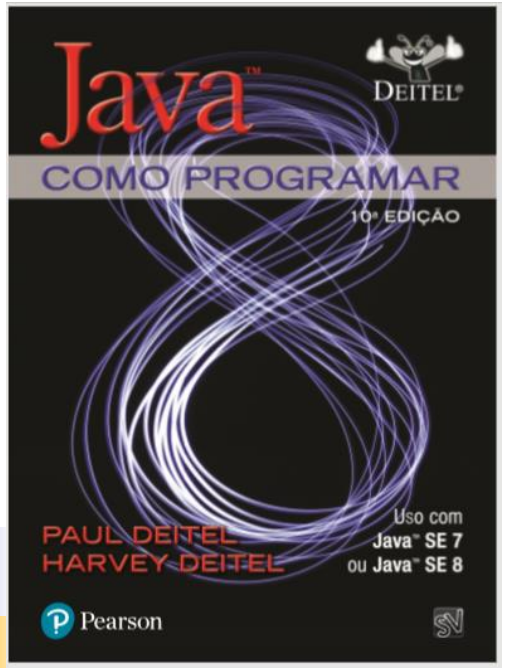


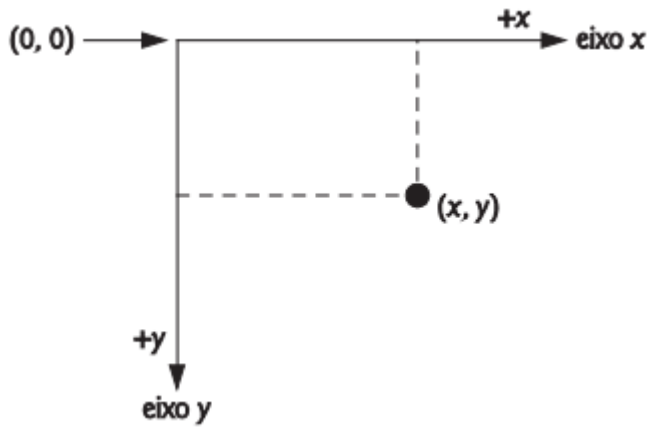
Graphics e Java 2D

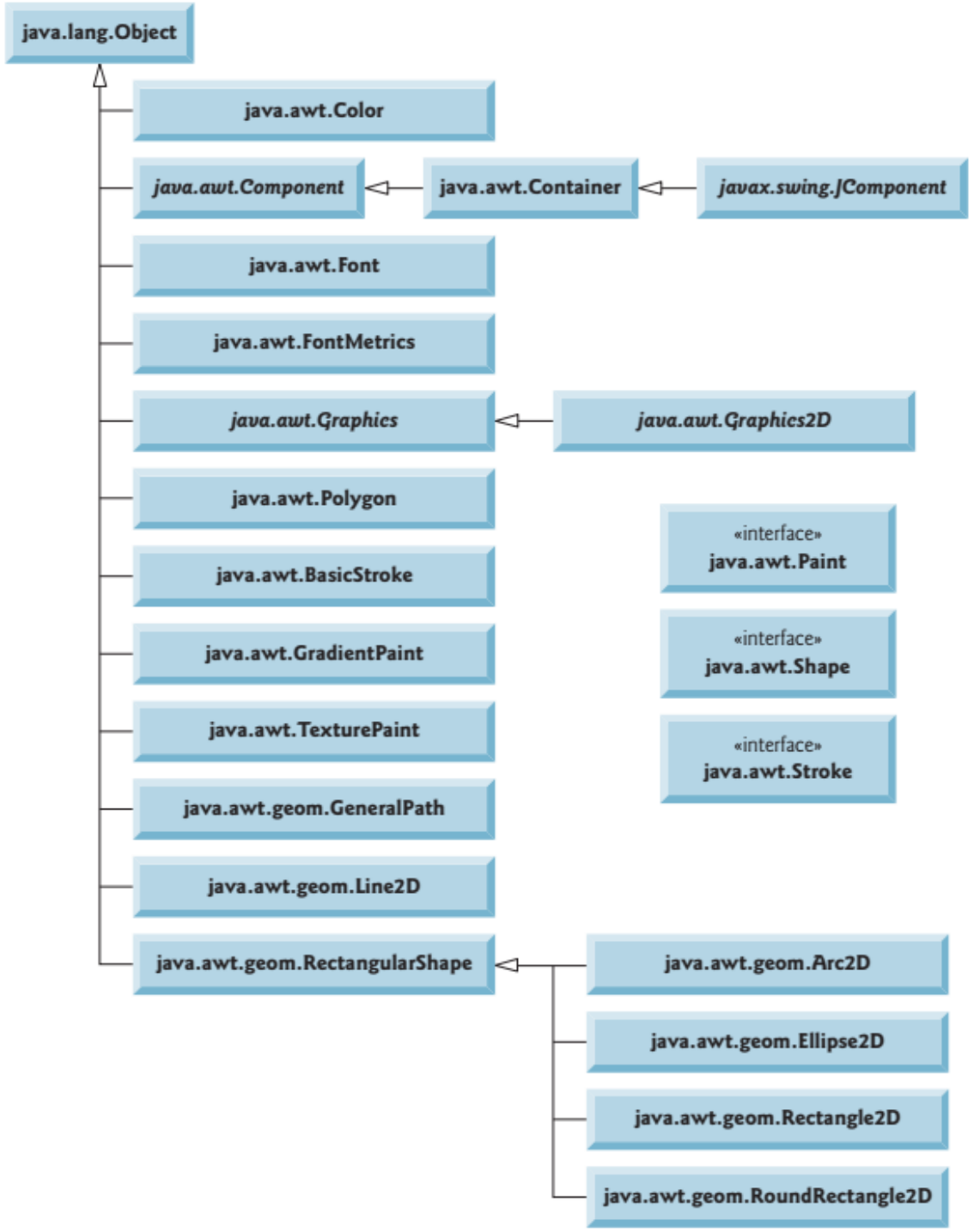
Notas de Aula

Prof. André Bernardi

andrebernardi@unifei.edu.br

Introdução


(0, 0)
+x
eixo x
(x, y)
+y
eixo y


java.lang.Object
java.awt.Color
java.awt.Component
java.awt.Container
javax.swing.JComponent
java.awt.Font
java.awt.FontMetrics
java.awt.Graphics
java.awt.Graphics2D
java.awt.Polygon
java.awt.BasicStroke
java.awt.GradientPaint
java.awt.TexturePaint
java.awt.geom.GeneralPath
java.awt.geom.Line2D
java.awt.geom.RectangularShape
«interface»
java.awt.Paint
«interface»
java.awt.Shape
«interface»
java.awt.Stroke
java.awt.geom.Arc2D
java.awt.geom.Ellipse2D
java.awt.geom.Rectangle2D
java.awt.geom.RoundRectangle2D

