

# 12

## Imagens gráficas e Java 2D™



# OBJETIVOS

- Neste capítulo, você aprenderá:
- Como entender contextos gráficos e objetos gráficos.
- Como entender e ser capaz de manipular cores.
- Como entender e ser capaz de manipular fontes.
- Como utilizar métodos da classe `Graphics` para desenhar linhas, retângulos, retângulos com cantos arredondados, retângulos tridimensionais, ovais, arcos e polígonos.
- Como utilizar métodos da classe `Graphics2D` da API do Java 2D para desenhar linhas, retângulos, retângulos com cantos arredondados, ovais, arcos e caminhos gerais.
- Como ser capaz de especificar as características `Paint` e `stroke` de formas exibidas com `Graphics2D`.

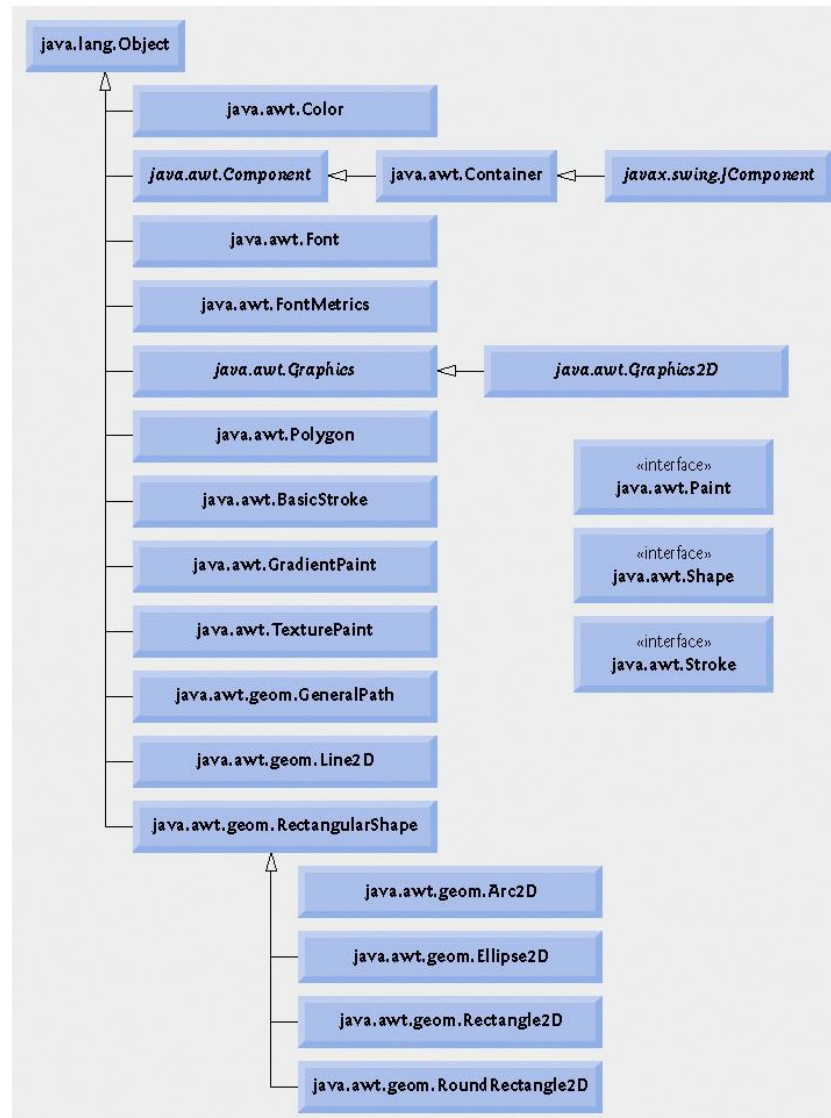


- 12.1**    **Introdução**
- 12.2**    **Contextos gráficos e objetos gráficos**
- 12.3**    **Controle de cor**
- 12.4**    **Controle de fonte**
- 12.5**    **Desenhando linhas, retângulos e ovais**
- 12.6**    **Desenhando arcos**
- 12.7**    **Desenhando polígonos e polilinhas**
- 12.8**    **API do Java 2D**
- 12.9**    **Conclusão**

## 12.1 Introdução

- **O Java contém suporte para imagens gráficas que permite aos programadores aprimorar visualmente as aplicações.**
- **O Java contém várias outras capacidades mais sofisticadas de desenho como parte da API do Java 2D™.**
- **Classes:**
  - **Color**
  - **Font, FontMetrics**
  - **Graphics2D**
  - **Polygon**
  - **BasicStroke**
  - **GradientPaint, TexturePaint**
  - **Classes Java para formas em 2D**



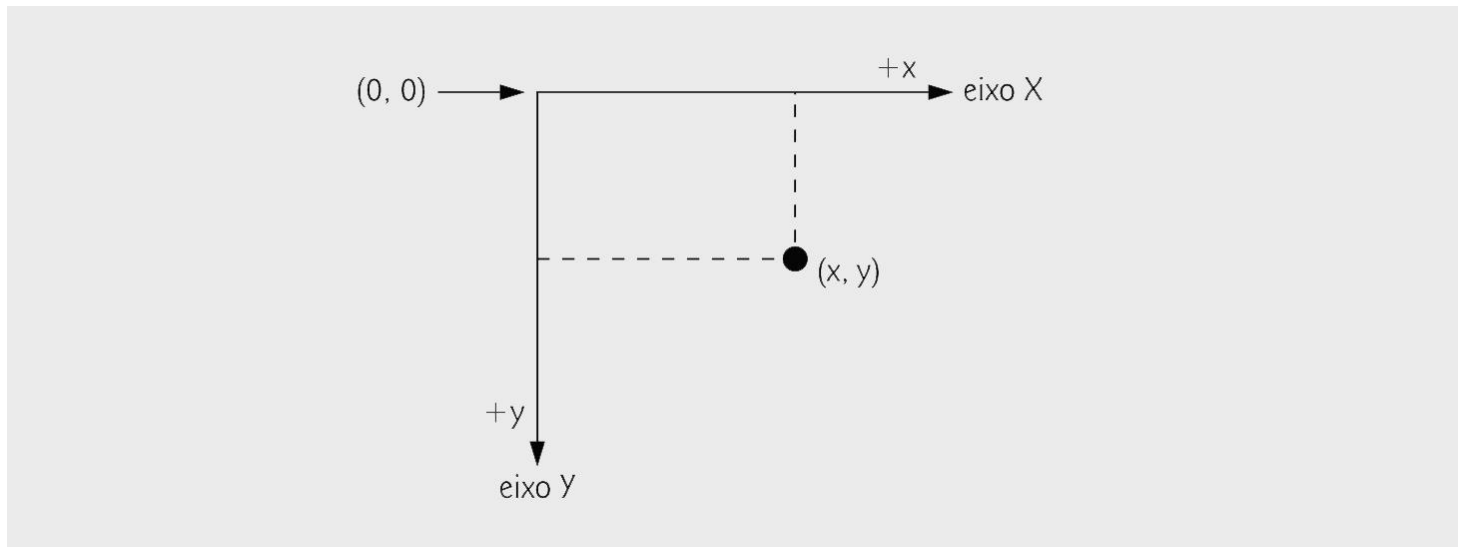


**Figura 12.1** | As classes e interfaces utilizadas neste capítulo são provenientes das capacidades gráficas originais do Java e da API do Java 2DI. [Nota: A classe Object aparece aqui porque é a superclasse da hierarquia de classes do Java.]

## 12.1 Introdução (*Continuação*)

- **Sistema de coordenadas do Java:**
  - O canto superior esquerdo de um componente GUI tem as coordenadas (0, 0).
  - Contém a coordenada *x* (coordenada horizontal) — distância horizontal que vai do lado direito ao lado esquerdo da tela.
  - Contém a coordenada *y* (coordenada vertical) — a distância vertical de baixo para cima na tela.
- **As unidades das coordenadas são medidas em pixels. Um pixel é a menor unidade de exibição de resolução do monitor.**





**Figura 12.2 | Sistema de coordenadas Java. As unidades são medidas em pixels.**

## Dica de portabilidade 12.1

---

**Monitores diferentes têm resoluções diferentes (isto é, a densidade dos pixels varia). Isso pode fazer com que as imagens gráficas pareçam ter diferentes tamanhos em diferentes monitores ou no mesmo monitor com diferentes configurações.**





## 12.2 Contextos gráficos e objetos gráficos

- Um contexto gráfico Java permite desenhar na tela.
- Classe `Graphics`:
  - Gerencia um contexto gráfico e desenha pixels na tela.
  - Uma classe `abstract` – contribui para a portabilidade do Java.
- Método `paintComponent`:
  - Utilizado para desenhar imagens gráficas.
  - Membro da classe `JComponent`, subclasse de `Component`.
  - O objeto `Graphics` passado para `paintComponent` pelo sistema quando um componente Swing leve precisa ser repintado.
  - Se o programador precisar executar `paintComponent`, será feita uma chamada ao método `repaint`.



## 12.3 Controle de cor

- A classe **Color** declara métodos e constantes para manipular cores em um programa Java.
- Cada cor é criada a partir de um componente vermelho, um verde e um azul — valores **RGB**.



