 2;
43 int yD = yA + ( yC - yA ) / 2 + ( xC - xA ) / 2;
44
45 // desenha recursivamente o Fractal
46 drawFractal( level - 1, xD, yD, xA, yA, g );
47 drawFractal( level - 1, xD, yD, xC, yC, g );
48 drawFractal( level - 1, xD, yD, xB, yB, g );
49 } // fim de else
50 } // fim do método drawFractal
51

```

Coordenadas do primeiro ponto

Coordenadas do segundo ponto para a
linha, o padrão não é aplicado

Caso básico: Simplesmente desenha a
linha, o padrão não é aplicado

(2 de 6)

Calcula o ponto intermediário

Passo de recursão: Aplica o padrão de fractal

Calcula o ponto para formar o
triângulo direito

Aplica o padrão a três novas
linhas



Resumo

JPanel.java

Faz a primeira chamada ao método recursivo sempre que janela é redesenhada

(3 de 6)

```
52 // inicia o desenho do fractal
53 public void paintComponent( Graphics g )
54 {
55     super.paintComponent( g );
56
57     // desenha o padrão do fractal
58     g.setColor( color );
59     drawFractal( level, 100, 90, 290, 200, g );
60 } // fim do método paintComponent
61
62 // configura a cor como c
63 public void setColor( Color c )
64 {
65     color = c;
66 } // fim do método setColor
67
68 // configura o novo nível de recursão
69 public void setLevel( int currentLevel )
70 {
71     level = currentLevel;
72 } // fim do método setLevel
73
```

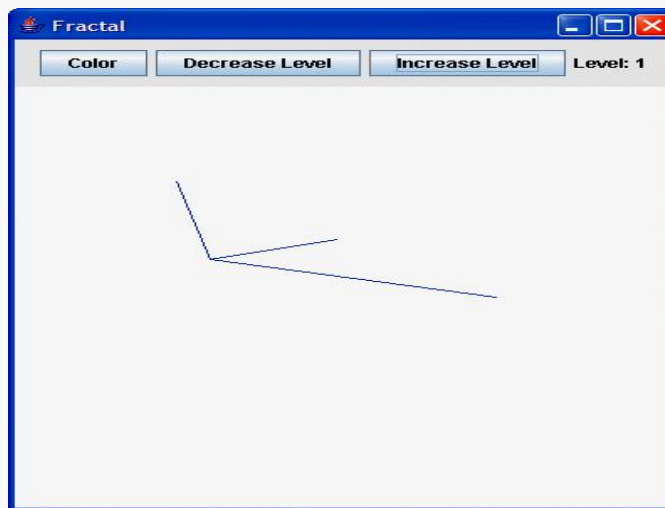
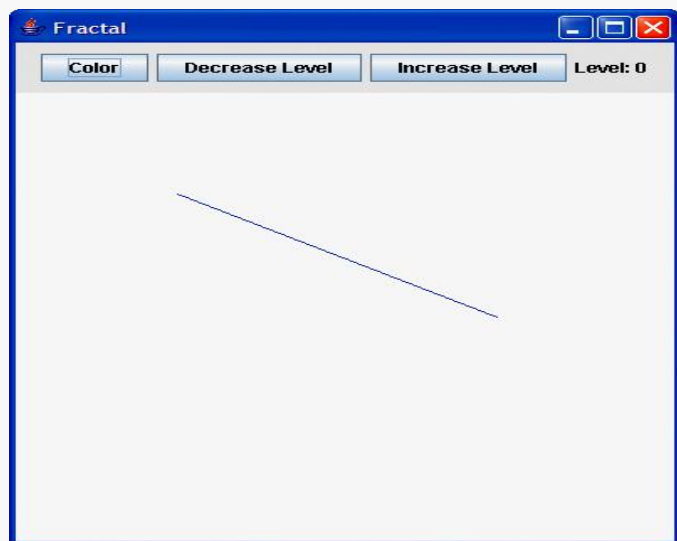


Resumo

FractalJPanel.java

(4 de 6)

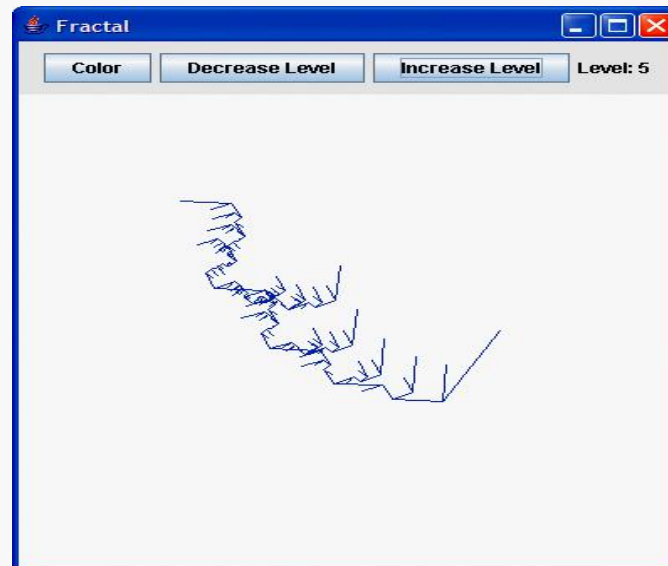
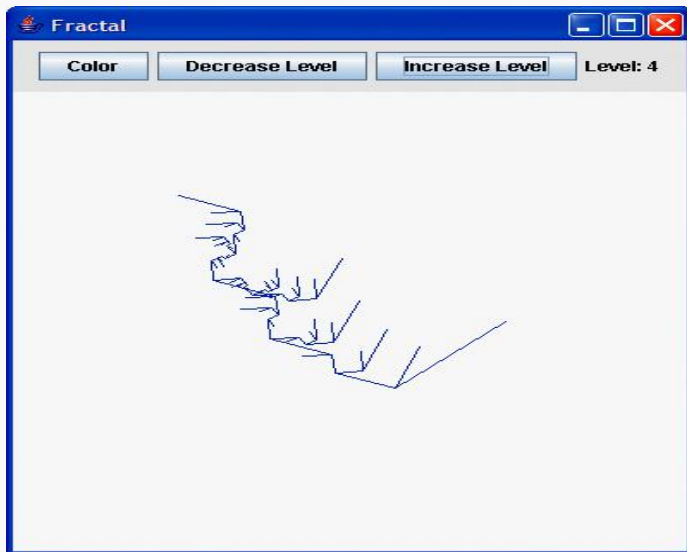
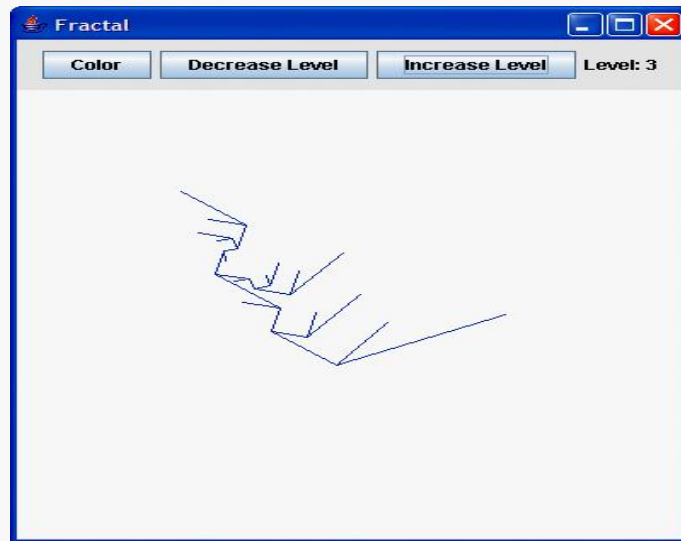