Contexto e Objeto Graphics

- O Contexto Gráfico permite desenhar na tela em um programa escrito em Java.
- O objeto **Graphics** gerencia um contexto gráfico.
- O método ***paintComponent*** da classe *Component* recebe um contexto gráfico como parâmetro.
  - `public void paintComponent(Graphics g)`

Contexto e Objeto Graphics

- Para desenhar na tela deve-se sobrescrever o método *paintComponent* em uma classe derivada de Component.
- A classe ***JPanel*** será usada como base para os desenhos em Java.
- Para atualizar os elementos gráficos desenhado em um componente utiliza-se o método:
  - `public void repaint( )`

Controle de Cor

A classe Color declara métodos e constantes para manipular cores em um programa Java.

Constante de cor	Valor RGB
public static final Color RED	255, 0, 0
public static final Color GREEN	0, 255, 0
public static final Color BLUE	0, 0, 255
public static final Color ORANGE	255, 200, 0
public static final Color PINK	255, 175, 175
public static final Color CYAN	0, 255, 255
public static final Color MAGENTA	255, 0, 255
public static final Color YELLOW	255, 255, 0
public static final Color BLACK	0, 0, 0
public static final Color WHITE	255, 255, 255
public static final Color GRAY	128, 128, 128
public static final Color LIGHT_GRAY	192, 192, 192
public static final Color DARK_GRAY	64, 64, 64

Métodos da classe Color

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

Exemplo uso da classe Color

- `public Color( int r, int g, int b )`
- `public Color( float r, float g, float b )`
- `public int getRed()`
- `public int getGreen()`
- `public int getBlue()`

- `public Color getColor()`
- `public void setColor( Color c )`

Exemplo uso da classe Color

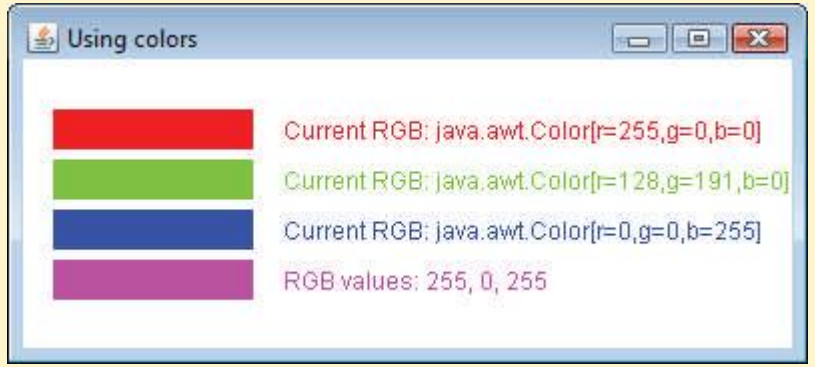
```
1 // Figura 13.5: ColorJPanel.java
2 // Alterando cores de desenho.
3 import java.awt.Graphics;
4 import java.awt.Color;
5 import javax.swing.JPanel;
6
7 public class ColorJPanel extends JPanel
8 {
9     // desenha retângulos e Strings em cores diferentes
10    @Override
11    public void paintComponent(Graphics g)
12    {
13       super.paintComponent(g);
14       this.setBackground(Color.WHITE);
15
16       // nova cor de desenho configurada utiliza 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);
```

Exemplo uso da classe Color

```
21        // nova cor de desenho configurada utiliza 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        // nova cor de desenho configurada usa objetos Color estáticos
27        g.setColor(Color.BLUE);
28        g.fillRect(15, 75, 100, 20);
29        g.drawString("Current RGB: " + g.getColor(), 130, 90);
30
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     }
38  } // fim da classe ColorJPanel
```