Constante	Color	Cor	Valor RGB
<code>public final static Color</code>	<code>RED</code>	vermelho	255, 0, 0
<code>public final static Color</code>	<code>GREEN</code>	verde	0, 255, 0
<code>public final static Color</code>	<code>BLUE</code>	azul	0, 0, 255
<code>public final static Color</code>	<code>ORANGE</code>	laranja	255, 200, 0
<code>public final static Color</code>	<code>PINK</code>	rosa	255, 175, 175
<code>public final static Color</code>	<code>CYAN</code>	ciano	0, 255, 255
<code>public final static Color</code>	<code>MAGENTA</code>	magenta	255, 0, 255
<code>public final static Color</code>	<code>YELLOW</code>	amarelo	255, 255, 0
<code>public final static Color</code>	<code>BLACK</code>	preto	0, 0, 0
<code>public final static Color</code>	<code>WHITE</code>	branco	255, 255, 255
<code>public final static Color</code>	<code>GRAY</code>	cinza	128, 128, 128
<code>public final static Color</code>	<code>LIGHT_GRAY</code>	cinza-claro	192, 192, 192
<code>public final static Color</code>	<code>DARK_GRAY</code>	cinza-escuro	64, 64, 64

**Figura 12.3 | Constantes Color e seus valores de RGB.**

Método	Descrição
<i>Construtores e métodos Color</i>	
<code>public Color( int r, int g, int b )</code>	Cria uma cor com base nos componentes azul, verde, vermelho expressos como valores de ponto flutuante de 0,0 a 1,0.
<code>public Color( float r, float g, float b )</code>	Cria uma cor com base nos componentes vermelho, verde e azul expressos como valores de ponto flutuante de 0,0 a 1,0.
<code>public int getRed()</code>	Retorna um valor entre 0 e 255 representando o conteúdo de vermelho.
<code>public int getGreen()</code>	Retorna um valor entre 0 e 255 representando o conteúdo de verde.
<code>public int getBlue()</code>	Retorna um valor entre 0 e 255 representando o conteúdo de azul.
<i>Métodos Graphics para manipular Colors</i>	
<code>public Color getColor()</code>	Retorna o objeto Color que representa as cores atuais no contexto gráfico.
<code>public void setColor( Color c )</code>	Configura a cor atual para desenho com o contexto gráfico.

**Figura 12.4 | Métodos Color e métodos Graphics relacionados com cor.**

# Resumo

ColorJPanel.java

1 // Fig. 12.5: ColorJPanel.java

2 // Demonstrando Colors.

3 import java.awt.Graphics;

4 import java.awt.Color;

5 import javax.swing.JPanel;

6

7 public class ColorJPanel extends JPanel

8 {

9 // desenha retângulos e strings em cores diferentes

10 public void paintComponent( Graphics g )

11 {

12 super.paintComponent( g ); // chama o paintComponent da superclasse

13

14 this.setBackground( Color.WHITE );

15

16 // configura nova cor de desenho utilizando inteiros

17 g.setColor( new Color( 255, 0, 0 ) );

18 g.fillRect( 15, 25, 100, 20 );

19 g.drawString( "Current RGB: " + g.getColor(), 130, 40 );

20

21 // configura nova cor de desenho utilizando floats

22 g.setColor( new Color( 0.50f, 0.75f, 0.0f ) );

23 g.fillRect( 15, 50, 100, 20 );

24 g.drawString( "Current RGB: " + g.getColor(), 130, 65 );

25

26 // configura nova cor de desenho utilizando objetos static

27 g.setColor( Color.BLUE );

28 g.fillRect( 15, 75, 100, 20 );

29 g.drawString( "Current RGB: " + g.getColor(), 130, 90 );

30

O método paintComponent  
pinta o JPanel

1 de 2)

Configura a cor atual do desenho  
com o método setColor

Desenha um retângulo preenchido  
utilizando a cor atual

Obtém o valor de texto da cor atual

Configura a cor atual do desenho,  
especifica argumentos float para  
o construtor Color

Configura a cor atual do desenho  
utilizando a constante Color



# Resumo

ColorJPanel.java

(2 de 2)

```
31 // exibe valores individuais de RGB
32 color color = Color.MAGENTA;
33 g.setColor( color );
34 g.fillRect( 15, 100, 100, 20 );
35 g.drawString( "RGB values: " + color.getRed() + ", " +
36             color.getGreen() + ", " + color.getBlue(), 130, 115 );
37 } // fim do método paintComponent
38 } // fim da classe ColorJPanel
```

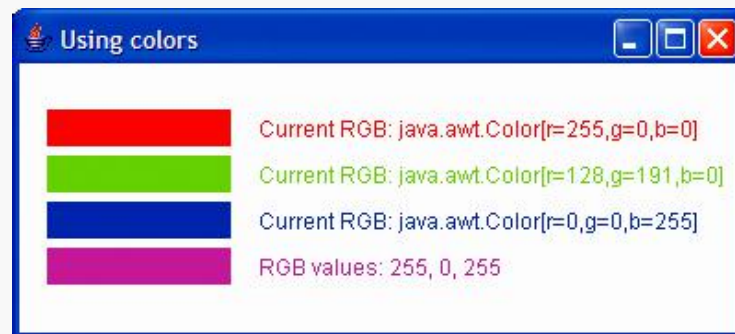
Recuperando os valores RGB com  
os métodos `getRed`, `getGreen` e  
`getBlue`



# Resumo

## ShowColors.java

```
1 // Fig. 12.6: ShowColors.java
2 // Demonstrando Cores.
3 import javax.swing.JFrame;
4
5 public class ShowColors
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria o frame para ColorJPanel
11         JFrame frame = new JFrame( "Using colors" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         ColorJPanel colorJPanel = new ColorJPanel(); // cria ColorJPanel
15         frame.add( colorJPanel ); // adiciona colorJPanel ao frame
16         frame.setSize( 400, 180 ); // configura o tamanho do frame
17         frame.setVisible( true ); // exibe o frame
18     } // fim de main
19 } // fim da classe ShowColors
```



# Observação sobre aparência e comportamento 12.1

---

**Cada um percebe as cores de uma maneira diferente. Escolha suas cores cuidadosamente para assegurar que seu aplicativo é legível. Tente evitar utilizar várias cores diferentes com valores muito próximos.**



# Observação de engenharia de software 12.1

---

**Para alterar as cores, você deve criar um novo objeto `Color` (ou utilizar uma das constantes `Color` pré-declaradas). Como ocorre com objetos `String`, objetos `Color` são imutáveis (não-modificáveis).**

## 12.3 Controle de cor (*Continuação*)

- O componente GUI de **JColorChooser** permite que os usuários do aplicativo selecionem cores.
  - O método **showDialog** cria um objeto **JColorChooser**, anexa-o a uma caixa de diálogo e exibe o diálogo.
  - Diálogo modal.
  - Permite que o usuário selecione uma cor entre uma variedade de swatches (amostras) de cores.
  - Guias: Swatches, HSB e RGB.




# Resumo

ShowColors2JFrame  
.java

(1 de 2)

```
1 // Fig. 12.7: ShowColors2JFrame.java
2 // Escolhendo cores com JColorChooser.
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import javax.swing.JButton;
8 import javax.swing.JFrame;
9 import javax.swing.JColorChooser;
10 import javax.swing.JPanel;
11
12 public class ShowColors2JFrame extends JFrame
13 {
14     private JButton changeColorJButton;
15     private Color color = Color.LIGHT_GRAY;
16     private JPanel colorJPanel;
17
18     // configura a GUI
19     public ShowColors2JFrame()
20     {
21         super( "Using JColorChooser" );
22
23         // cria JPanel para exibir cor
24         colorJPanel = new JPanel();
25         colorJPanel.setBackground( color );
26
27         // configura changeColorJButton e registra seu handler de evento
28         changeColorJButton = new JButton( "Change Color" );
29         changeColorJButton.addActionListener(
30
```

Importa a classe  
JColorChooser



# Resumo

```

31 new ActionListener() // classe interna anônima
32 {
33     // exibe JColorChooser quando o usuário clica no botão
34     public void actionPerformed( ActionEvent e)
35     {
36         color = JColorChooser.showDialog(
37             ShowColors2JFrame.this, "Choose a color", color );
38         // configura cor padrão, se nenhuma cor for retornada
39         color = Color.LIGHTBLUE;
40     }
41 }
42 // muda a cor de fundo do painel de conteúdo
43 colorJPanel.setBackground( color );
44 } // fim do método actionPerformed
45 } // fim da classe interna anônima
46 ); // fim da chamada para addActionListener
47
48
49 add( colorJPanel, BorderLayout.CENTER ); // adiciona colorJPanel
50 add( changeColorJButton, BorderLayout.SOUTH ); // adiciona botão
51
52 setSize( 400, 130 ); // configura o tamanho do frame
53 setVisible( true ); // exibe o frame
54 } // fim do construtor ShowColor2JFrame
55 } // fim da classe ShowColors2JFrame

```

Exibe o diálogo JColorChooser

ShowColors2JFrame  
.java

Referência ao componente-pai

Texto da barra de  
título

Cor inicial selecionada

Altera a cor do segundo plano do JPanel1



# Resumo

## ShowColors2.java

(1 de 2)

```
1 // Fig. 12.8: ShowColors2.java
2 // Escolhendo cores com JColorChooser.
3 import javax.swing.JFrame;
4
5 public class ShowColors2
6 {
7     // executa aplicativo
8     public static void main( String args[] )
9     {
10         ShowColors2JFrame application = new ShowColors2JFrame();
11         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12     } // fim de main
13 } // fim da classe ShowColors2
```