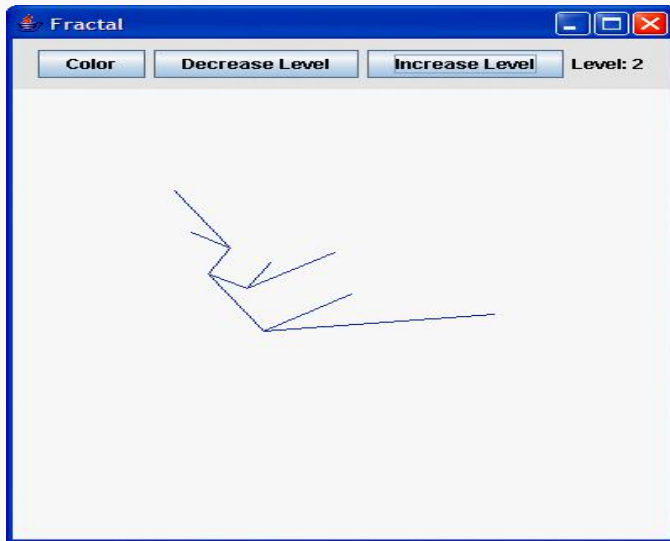
```
74 // retorna o nível de recursão
75 public int getLevel()
76 {
77     return level;
78 } // fim do método getLevel
79 } // fim da classe FractalJPanel
```



Resumo

FractalJPanel.java

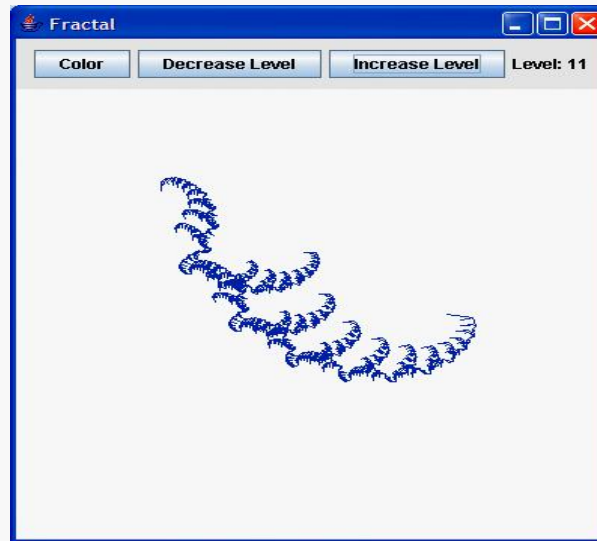
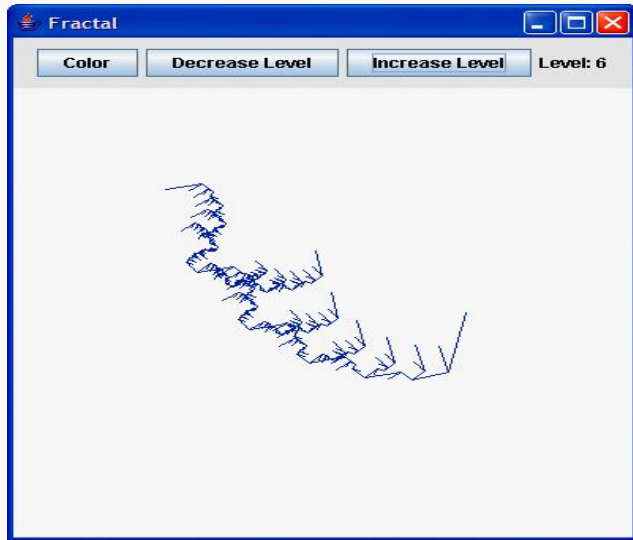
(5 de 6)



Resumo

FractalJPanel.java

(6 de 6)



15.10 Retorno recursivo

- **Reversão recursiva** – processo de utilização da recursão para retornar ao ponto de decisão anterior.
- **Se um conjunto de chamadas recursivas não resultar em uma solução, o programa volta ao ponto de decisão e toma uma decisão diferente, resultando freqüentemente em outro conjunto de chamadas recursivas.**
- **Exemplos:**
 - **Problema do labirinto.**
 - **Problema das Oito Rainhas.**