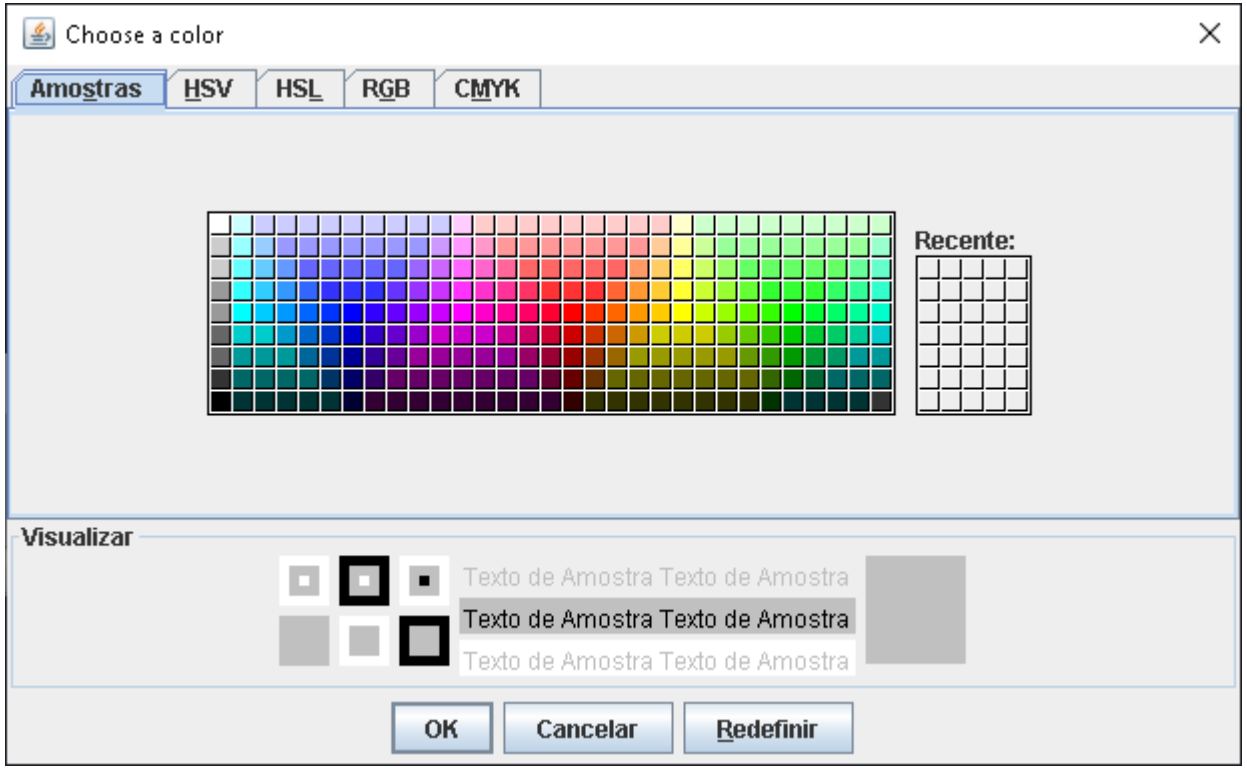
Exemplo uso da classe Color

```
1 // Figura 13.6: ShowColors.java
2 // Demonstrando Colors.
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();
15        frame.add(colorJPanel);
16        frame.setSize(400, 180);
17        frame.setVisible(true);
18    }
19 } // fim da classe ShowColors
```