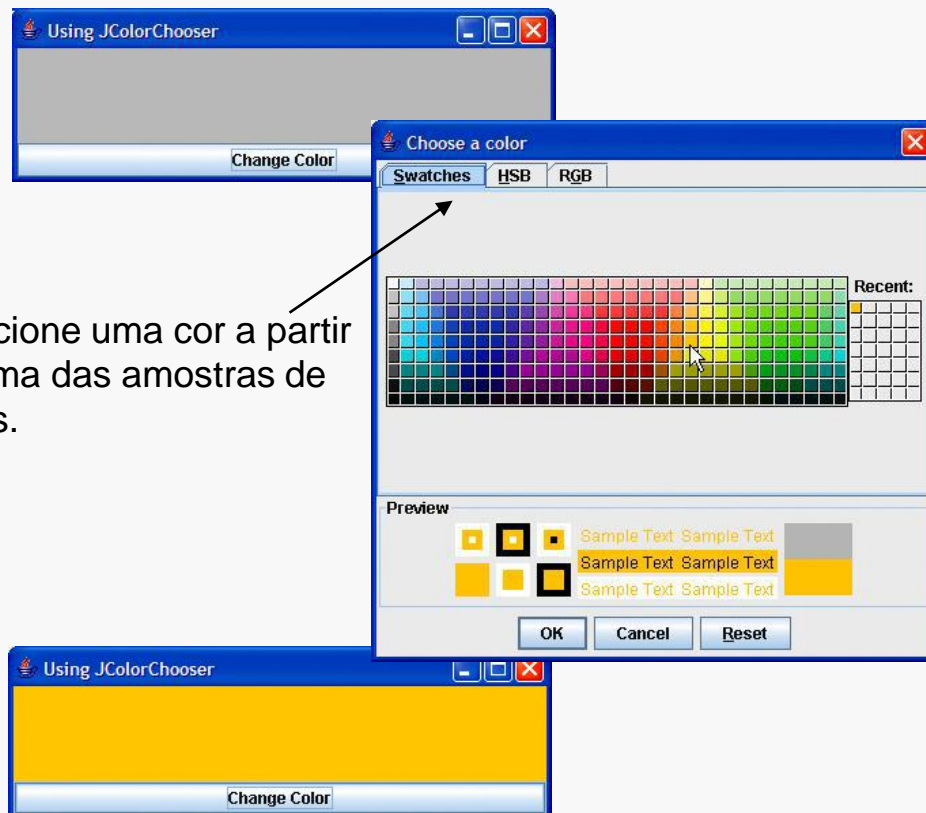


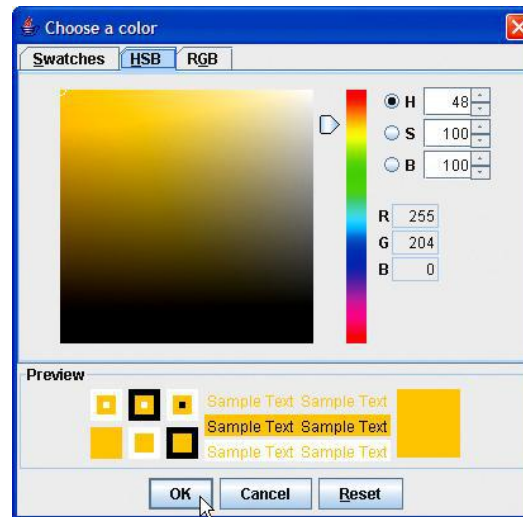
# Resumo

showColors2.java

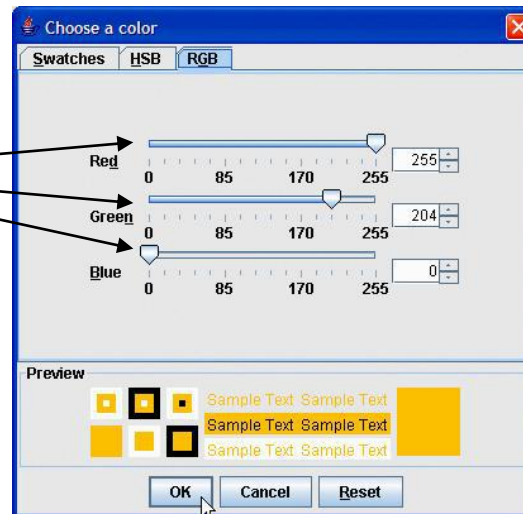
(2 de 2)

Selecione uma cor a partir de uma das amostras de cores.





Os controles  
deslizantes para  
selecionar os  
componentes de  
cor azul, verde e  
vermelho



**Figura 12.9 | Guias HSB e RGB do diálogo JColorChooser .**

## 12.4 Controle de fonte

- **Classe Font:**

- O construtor recebe três argumentos — *nome*, *estilo* e *tamanho da fonte*.
  - Nome da fonte — qualquer fonte atualmente suportada pelo sistema em que o programa está em execução.
  - Estilo da fonte — `Font.PLAIN`, `Font.ITALIC` ou `Font.BOLD`. Estilos de fontes podem ser utilizados em combinação.
  - Tamanho da fonte — medido em pontos. Um ponto tem 1/72 de uma polegada.
- Os métodos `getNames`, `getStyle` e `getSize` recuperam informações sobre o objeto `Font`.
- Os métodos `getFonts` e `setFont` de `Graphics` recuperam e configuram a fonte atual, respectivamente.





Método ou constante	Descrição
<i>Construtores, constantes e métodos <b>Font</b></i>	
<code>public final static int PLAIN</code>	Uma constante representando um estilo de fonte simples.
<code>public final static int BOLD</code>	Uma constante representando um estilo de fonte negrito.
<code>public final static int ITALIC</code>	Uma constante representando um estilo de fonte itálico.
<code>public Font( String name, int style, int size )</code>	Cria um objeto <b>Font</b> com o nome, o estilo e o tamanho de fonte especificados.
<code>public int getStyle()</code>	Retorna um valor de inteiro indicando o estilo de fonte atual.
<code>public int getSize()</code>	Retorna um valor de inteiro indicando o tamanho da fonte atual.

**Figura 12.10 | Métodos e constantes relacionados com Font.**  
(Parte 1 de 2.)

Método ou constante	Descrição
<code>public String getName()</code>	Retorna o nome da fonte atual como uma string.
<code>public String getFamily()</code>	Retorna o nome da família de fontes como uma string.
<code>public boolean isPlain()</code>	Retorna true se a fonte for simples, caso contrário false.
<code>public boolean isBold()</code>	Retorna true se a fonte for negrito, caso contrário false.
<code>public boolean isItalic()</code>	Retorna true se a fonte for itálica, caso contrário false.
<i>Metódos <b>Graphics</b> para manipular <b>Fonts</b></i>	
<code>public Font getFont()</code>	Retorna uma referência de objeto Font que representa a fonte atual.
<code>public void setFont( Font f )</code>	Configura a fonte atual como a fonte, o estilo e o tamanho especificados pela referência de objeto Font f.

**Figura 12.10 | Métodos e constantes relacionados com Font.**  
(Parte 2 de 2.)



## Dica de portabilidade 12.2

---

**O número de fontes varia significativamente de um sistema para outro. O Java fornece cinco nomes lógicos de fontes — `Serif`, `Monospaced`, `SansSerif`, `Dialog` and `DialogInput` — que podem ser utilizados em todas as plataformas Java. O ambiente de tempo de execução Java (Java Runtime Environment – JRE) em cada plataforma mapeia os nomes lógicos dessas fontes para as fontes reais instaladas na plataforma. As fontes reais utilizadas podem variar entre plataformas.**

# Resumo

## FontJPanel.java

(1 de 2)

```

1 // Fig. 12.11: FontJPanel.java
2 // Exibe strings em diferentes fontes e cores.
3 import java.awt.Font;
4 import java.awt.Color;
5 import java.awt.Graphics;
6 import javax.swing.JPanel;
7
8 public class FontJPanel extends JPanel
9 {
10     // exibe strings em diferentes fontes e cores
11     public void paintComponent( Graphics g )
12     {
13         super.paintComponent( g );
14
15         // configura fonte como Serif (Times), negrito, 12pt e desenha uma string
16         g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
17         g.drawString( "Serif 12 point bold.", 20, 50 );
18
19         // configura fonte como monoespaçada (Courier), 24 pt, itálico e desenha uma string
20         g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
21         g.drawString( "Monospaced 24 point italic.", 20, 70 );
22
23         // configura fonte como SansSerif (Helvetica), simples, 14pt e desenha uma string
24         g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
25         g.drawString( "SansSerif 14 point plain.", 20, 90 );
26

```



Combinando estilos

# Resumo

FontJPanel.java

(2 de 2)

```
27 // configura fonte como Serif (Times), 18 pt negrito/itálico e de
28 g.setColor( Color.RED );
29 g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
30 g.drawString( g.getFont().getName() + " " + g.getFont().getSize() +
31     " point bold italic.", 20, 110 );
32 } // fim do método paintComponent
33 } // fim da classe FontJPanel
```

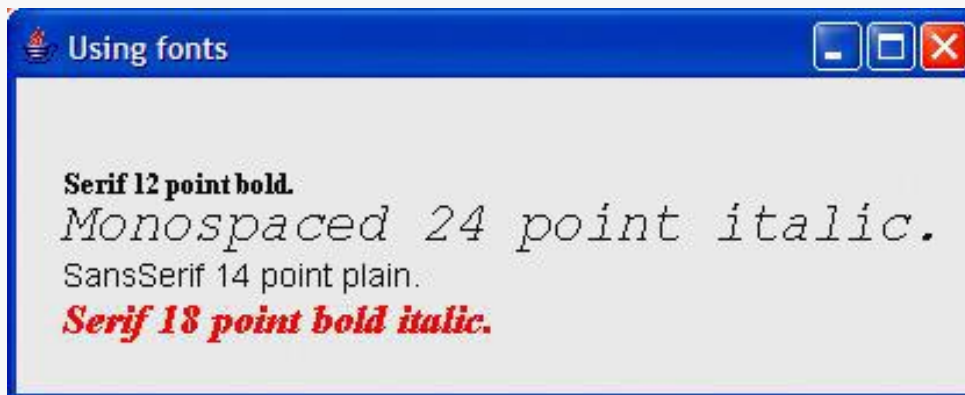
Recupera o nome e o tamanho de fonte da **Font** atual do objeto **Graphics**.



# Resumo

## Fonts.java

```
1 // Fig. 12.12: Fonts.java
2 // Utilizando fontes.
3 import javax.swing.JFrame;
4
5 public class Fonts
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria frame para FontJPanel
11         JFrame frame = new JFrame( "Using fonts" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         FontJPanel fontJPanel = new FontJPanel(); // cria FontJPanel
15         frame.add( fontJPanel ); // adiciona fontJPanel ao frame
16         frame.setSize( 420, 170 ); // configura o tamanho do frame
17         frame.setVisible( true ); // exhibe o frame
18     } // fim de main
19 } // fim da classe Fonts
```



# Observação de engenharia de software 12.2

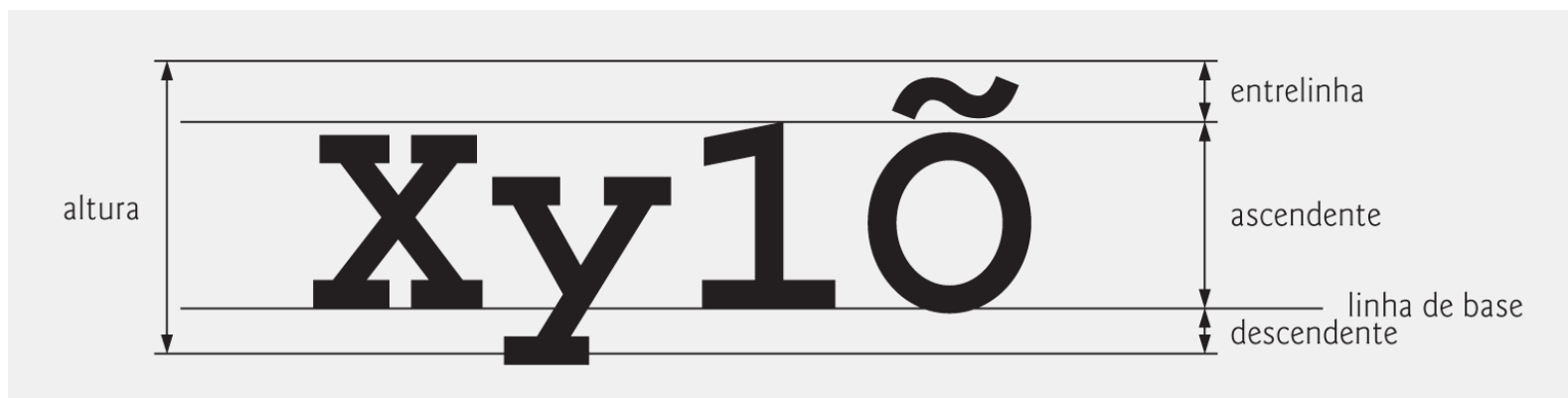
---

**Para alterar a fonte, você deve criar um novo objeto `Font`. Objetos `Font` são imutáveis — a classe `Font` não tem nenhum método *set* para alterar as características da fonte atual.**

# Métrica de fontes

- Métodos da classe **Font**
  - **getFamily** – retorna o nome da família de fontes à qual pertence a fonte atual
  - **isPlain**, **isBold**, **isItalic** – utilizados para determinar o estilo da fonte
- Métrica de fonte – informações precisas sobre uma fonte
  - Altura
  - Descendente – o valor mais baixo de um caractere em relação à linha de base
  - Ascendente – o valor mais alto de um caractere em relação à linha de base
  - Entrelinha – o espaçamento entre linhas
  - A classe **FontMetrics** declara vários métodos para obter a métrica da fonte.





**Figura 12.13 | Medidas de fonte.**

Método	Descrição
<i>Métodos FontMetrics</i>	
<code>public int getAscent()</code>	Retorna a ascendente de uma fonte em pontos.
<code>public int getDescent()</code>	Retorna a descendente de uma fonte em pontos.
<code>public int getLeading()</code>	Retorna a entrelinha de uma fonte em pontos.
<code>public int getHeight()</code>	Retorna a altura de uma fonte em pontos.
<i>Métodos Graphics para obter a FontMetrics de uma Font</i>	
<code>public FontMetrics getFontMetrics()</code>	Retorna o objeto FontMetrics para Font do desenho atual.
<code>public FontMetrics getFontMetrics( Font f )</code>	Retorna o objeto FontMetrics para o argumento Font especificado.

**Figura 12.14 | Métodos FontMetrics e Graphics para obter a medidas de fonte.**

# Resumo

## MetricsJPanel.java

(1 de 2)

```
1 // Fig. 12.15: MetricsJPanel.java
2 // Métodos FontMetrics e Graphics úteis para obter a métrica das fontes.
3 import java.awt.Font;
4 import java.awt.FontMetrics;
5 import java.awt.Graphics;
6 import javax.swing.JPanel;
7
8 public class MetricsJPanel extends JPanel
9 {
10     // exibe a métrica das fontes
11     public void paintComponent( Graphics g )
12     {
13         super.paintComponent( g ); // chama paintComponent da superclasse
14
15         g.setFont( new Font( "SansSerif", Font.BOLD, 12 );
16         FontMetrics metrics = g.getFontMetrics();
17
18         g.drawString( "Current font: " + g.getFont(), 10, 40 );
19         g.drawString( "Ascent: " + metrics.getAscent(), 10, 55 );
20         g.drawString( "Descent: " + metrics.getDescent(), 10, 70 );
21         g.drawString( "Height: " + metrics.getHeight(), 10, 85 );
22         g.drawString( "Leading: " + metrics.getLeading(), 10, 100 );
```

Recupera o objeto FontMetrics da Font atual

Recupera os valores da métrica da fonte



```
23 Font font = new Font( "Serif", Font.ITALIC, 14 );
24 metrics = g.getFontMetrics( font );
25 g.setFont( font );
26 g.drawString( "Current font: " + font, 10, 130 );
27 g.drawString( "Ascent: " + metrics.getAscent(), 10, 145 );
28 g.drawString( "Descent: " + metrics.getDescent(), 10, 160 );
29 g.drawString( "Height: " + metrics.getHeight(), 10, 175 );
30 g.drawString( "Leading: " + metrics.getLeading(), 10, 190 );
31 } // fim do método paintComponent
32 } // fim da classe MetricsJPanel
```

# Resumo

MetricsJPanel.java

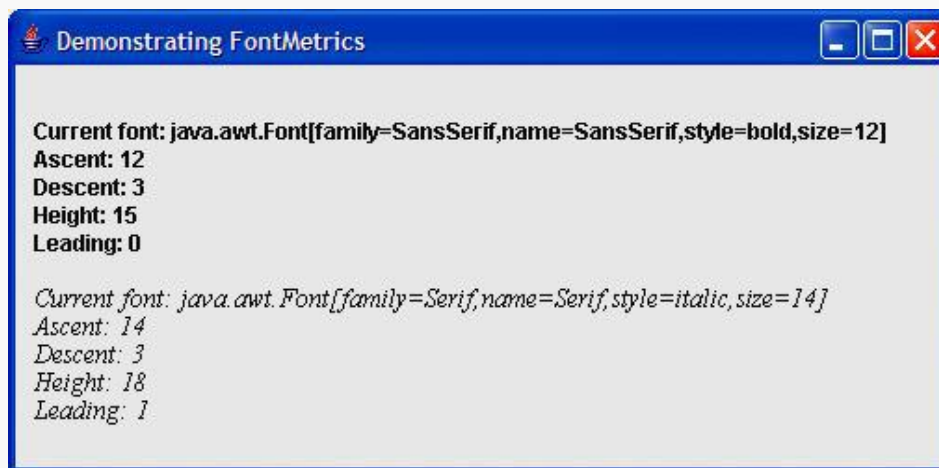
(2 de 2)



# Resumo

## Metrics.java

```
1 // Fig. 12.16: Metrics.java
2 // Exibindo a métrica de fonte.
3 import javax.swing.JFrame;
4
5 public class Metrics
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria o frame para MetricsJPanel
11         JFrame frame = new JFrame( "Demonstrating FontMetrics" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         MetricsJPanel metricsJPanel = new MetricsJPanel();
15         frame.add( metricsJPanel ); // adiciona metricsJPanel ao frame
16         frame.setSize( 510, 250 ); // configura o tamanho do frame
17         frame.setVisible( true ); // exibe o frame
18     } // fim de main
19 } // fim da classe Metrics
```



## 12.5 Desenhando linhas, retângulos e ovais

- **Métodos Graphics para desenhar linhas, retângulos e ovais:**
  - **fillRoundRect** e **drawRoundRect** — desenharam retângulos com cantos arredondados.
  - **Retângulo delimitador** — a área em que uma oval ou um retângulo arredondado serão desenhados.
  - **draw3DRect** e **fill3DRect** — desenharam um retângulo em 3D em alto ou baixo relevo.
  - **drawOval** e **fillOval** — desenharam ovais.



Método	Descrição
<code>public void drawLine( int x1, int y1, int x2, int y2 )</code>	Desenha uma linha entre o ponto (x1, y1) e o ponto (x2, y2).
<code>public void drawRect( int x, int y, int largura, int altura )</code>	Desenha um retângulo com a largura e altura especificadas. O canto superior esquerdo do retângulo tem as coordenadas (x, y). Somente o contorno do retângulo é desenhado utilizando a cor do objeto Graphics—o corpo do retângulo não é preenchido com essa cor.
<code>public void fillRect( int x, int y, int width, int height )</code>	Desenha um retângulo sólido com a largura e altura especificadas. O canto superior esquerdo do retângulo tem a coordenada (x, y). O retângulo é preenchido com a cor do objeto Graphics.
<code>public void clearRect( int x, int y, int largura, int altura )</code>	Desenha um retângulo preenchido com a largura e altura na cor de fundo atual. O canto superior esquerdo do retângulo tem a coordenada (x, y). Esse método é útil se o programador quiser remover uma parte de uma imagem.
<code>public void drawRoundRect( int x, int y, int largura, int altura, int arcwidth, int arcHeight )</code>	Desenha um retângulo com cantos arredondados na cor atual com a largura e altura especificadas. A arcwidth e arcHeight determinam o arredondamento dos cantos (veja Figura 12.20). Somente o contorno da forma é desenhado.

**Figura 12.17 | Métodos Graphics que desenhavam linhas, retângulos e ovais.**  
(Parte 1 de 2.)

Método	Descrição
<code>public void fillRoundRect( int x, int y, int largura, int altura, int larguraDoArco, int alturaDoArco )</code>	Desenha um retângulo preenchido com cantos arredondados na cor atual com a <code>largura</code> e a <code>altura</code> especificadas. A <code>arcWidth</code> e a <code>arcHeight</code> determinam o arredondamento dos cantos (veja Fig. 12.20).
<code>public void draw3DRect( int x, int y, int largura, int altura, boolean b )</code>	Desenha um retângulo tridimensional na cor atual com a <code>largura</code> e a <code>altura</code> especificadas. O canto superior esquerdo do retângulo tem as coordenadas (x, y). O retângulo parece em alto relevo quando <code>b</code> é verdadeiro e em baixo relevo quando <code>b</code> é falso. Somente o contorno da forma é desenhado.
<code>public void fill3DRect( int x, int y, int largura, int altura, boolean b )</code>	Desenha um retângulo tridimensional preenchido na cor atual com a <code>largura</code> e a <code>altura</code> especificadas. O canto superior esquerdo do retângulo tem as coordenadas (x, y). O retângulo parece em alto relevo quando <code>b</code> é verdadeiro e em baixo relevo quando <code>b</code> é falso.
<code>public void drawOval( int x, int y, int largura, int altura )</code>	Desenha uma oval na cor atual com a <code>largura</code> e a <code>altura</code> especificadas. O canto superior esquerdo do retângulo delimitador está nas coordenadas (x, y). A oval toca todos quatro lados do retângulo associado no centro de cada lado (veja Fig. 12.21). Somente o contorno da forma é desenhado.
<code>public void fillOval( int x, int y, int largura, int altura )</code>	Desenha uma oval preenchida na cor atual com a <code>largura</code> e a <code>altura</code> especificadas. O canto superior esquerdo do retângulo delimitador está nas coordenadas (x, y). A oval toca todos quatro lados do retângulo associado no centro de cada lado (veja Fig. 12.21).