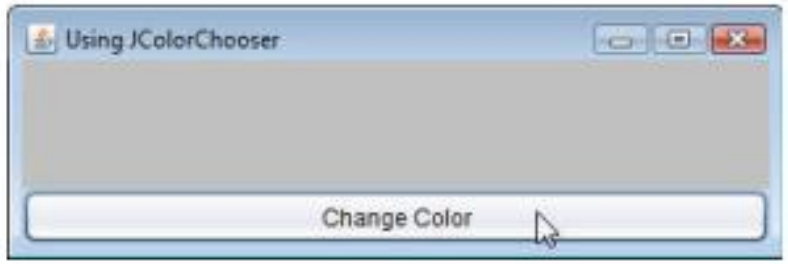
JColorChooser

- `JColorChooser.showDialog( Component, "Titulo do diálogo", corInicial )`

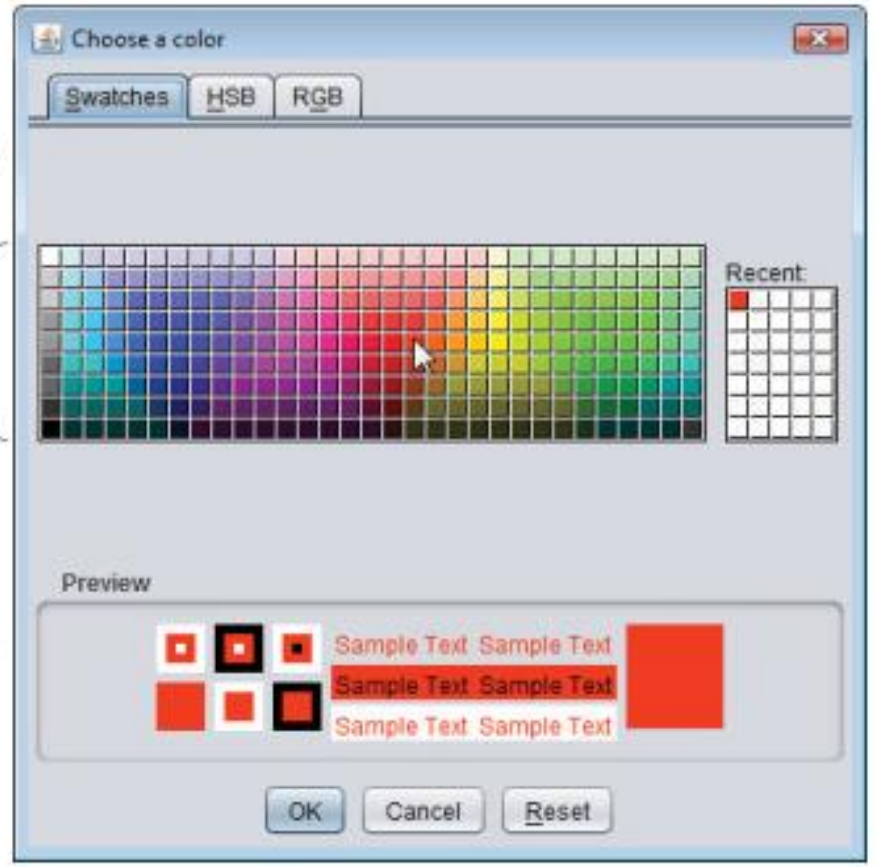


Componente **JColorChooser**

a) Janela inicial do aplicativo



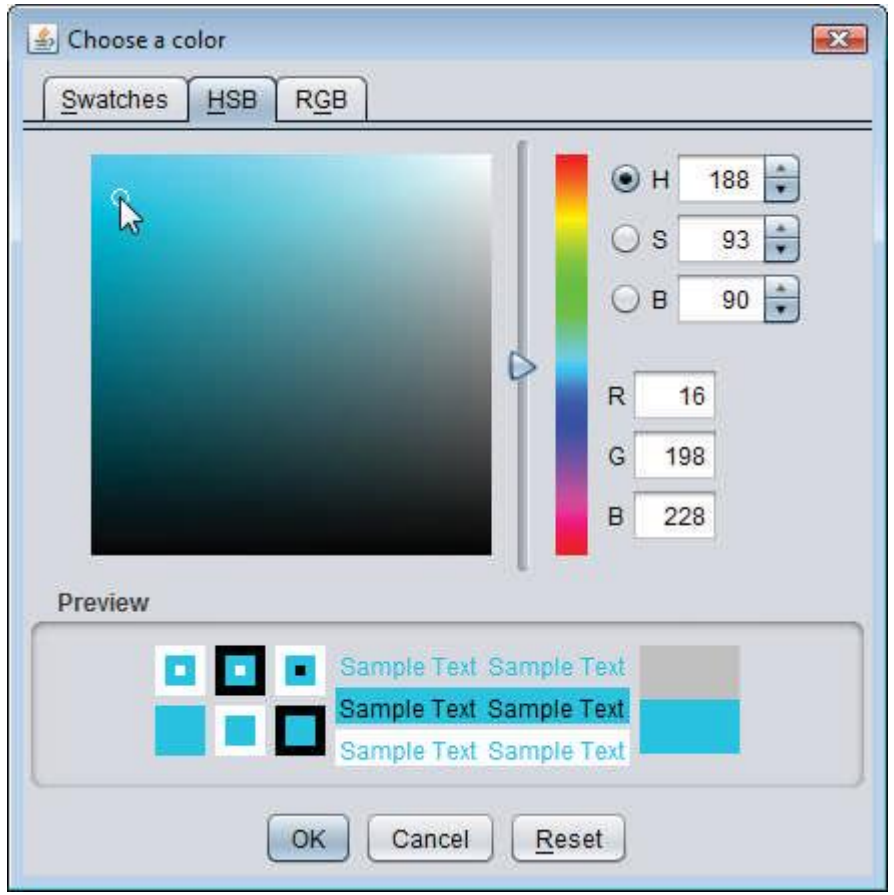
(b) Janela `JColorChooser`



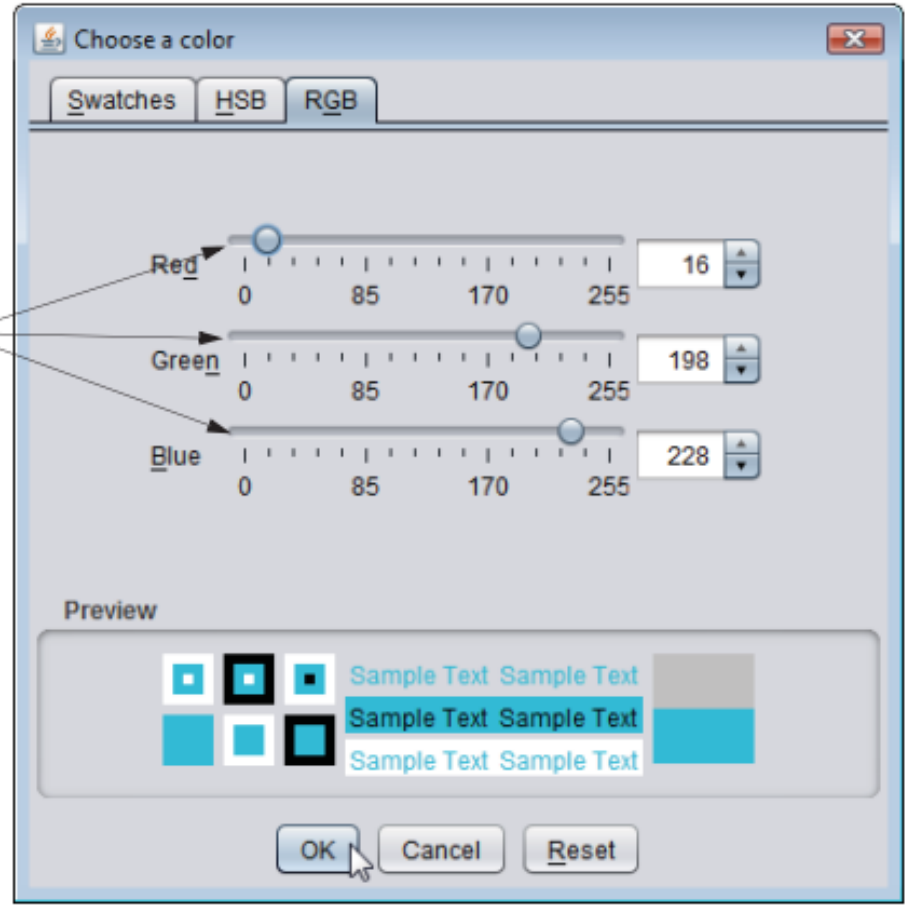
Selecione uma cor de uma das amostras de cor

(c) Janela do aplicativo depois de mudar a cor de fundo do `JPanel`

Componente **JColorChooser**



Controles deslizantes para selecionar as cores componentes vermelho, verde e azul



Componente **JColorChooser**

```
1 // Figura 13.7: 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 final JButton changeColorJButton;
15    private Color color = Color.LIGHT_GRAY;
16    private final JPanel colorJPanel;
17
```

Componente **JColorChooser**

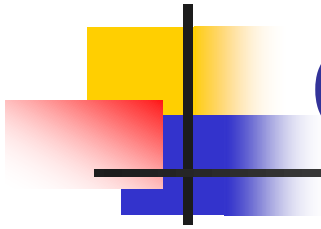
```
18    // configura a GUI
19    public ShowColors2JFrame()
20    {
21       super("Using JColorChooser");
22
23       // cria JPanel para exibir as cores
24       colorJPanel = new JPanel();
25       colorJPanel.setBackground(color);
26
27 // configura changeColorJButton e registra sua rotina de tratamento de evento
28       changeColorJButton = new JButton("Change Color");
29       changeColorJButton.addActionListener(
30             new ActionListener() // classe interna anônima
31             {
32                // exibe JColorChooser quando o usuário clica no botão
33                @Override
34                public void actionPerformed(ActionEvent event)
35                {
36                   color = JColorChooser.showDialog(
37                      ShowColors2JFrame.this, "Choose a color", color);
```