**Figura 12.17 | Métodos Graphics que desenhavam linhas, retângulos e ovais.**  
(Parte 2 de 2)



```
1 // Fig. 12.18: LinesRectsOvalsJPanel.java
2 // Desenhando linhas, retângulos e ovals.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import javax.swing.JPanel;
6
7 public class LinesRectsOvalsJPanel extends JPanel
8 {
9     // exibe várias linhas, retângulos e ovals
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g ); // chama o método paint da superclasse
13
14        this.setBackground( Color.WHITE );
15
16        g.setColor( Color.RED );
17        g.drawLine( 5, 30, 380, 30 );
18
19        g.setColor( Color.BLUE );
20        g.drawRect( 5, 40, 90, 55 );
21        g.fillRect( 100, 40, 90, 55 );
22    }
```

Desenha uma linha reta

Desenha um retângulo vazio

Desenha um retângulo preenchido

# Resumo

LinesRectsOvals

JPanel.java

(1 de 2)



## Resumo

## LinesRectsOvals

## JPanel.java

(2 de 2)

```

23 g.setColor( Color.CYAN );
24 g.fillRect( 195, 40, 90, 55, 50, 50 );
25 g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
26
27 g.setColor( Color.YELLOW );
28 g.draw3DRect( 5, 100, 90, 55, true );
29 g.fill3DRect( 100, 100, 90, 55, false );
30
31 g.setColor( Color.MAGENTA );
32 g.drawOval( 195, 100, 90, 55 );
33 g.fillOval( 290, 100, 90, 55 );
34 } // fim do método paintComponent
35 } // fim da classe LinesRectsOvalsJPanel

```

Desenha um retângulo preenchido com cantos arredondados

Desenha um retângulo vazio com cantos arredondados

Desenha um retângulo vazio em alto relevo

Desenha um retângulo preenchido em baixo relevo

Desenha uma oval vazia

Desenha uma oval preenchida



```
1 // Fig. 12.19: LinesRectsOvals.java
2 // Desenhando linhas, retângulos e ovais.
3 import java.awt.Color;
4 import javax.swing.JFrame;
5
6 public class LinesRectsOvals
7 {
8     // executa o aplicativo
9     public static void main( String args[] )
10    {
11        // cria o frame para LinesRectsOvalsJPanel
12        JFrame frame =
13            new JFrame( "Drawing lines, rectangles and ovals" );
14        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
15    }
```

# Resumo

LinesRectsOvals

.java

(1 de 2)



```

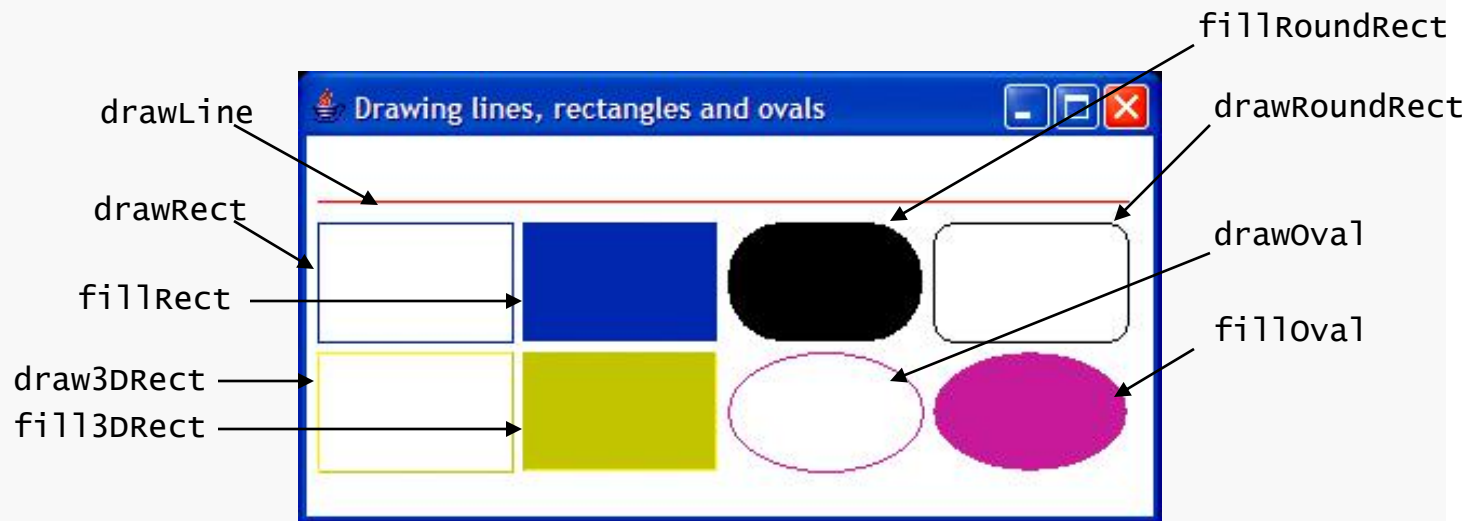
16 LinesRectsOvalsJPanel linesRectsOvalsJPanel =
17     new LinesRectsOvalsJPanel();
18 linesRectsOvalsJPanel.setBackground( Color.WHITE );
19 frame.add( linesRectsOvalsJPanel ); // adiciona o painel ao frame
20 frame.setSize( 400, 210 ); // configura o tamanho do frame
21 frame.setVisible( true ); // exhibe o frame
22 } // fim de main
23 } // fim da classe LinesRectsOvals

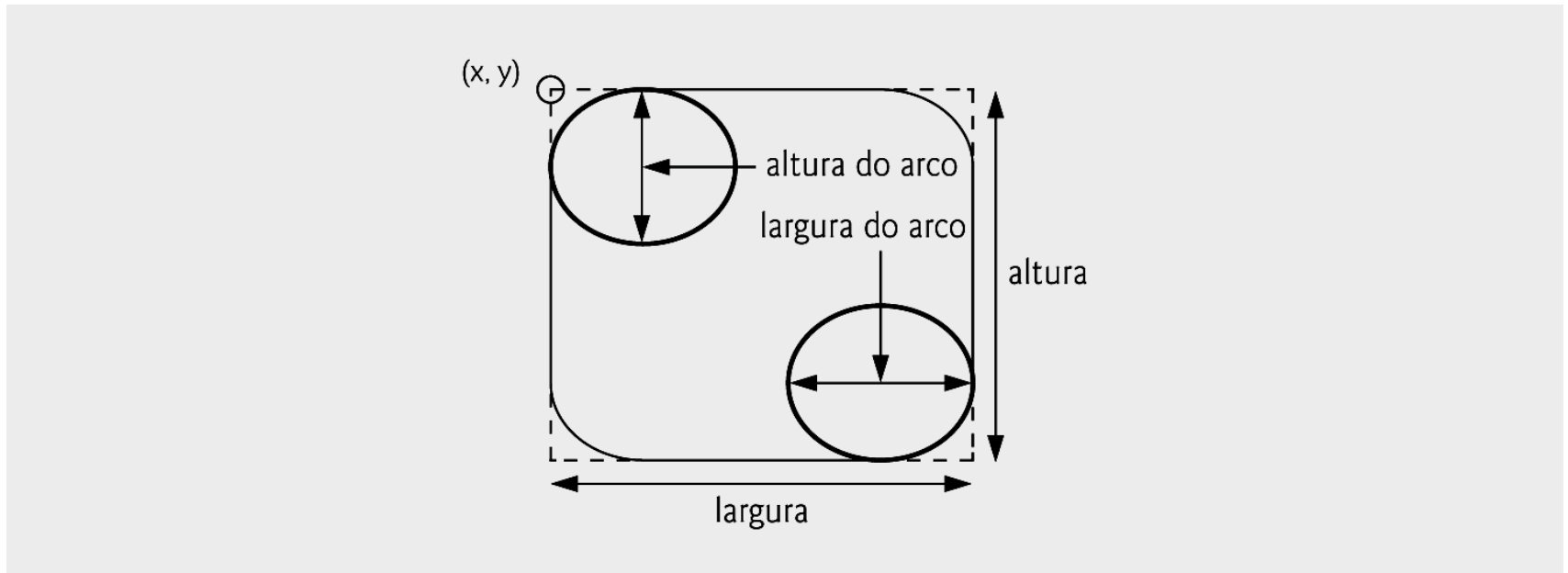
```

# Resumo

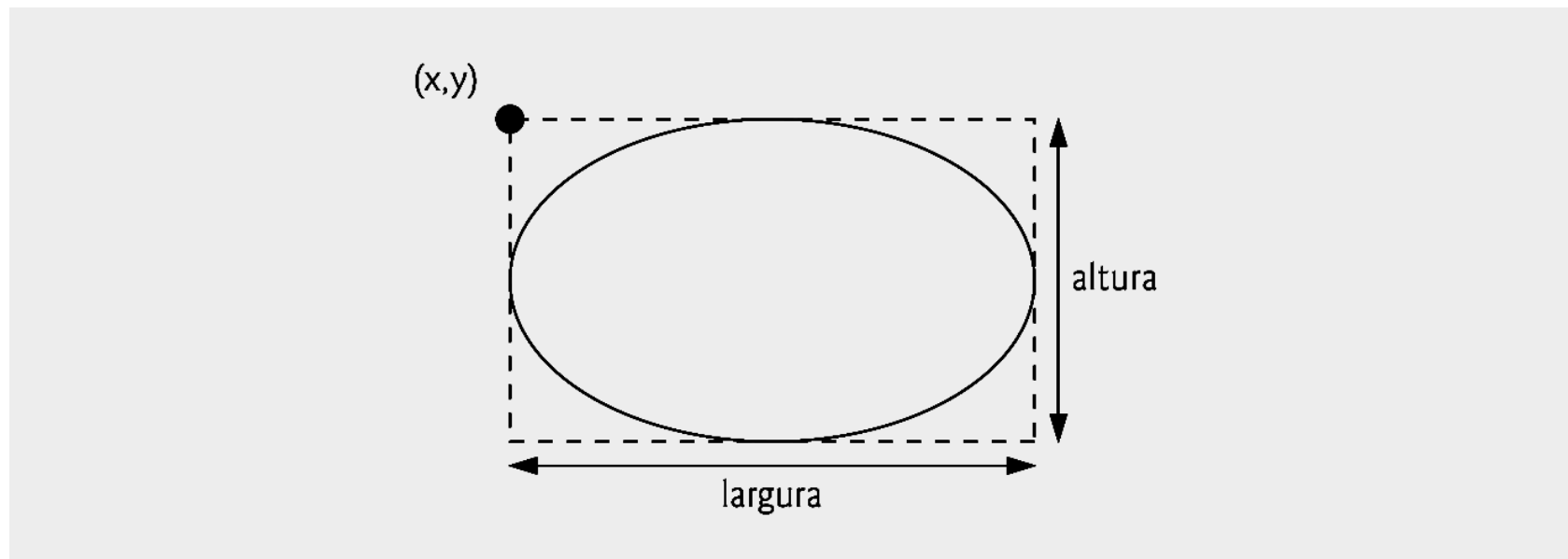
LinesRectsOvals  
.java

(2 de 2)





**Figura 12.20 | Largura do arco e altura do arco para retângulos arredondados.**

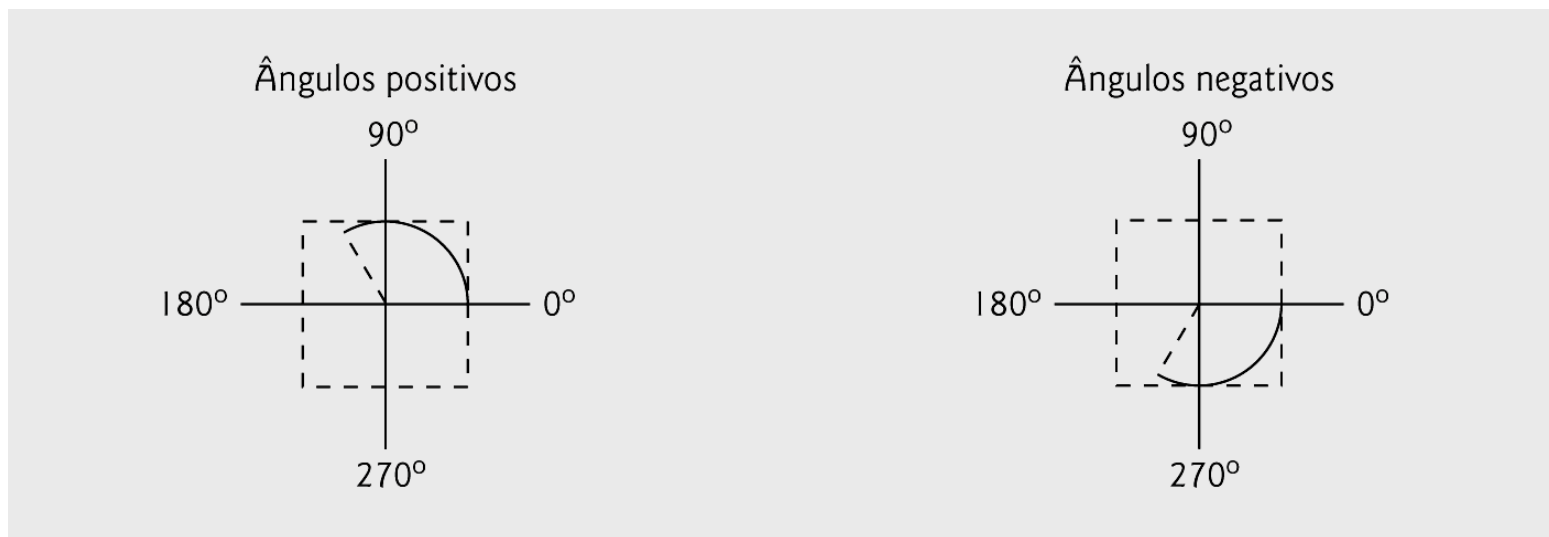


**Figura 12.21 | Oval delimitada por um retângulo.**

## 12.6 Desenhando arcos

- Um arco é desenhado como uma parte de uma elipse.
- Os arcos *varrem* (isto é, movem-se ao longo de uma curva) a partir de um *ângulo inicial* pelo número de graus especificado pelo ângulo do seu arco.
  - Varredura no sentido anti-horário medida em graus positivos.
  - Varredura no sentido horário medida em graus negativos.
- Os métodos `drawArcs` e `fillArc` de `Graphics` são utilizados para desenhar arcos





**Figura 12.22 | Ângulos de arco positivos e negativos.**



Método	Descrição
<code>public void drawArc( int x, int y, int largura, int altura, int ânguloInicial, int ânguloDoArco )</code>	Desenha um arco em relação ao canto superior esquerdo do retângulo delimitador e coordenadas x e y com a largura e altura especificadas. Métodos Graphics para desenhar arcos.
<code>public void fillArc( int x, int y, int largura, int altura, int ânguloInicial, int ânguloDoArco )</code>	Desenha um arco preenchido (isto é, um setor) em relação às coordenadas x e y do canto superior esquerdo do retângulo delimitador com a largura e a altura especificadas. Métodos Graphics para desenhar arcos.

**Figura 12.23 | Métodos Graphics para desenhar arcos.**

# Resumo

## ArcsJPanel.java

(1 de 2)

1 // Fig. 12.24: ArcsJPanel.java

2 // Desenhando arcos.

3 import java.awt.Color;

4 import java.awt.Graphics;

5 import javax.swing.JPanel;

6

7 public class ArcsJPanel extends JPanel

8 {

9 // desenha retângulo

10 public void paintComponent(

11 {

12 super.paintComponent(

13

14 // inicia em 0 e varre 360 graus

15 g.setColor( Color.RED );

16 g.drawRect( 15, 35, 80, 80 );

17 g.setColor( Color.BLACK );

18 g.drawArc( 15, 35, 80, 80, 0, 360 );

19

20 // inicia em 0 e varre 110 graus

21 g.setColor( Color.RED );

22 g.drawRect( 100, 35, 80, 80 );

23 g.setColor( Color.BLACK );

24 g.drawArc( 100, 35, 80, 80, 0, 110 );

25

As coordenadas  $x$  e  $y$  para o canto esquerdo superior do retângulo delimitador

Largura e altura do retângulo delimitador

Ângulo inicial

Ângulo de varredura

Desenha arcos vazios



# Resumo

ArcsJPanel.java

(2 de 2)

```
26 // inicia em 0 e varre -270 graus
27 g.setColor( Color.RED );
28 g.drawRect( 185, 35, 80, 80 );
29 g.setColor( Color.BLACK );
30 g.drawArc( 185, 35, 80, 80, 0, -270 );
31
32 // inicia em 0 e varre 360 graus
33 g.fillArc( 15, 120, 80, 40, 0, 360 );
34
35 // inicia em 270 e varre -90 graus
36 g.fillArc( 100, 120, 80, 40, 270, -90 );
37
38 // inicia em 0 e varre -270 graus
39 g.fillArc( 185, 120, 80, 40, 0, -270 );
40 } // fim do método paintComponent
41 } // fim da classe ArcsJPanel
```

Desenha arcos preenchidos

Valores negativos indicam a  
varredura do arco no sentido  
horário



# Resumo

DrawArcs.java

(1 de 2)

```
1 // Fig. 12.25: DrawArcs.java
2 // Desenhando arcos.
3 import javax.swing.JFrame;
4
5 public class DrawArcs
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria o frame para ArcsJPanel
11         JFrame frame = new JFrame( "Drawing Arcs" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13     }
```

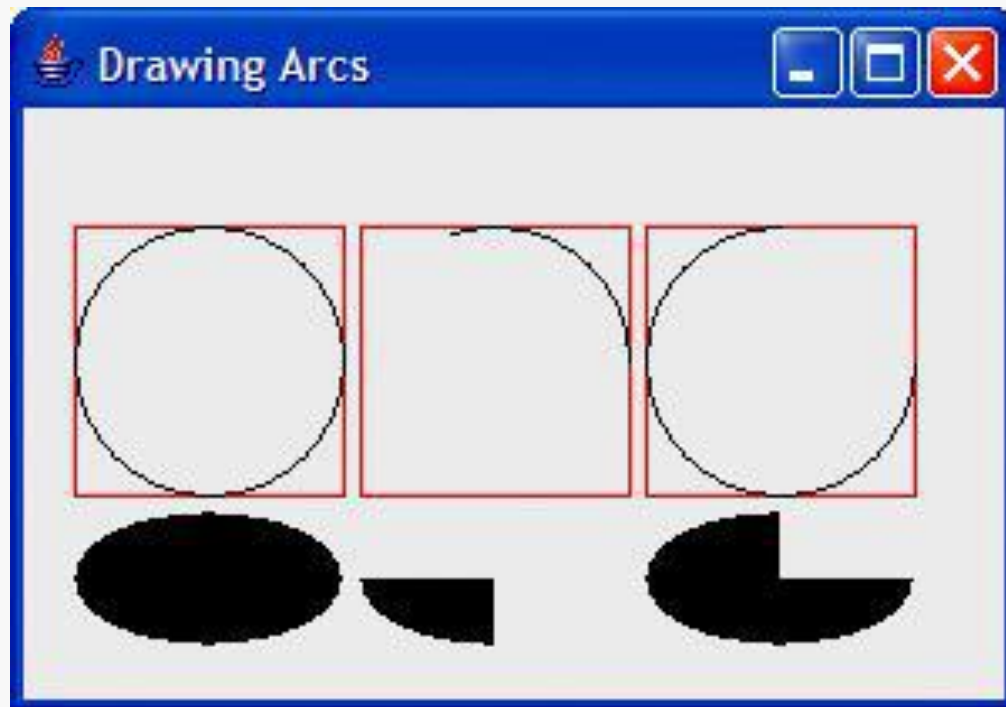


```
14 ArcsJPanel arcsJPanel = new ArcsJPanel(); // cria ArcsJPanel
15 frame.add( arcsJPanel ); // adiciona arcsJPanel ao frame
16 frame.setSize( 300, 210 ); // configura o tamanho do frame
17 frame.setVisible( true ); // exhibe o frame
18 } // fim de main
19 } // fim da classe DrawArcs
```

# Resumo

DrawArcs.java

(2 de 2)



# 12.7 Desenhando polígonos e polilinhas

- **Polígonos:**

- Formas fechadas com múltiplos lados compostas de segmentos de linha retos.
- O método `drawPolygon` e `fillPolygon` de `Graphics` para exibir polígonos.
- Os polígonos podem ser representados utilizando a classe `Polygon` — essa classe contém o método `addPoint` para adicionar pontos a um `Polygon`.