Componente **JColorChooser**

```
39                // configura a cor padrão, se nenhuma cor for retornada
40                   if (color == null)
41                      color = Color.LIGHT_GRAY;
42
43                   // muda a cor de fundo do painel de conteúdo
44                   colorJPanel.setBackground(color);
45                } // fim do método actionPerformed
46             } // fim da classe interna anônima
47          ); // fim da chamada para addActionListener
48
49       add(colorJPanel, BorderLayout.CENTER);
50       add(changeColorJButton, BorderLayout.SOUTH);
51
52       setSize(400, 130);
53       setVisible(true);
54    } // fim do construtor ShowColor2JFrame
55 } // fim da classe ShowColors2JFrame
```

Componente **JColorChooser**

```
1 // Figura 13.8: ShowColors2.java
2 // Escolhendo cores com JColorChooser.
3 import javax.swing.JFrame;
4
5  public class ShowColors2
6  {
7  // executa o aplicativo
8     public static void main(String[] args)
9     {
10          ShowColors2JFrame application = new ShowColors2JFrame();
11          application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 } // fim da classe ShowColors2
```



Controle **Fonts**

- `public final static int PLAIN`
- `public final static int BOLD`
- `public final static int ITALIC`
- `public Font( String name, int style, int size )`
- `public int getStyle()`
- `public int getSize()`
- `public String getName()`
- `public String getFamily()`

- `public boolean isPlain()`
- `public boolean isBold()`
- `public boolean isItalic()`

- `public Font getFont()`
- `public void setFont(Font f)`

Controle **Fonts**

Método ou constante	Descrição
Constantes, construtores e métodos de `Font`	
`public static final int PLAIN`	Uma constante representando um estilo de fonte simples.
`public static final int BOLD`	Uma constante representando um estilo de fonte negrito.
`public static final int ITALIC`	Uma constante representando um estilo de fonte itálico.
`public Font(String name, int style, int size)`	Cria um objeto `Font` com o nome, o estilo e o tamanho de fonte especificados.
`public int getStyle()`	Retorna um `int` indicando o estilo da fonte atual.
`public int getSize()`	Retorna um `int` indicando o tamanho da fonte atual.
`public String getName()`	Retorna o nome da fonte atual como uma string.
`public String getFamily()`	Retorna o nome da família de fontes como uma string.
`public boolean isPlain()`	Retorna `true` se a fonte for simples, caso contrário `false`.
`public boolean isBold()`	Retorna `true` se a fonte for negrito, caso contrário `false`.
`public boolean isItalic()`	Retorna `true` se a fonte for itálica, caso contrário `false`.
Métodos `Graphics` *para manipular* `Fonts`	
`public Font getFont()`	Retorna uma referência de objeto `Font` representando a fonte atual.
`public void setFont(Font f)`	Configura a fonte atual como a fonte, o estilo e o tamanho especificados pela referência de objeto `Font f`.

Controle **Fonts**


```
1 // Figura 13.11: 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    @Override
12    public void paintComponent(Graphics g)
13    {
14       super.paintComponent(g);
16       // configura fonte com Serif (Times), negrito, 12 pt
17       g.setFont(new Font("Serif", Font.BOLD, 12));
18       g.drawString("Serif 12 point bold.", 20, 30);
```

Controle **Fonts**

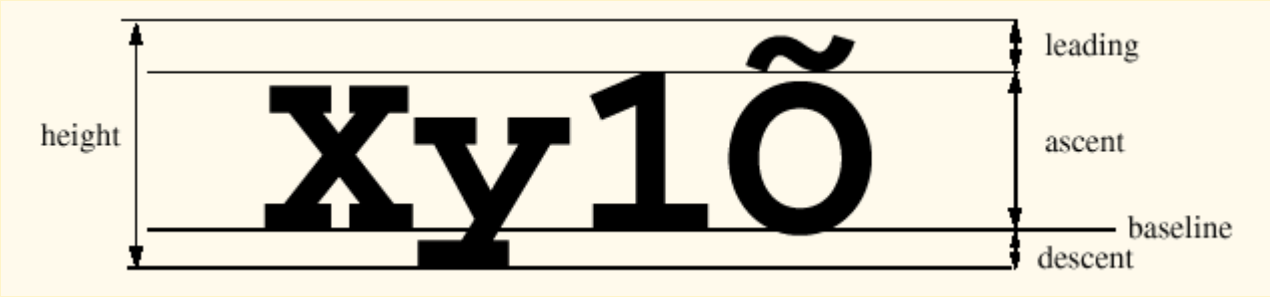

```
20        // define a fonte como Monospaced (Courier), itálico, 24 pt
21        g.setFont(new Font("Monospaced", Font.ITALIC, 24));
22        g.drawString("Monospaced 24 point italic.", 20, 50);
23
24        // define a fonte como SansSerif (Helvetica), simples, 14 pt
25        g.setFont(new Font("SansSerif", Font.PLAIN, 14));
26        g.drawString("SansSerif 14 point plain.", 20, 70);
27
28        // configura fonte com Serifa (Times), 18 pt, negrito/itálico
29        g.setColor(Color.RED);
30        g.setFont(new Font("Serif", Font.BOLD + Font.ITALIC, 18));
31        g.drawString(g.getFont().getName() + " " + g.getFont().getSize()
32           + " point bold italic.", 20, 90);
33     }
34  } // fim da classe FontJPanel
```

Controle **Fonts**


```
1 // Figura 13.12: Fonts.java
2 // Utilizando fontes.
3 import javax.swing.JFrame;
4
5 public class Fonts
6 {
8    public static void main(String[] args) // executa o aplicativo
9    {
10      // cria frame para FontJPanel
11      JFrame frame = new JFrame("Using fonts");
12      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13
15      frame.add(new FontJPanel());
16      frame.setSize(420, 150);
17      frame.setVisible(true);
18   }
19 } // fim da classe Fonts
```

Classe FontMetrics

- `public int getAscent()`
- `public int getDescent()`
- `public int getLeading()`
- `public int getHeight()`