- **Polilinhas:**

- Sequências de pontos conectados.
- Método `drawPolyline` de `Graphics` para exibir polilinhas.



Método	Descrição
<i>Métodos Graphics para desenhar polígonos</i>	
<code>public void drawPolygon( int xPoints[], int yPoints[], int points )</code>	Desenha um polígono. A coordenada x de cada ponto é especificada no array <code>xPoints</code> e, a coordenada y de cada ponto, no array <code>yPoints</code> . O último argumento especifica o número de <code>points</code> . Esse método desenha um polígono. Se o último ponto for diferente do primeiro, o polígono é fechado por uma linha que conecta o último ponto ao primeiro.
<code>public void drawPolyline( int xPoints[], int yPoints[], int points )</code>	Desenha uma seqüência de linhas conectadas. A coordenada x de cada ponto é especificada no array <code>xPoints</code> e a coordenada y de cada ponto é especificada no array <code>yPoints</code> . O último argumento especifica o número de <code>points</code> . Se o último ponto for diferente do primeiro, a polilinha não é fechada.
<code>public void drawPolygon( Polygon p )</code>	Desenha o polígono especificado.
<code>public void fillPolygon( int xPoints[], int yPoints[], int points )</code>	Desenha um polígono preenchido. A coordenada x de cada ponto é especificada no array <code>xPoints</code> e a coordenada y de cada ponto é especificada no array <code>yPoints</code> . O último argumento especifica o número de <code>points</code> . Esse método desenha um polígono. Se o último ponto for diferente do primeiro, o polígono é fechado por uma linha que conecta o último ponto ao primeiro.

**Figura 12.26 | Métodos Graphics para polígonos e métodos da classe Polygon. (Parte 1 de 2.)**

Método	Descrição
<code>public void fillPolygon( Polygon p )</code>	Desenha o polígono preenchido especificado. O polígono é fechado.
<i>Construtores e métodos Polygon</i>	
<code>public Polygon()</code>	Constrói um novo objeto de polígono. O polígono não contém nenhum ponto.
<code>public Polygon( int xValues[], int yValues[], int numberOfPoints )</code>	Constrói um novo objeto de polígono. O polígono tem numberOfPoints lados, com cada ponto consistindo em uma coordenada x de xValues e uma coordenada y de yValues.
<code>public void addPoint( int x, int y )</code>	Adiciona pares das coordenadas x e y ao Polygon.

**Figura 12.26 | Métodos Graphics para polígonos e métodos da classe Polygon. (Parte 2 de 2)**



# Resumo

## PolygonsJPanel .java

(1 de 2)

```

1 // Fig. 12.27: PolygonsJPanel.java
2 // Desenhando polígonos.
3 import java.awt.Graphics;
4 import java.awt.Polygon;
5 import javax.swing.JPanel;
6
7 public class PolygonsJPanel extends JPanel
8 {
9     // desenha polígonos e polilinhas
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g ); // chama paintComponent da superclasse
13
14        // desenha polígono com o objeto Polygon
15        int xValues[] = { 20, 40, 50, 30, 20, 15 };
16        int yValues[] = { 50, 50, 60, 80, 80, 60 };
17        Polygon polygon1 = new Polygon( xValues, yValues, 6 );
18        g.drawPolygon( polygon1 );
19
20        // desenha polilinhas com dois arrays
21        int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
22        int yValues2[] = { 100, 100, 110, 110, 130, 110, 100 };
23        g.drawPolyline( xValues2, yValues2, 7 );
24    }

```

Cria um objeto `Polygon` a partir dos conjuntos de coordenadas  $x$  e  $y$

Desenha um `Polygon` vazio

Desenha polilinhas a partir dos conjuntos de coordenadas  $x$  e  $y$



## Resumo

```
25 // preenche polígono com dois arrays
26 int xValues3[] = { 120, 140, 150, 190 };
27 int yValues3[] = { 40, 70, 80, 60 };
28 g.fillPolygon( xValues3, yValues3, 4 );
29
30 // desenha polígono preenchido com o objeto Polygon
31 Polygon polygon2 = new Polygon();
32 polygon2.addPoint( 165, 135 );
33 polygon2.addPoint( 175, 150 );
34 polygon2.addPoint( 270, 200 );
35 polygon2.addPoint( 200, 220 );
36 polygon2.addPoint( 130, 180 );
37 g.fillPolygon( polygon2 );
38 } // fim do método paintComponent
39 } // fim da classe PolygonsJPanel
```

Desenha um polígono a partir dos conjuntos de coordenadas  $x$  e  $y$  sem criar um objeto **Polygon**

.java

Adiciona coordenadas a **Polygon** com o método **addPoint**

(2 de 2)



# Resumo

## DrawPolygons.java

(1 de 2)

```
1 // Fig. 12.28: DrawPolygons.java
2 // Desenhando polígonos.
3 import javax.swing.JFrame;
4
5 public class DrawPolygons
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria frame para PolygonsJPanel
11         JFrame frame = new JFrame( "Drawing Polygons" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13     }
```

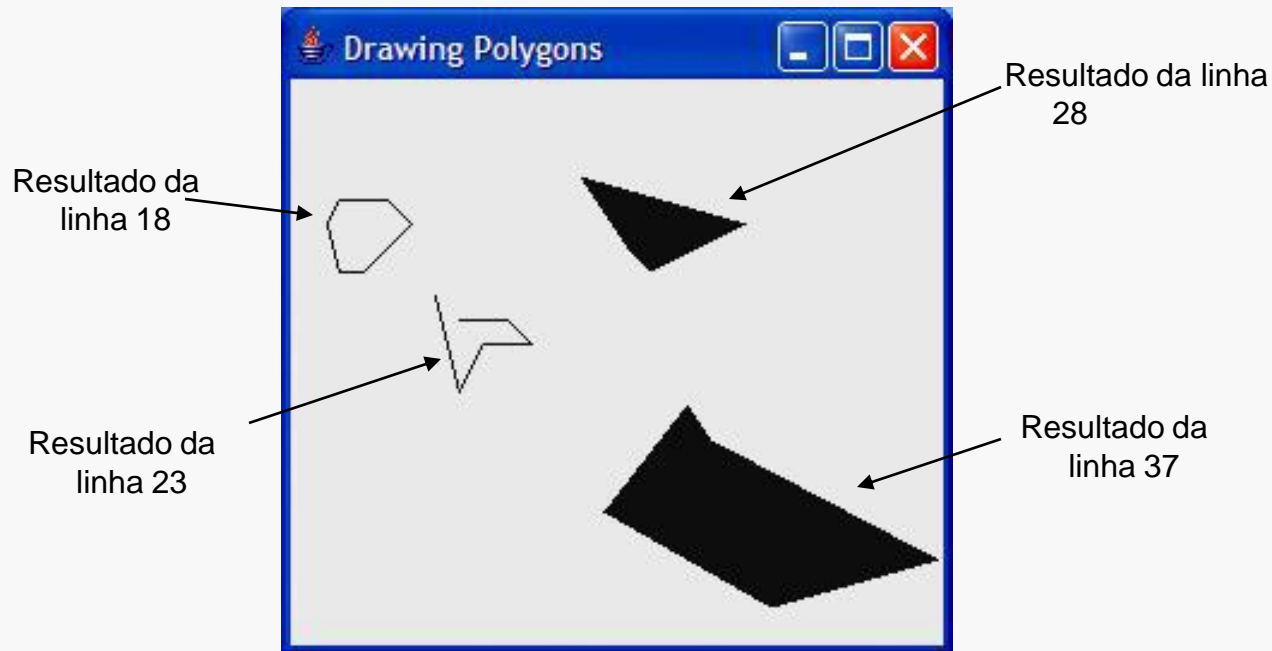


```
14 PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
15 frame.add( polygonsJPanel ); // adiciona polygonsJPanel ao frame
16 frame.setSize( 280, 270 ); // configura o tamanho do frame
17 frame.setVisible( true ); // exibe o frame
18 } // fim de main
19 } // fim da classe DrawPolygons
```

# Resumo

## DrawPolygons.java

(2 de 2)



## Erro comum de programação 12.1

---

**Uma `ArrayIndexOutOfBoundsException` é lançada se o número de pontos especificados no terceiro argumento para o método `drawPolygon` ou para o método `fillPolygon` for maior que o número de elementos nos arrays de coordenadas que especificam o polígono a exibir.**

## 12.8 API do Java 2D

- Fornece capacidades gráficas bidimensionais avançadas para manipulações gráficas detalhadas e complexas.
- Recursos para processar *line art* (desenho a traço), texto e imagens.
- Realizado com a classe `Graphics2D`.



# Linhas, retângulos, retângulos arredondados, arcos e ovais

- Formas de Java 2D especificadas com valores de pontos flutuantes de dupla precisão — `Line2D.Double`, `Rectangle2D.Double`, `RoundRectangle2D.Double`, `Arc2D.Double`, `Ellipse2D.Double`.
- Pintando com o objeto `Graphics2D`.
  - O método `setPaint` configura as cores para o objeto `Graphics2D` quando formas são desenhadas como um objeto `Paint`.
  - O objeto `Paint` pode ser um objeto `Color` pré-declarado ou uma instância das classes `GradientPaint`, `SystemColor` ou `TexturePaint`.
  - A `GradientPaint` é utilizada para desenhar com um gradiente (degradê) — gradientes podem ser cíclicos ou acíclicos.
  - `TexturePaint` é utilizada para pintar replicando uma imagem armazenada.