- `public FontMetrics getFontMetrics()`
- `public FontMetrics getFontMetrics( Font f )`

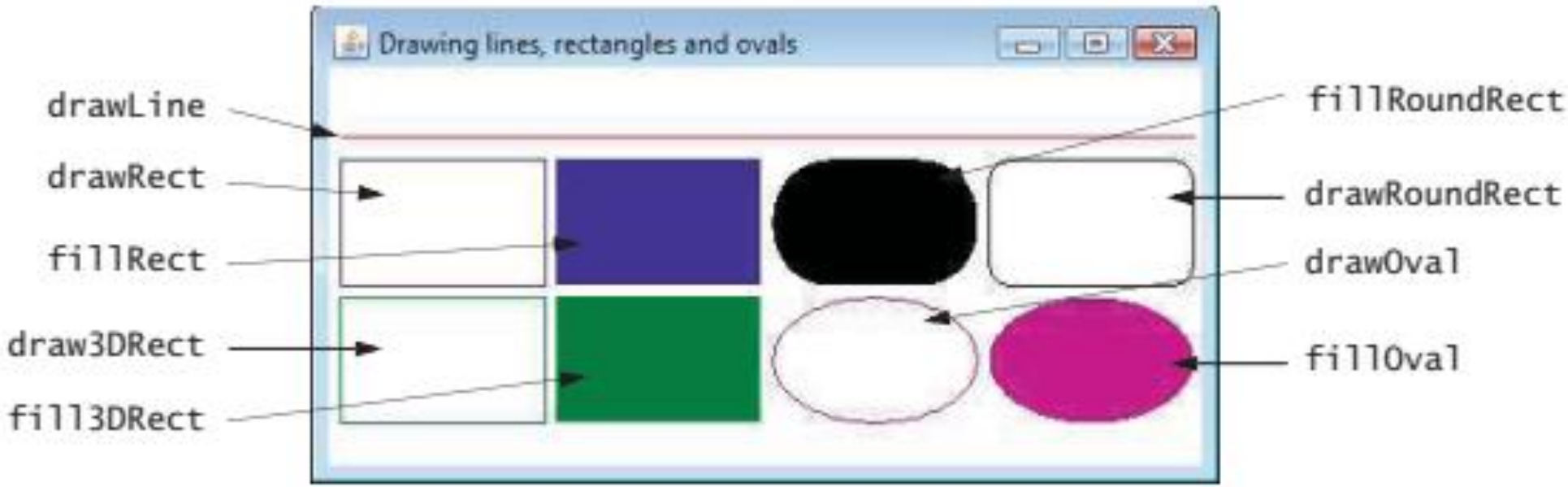
Desenhando Linhas, Retângulos e Ovais

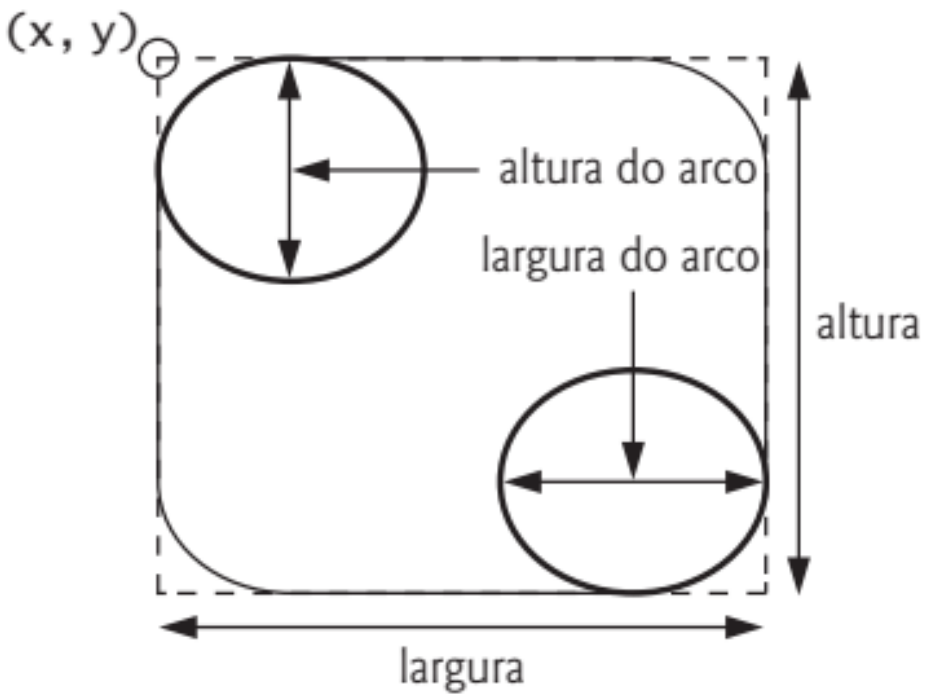
- `public void drawLine(int x1, int y1, int x2, int y2 )`
- `public void drawRect(int x, int y, int width, int height )`
- `public void clearRect(int x, int y, int width, int height )`
- `public void fillRect(int x, int y, int width, int height )`
- `public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight )`

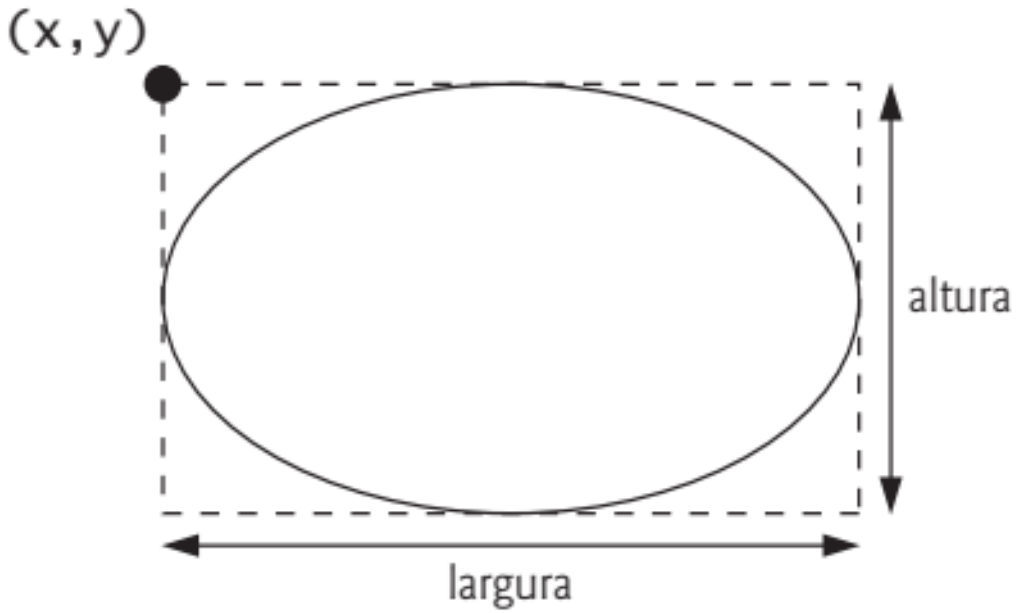
Desenhando Linhas, Retângulos e Ovais

- `public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`
- `public void draw3DRect( int x, int y, int width, int height, boolean b )`
- `public void fill3DRect( int x, int y, int width, int height, boolean b )`
- `public void drawOval( int x, int y, int width, int height )`
- `public void fillOval( int x, int y, int width, int height )`

Desenhando Linhas, Retângulos e Ovais

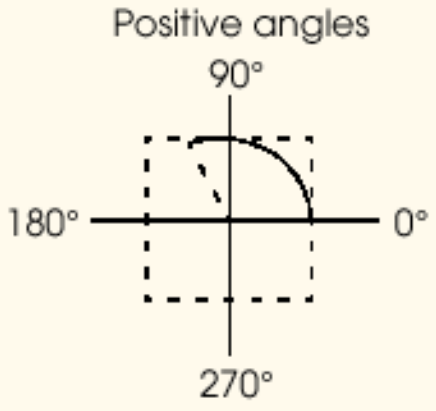


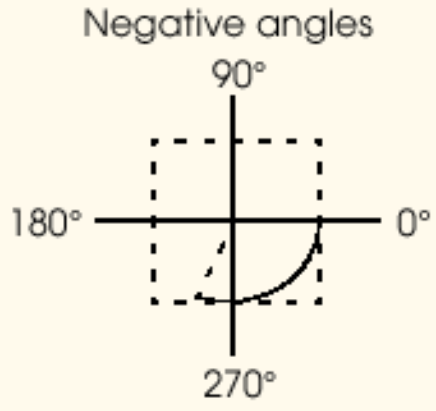

(x, y)
altura do arco
largura do arco
altura
largura

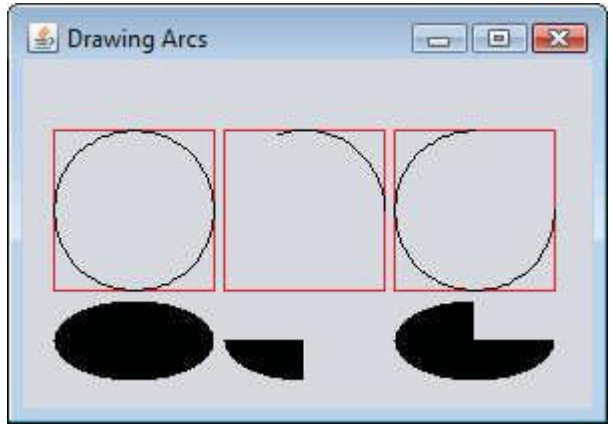

(x,y)
altura
largura

Desenhando Arcos

- `public void drawArc( int x, int y, int width, int height, int startAngle, int arcAngle )`
- `public void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle )`