# Linhas, retângulos, retângulos arredondados, arcos e ovais (Cont.)

- Método `fill` de `Graphics2D` utilizado para desenhar um objeto `Shape` preenchido — um objeto que implementa a interface `Shape`.
- Método `draw` de `Graphics2D` utilizado para desenhar um objeto de `Shape`.
- Configurando um traço de uma linha ou borda:
  - Método `setStroke` de `Graphics2D` requer um argumento que implementa a interface `Stroke`.
  - Classe `BasicStroke` – pode especificar a largura da linha, terminações de seta, junções de linhas.
- Constantes `Arc2D.Double`:
  - `Arc2D.PIE` – o arco deve ser fechado por duas linhas – uma do ponto inicial até o centro, uma do centro até o ponto final.
  - `Arc2D.CHORD` – desenha uma linha do ponto inicial ao ponto final.
  - `Arc2D.OPEN` – o arco não deve ser fechado.





# Resumo

ShapesJPanel.java

(1 de 3)

```

1 // Fig. 12.29: ShapesJPanel.java
2 // Demonstrando algumas formas 2D Java.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.BasicStroke;
6 import java.awt.GradientPaint;
7 import java.awt.TexturePaint;
8 import java.awt.Rectangle;
9 import java.awt.Graphics2D;
10 import java.awt.geom.Ellipse2D;
11 import java.awt.geom.Rectangle2D;
12 import java.awt.geom.RoundRectangle2D;
13 import java.awt.geom.Arc2D;
14 import java.awt.geom.Line2D;
15 import java.awt.image.BufferedImage;
16 import javax.swing.JPanel;
17
18 public class ShapesJPanel extends JPanel
19 {
20     // desenha formas com Java 2D API
21     public void paintComponent( Graphics g )
22     {
23         super.paintComponent( g ); // chama o paintComponent da superclasse
24
25         Graphics2D g2d = ( Graphics2D ) g; // faz coerção de g para Graphics2D
26

```

Classes de formas da API do Java 2D

Criado uma referência a Graphics2D



```

27 // desenha elipse 2D preenchida com um gradiente azul-amarelo
28 g2d.setPaint( new GradientPaint( 5, 30, Color.BLUE, 35, 100,
29     Color.YELLOW, true ) );
30 g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );

```

Desenha uma elipse preenchida utilizando gradientes

```

32 // desenha retângulo 2D em vermelho

```

```

33 g2d.setPaint( Color.RED );
34 g2d.setStroke( new BasicStroke( 10.0f ) );
35 g2d.draw( new Rectangle2D.Double( 80, 30, 65, 100 ) );

```

Configura o objeto Graphics2D a desenhar utilizando um gradiente de azul a amarelo

(2 de 3)

```

37 // desenha retângulo arredondado 2D com um fundo amarelo

```

```

38 BufferedImage buffImage = new BufferedImage( 10, 10,
39     BufferedImage.TYPE_INT_RGB );

```

Configura a largura da borda como 10 pixels

```

41 // obtém Graphics2D de buffImage e desenha nela

```

```

42 Graphics2D gg = buffImage.createGraphics();
43 gg.setColor( Color.YELLOW ); // desenha em amarelo
44 gg.fillRect( 0, 0, 10, 10 ); // desenha um retângulo
45 gg.setColor( Color.BLACK ); // desenha em preto
46 gg.drawRect( 1, 1, 6, 6 ); // desenha um retângulo
47 gg.setColor( Color.BLUE ); // desenha em azul
48 gg.fillRect( 1, 1, 3, 3 ); // desenha um retângulo preenchido
49 gg.setColor( Color.RED ); // desenha em vermelho
50 gg.fillRect( 4, 4, 3, 3 ); // desenha um retângulo preenchido
51

```

Cria imagem a ser utilizada para o objeto TexturePaint



# Resumo

ShapesJPanel.java

```

52 // pinta buffImage sobre o JFrame
53 g2d.setPaint( new TexturePaint( buffImage,
54     new Rectangle( 10, 10 ) ) );
55 g2d.fill(
56     new RoundRectangle2D.Double( 155, 30, 75,
57 // Desenha arco 2D em forma de tor
58 g2d.setPaint( Color.WHITE );
59 g2d.setStroke( new BasicStroke( 6.0f ) );
60 g2d.draw(
61     new Arc2D.Double( 240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
62
63
64 // Desenha linhas 2D em verde e amarelo
65 g2d.setPaint( Color.GREEN );
66 g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
67
68 // desenha uma linha em 2D utilizando traço
69 float dashes[] = { 10 }; // especifica padrão de traço
70 g2d.setPaint( Color.YELLOW );
71 g2d.setStroke( new BasicStroke( 4, BasicStroke.CAP_ROUND,
72     BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
73 g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
74 } // fim do método paintComponent
75 } // fim da classe ShapesJPanel
  
```

Cria o objeto TexturePaint a partir da imagem

Desenha um retângulo arredondado, preenchido com uma imagem que se repete

Desenha um arco utilizando uma borda branca, 6 pixels de largura

Desenha uma linha verde sólida

Configura um tracejado

Desenha uma linha tracejada amarela



# Resumo

## Shapes.java

(1 de 2)

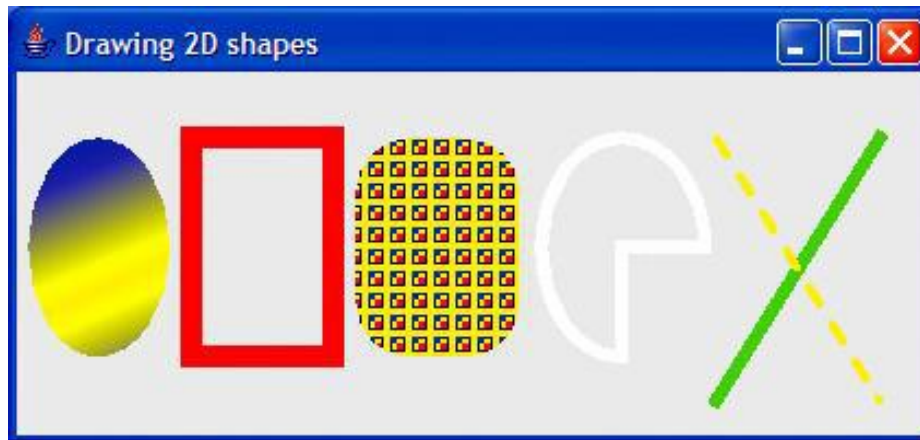
```
1 // Fig. 12.30: Shapes.java
2 // Demonstrando algumas formas 2D Java.
3 import javax.swing.JFrame;
4
5 public class Shapes
6 {
7     // executa o aplicativo
8     public static void main( String args[] )
9     {
10         // cria o frame para ShapesJPanel
11         JFrame frame = new JFrame( "Drawing 2D shapes" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         // cria ShapesJPanel
15         ShapesJPanel shapesJPanel = new ShapesJPanel();
16
17         frame.add( shapesJPanel ); // adiciona shapesJPanel ao frame
18         frame.setSize( 425, 200 ); // configura o tamanho do frame
19         frame.setVisible( true ); // exibe o frame
20     } // fim de main
21 } // fim da classe Shapes
```



# Resumo

Shapes.java

(2 de 2)



# Caminhos gerais

- Um caminho geral é uma forma construída de linhas retas e curvas complexas.
- Classe `GeneralPath`:
  - O método `moveTo` especifica o primeiro ponto em um caminho geral.
  - O método `lineTo` desenha uma linha para o próximo ponto no caminho.
  - O método `closePath` completa o caminho geral.
- Método `Graphics2D translate` — utilizado para mover a origem do desenho.
- Método `Graphics2D rotate` utilizado para rotacionar a próxima forma exibida.



# Resumo

Shapes2JPanel.java

(1 de 2)

```
1 // Fig. 12.31: Shapes2JPanel.java
2 // Demonstrando um caminho geral.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.Graphics2D;
6 import java.awt.geom.GeneralPath;
7 import java.util.Random;
8 import javax.swing.JPanel;
9
10 public class Shapes2JPanel extends JPanel
11 {
12     // desenha caminhos gerais
13     public void paintComponent( Graphics g )
14     {
15         super.paintComponent( g ); // chama o paintComponent da superclasse
16         Random random = new Random(); // obtém o gerador de números aleatórios
17
18         int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1,
19         int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
20
21         Graphics2D g2d = ( Graphics2D ) g;
22         GeneralPath star = new GeneralPath(); // cria objeto GeneralPath
23
24         // configura a coordenada do General Path
25         star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
26
```

Cria um objeto GeneralPath

Configura o ponto inicial do  
objeto GeneralPath



# Resumo

Shapes2JPanel.java

```

27 // cria a estrela -- isso não desenha a estrela
28 for ( int count = 1; count < xPoints.length; count++ )
29     star.lineTo( xPoints[ count ], yPoints[ count ] );
30
31 star.closePath(); // fecha a forma
32
33 g2d.translate( 200, 200 ); // converte a origem pa
34
35 // gira em torno da origem e
36 for ( int count = 1; count <= xPoints.length; count++ )
37 {
38     g2d.rotate( Math.PI / 10.0 ); // rotaciona o sistema de coordenadas
39
40     // configura cores aleatórias
41     g2d.setColor( new Color( random.nextInt( 256 ),
42         random.nextInt( 256 ), random.nextInt( 256 ) ) );
43
44     g2d.fill( star ); // desenha estrela preenchida
45 } // fim de for
46 } // fim do método paintComponent
47 } // fim da classe Shapes2JPanel

```

Adiciona linhas do caminho geral

Desenha uma linha do último ponto até o primeiro ponto

Rotaciona aproximadamente 18 graus

Desenha uma estrela no ângulo atual em torno da origem





# Resumo

## Shapes2.java

(1 de 2)

```
1 // Fig. 12.32: Shapes2.java
2 // Demonstrando um caminho geral.
3 import java.awt.Color;
4 import javax.swing.JFrame;
5
6 public class Shapes2
7 {
8     // executa o aplicativo
9     public static void main( String args[] )
10    {
11        // cria o frame para Shapes2JPanel
12        JFrame frame = new JFrame( "Drawing 2D Shapes" );
13        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
14    }
```



```
15 Shapes2JPanel shapes2JPanel = new Shapes2JPanel();
16 frame.add( shapes2JPanel ); // adiciona shapes2JPanel ao frame
17 frame.setBackground( Color.WHITE ); // configura cor de fundo do frame
18 frame.setSize( 400, 400 ); // configura o tamanho do frame
19 frame.setVisible( true ); // exibe o frame
20 } // fim de main
21 } // fim da classe Shapes2
```

# Resumo

## Shapes2.java

(2 de 2)