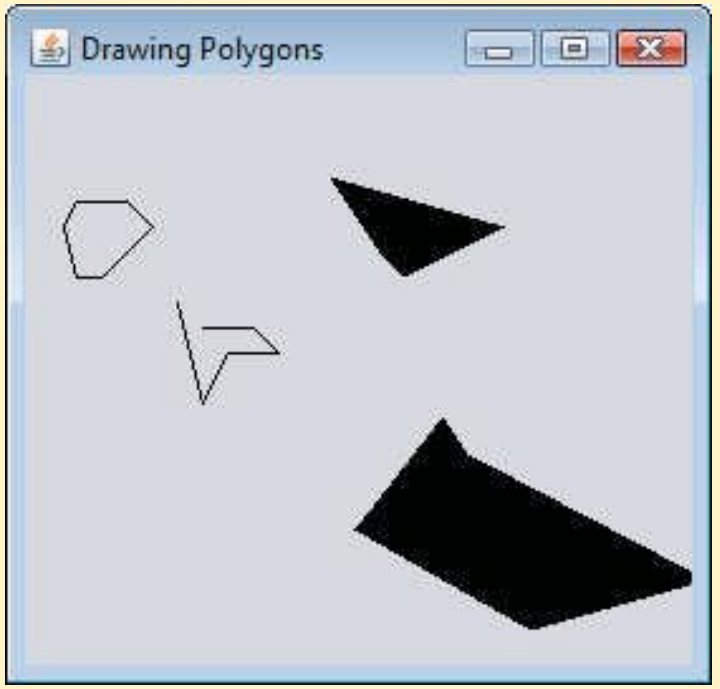




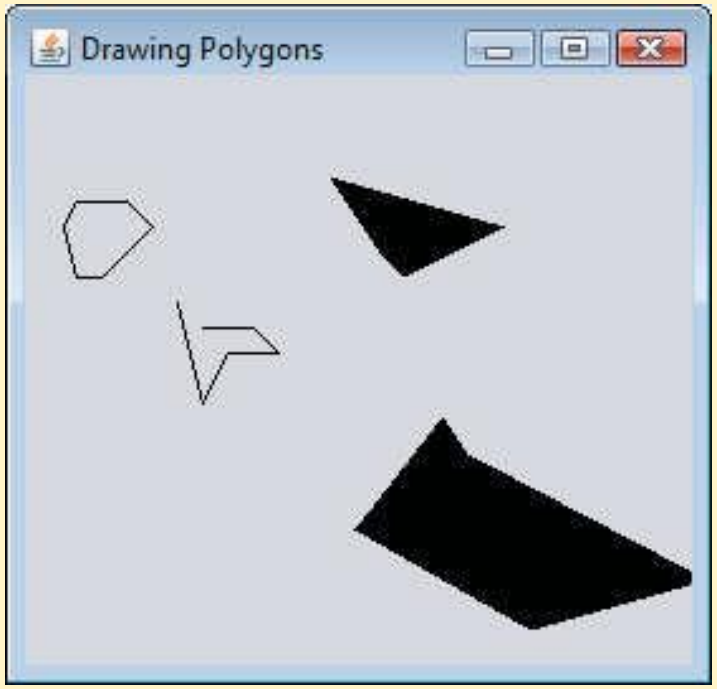
Desenhando Polígonos e Poli-linhas

- `public void drawPolygon( int xPoints[], int yPoints[], int points )`
- `public void drawPolyline( int xPoints[], int yPoints[], int points )`
- `public void drawPolygon( Polygon p )`
- `public void fillPolygon( int xPoints[], int yPoints[], int points )`
- `public void fillPolygon( Polygon p )`

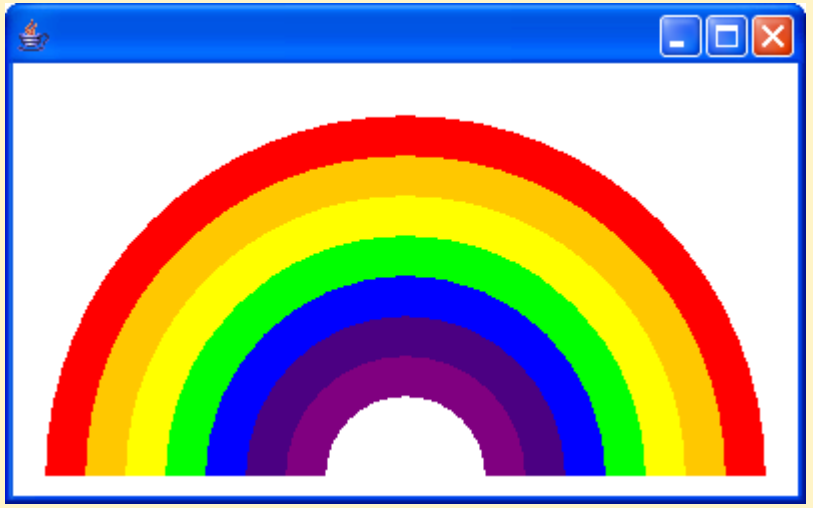
```java
// Figura 13.27: PolygonsJPanel.java
// Desenhando polígonos.
import java.awt.Graphics;
import java.awt.Polygon;
import javax.swing.JPanel;

public class PolygonsJPanel extends JPanel
{
   // desenha polígonos e polilinhas
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      // desenha o polígono com objeto Polygon
      int[] xValues = {20, 40, 50, 30, 20, 15};
      int[] yValues = {50, 50, 60, 80, 80, 60};
      Polygon polygon1 = new Polygon(xValues, yValues, 6);
      g.drawPolygon(polygon1);

      // desenha polilinhas com dois arrays
      int[] xValues2 = {70, 90, 100, 80, 70, 65, 60};
      int[] yValues2 = {100, 100, 110, 110, 130, 110, 90};
      g.drawPolyline(xValues2, yValues2, 7);

      // preenche o polígono com dois arrays
      int[] xValues3 = {120, 140, 150, 190};
      int[] yValues3 = {40, 70, 80, 60};
      g.fillPolygon(xValues3, yValues3, 4);

```



```
31       // desenha o polígono preenchido com objeto Polygon
32       Polygon polygon2 = new Polygon();
33       polygon2.addPoint(165, 135);
34       polygon2.addPoint(175, 150);
35       polygon2.addPoint(270, 200);
36       polygon2.addPoint(200, 220);
37       polygon2.addPoint(130, 180);
38       g.fillPolygon(polygon2);
39    }
40 } // fim da classe PolygonsJPanel
```

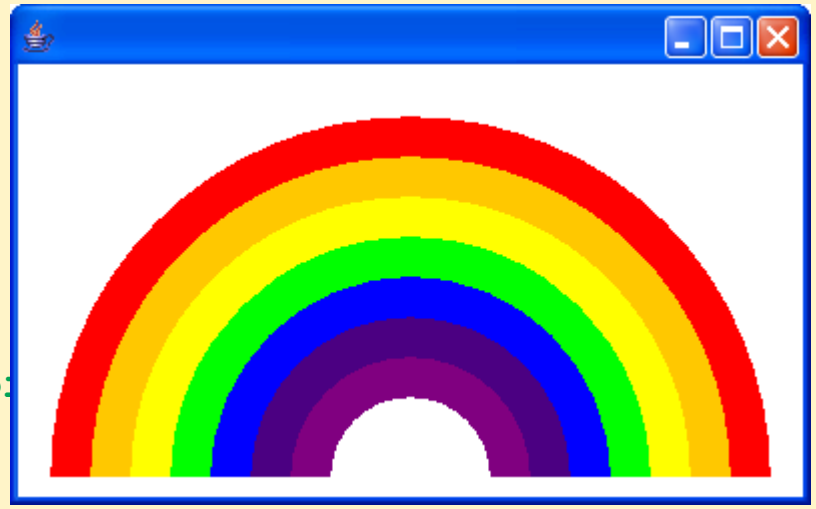



```
1 // Figura 13.28: DrawPolygons.java
2 // Desenhando polígonos.
3 import javax.swing.JFrame;
4
5 public class DrawPolygons
6 {
7    // executa o aplicativo
8    public static void main(String[] args)
9    {
10      // cria frame para PolygonsJPanel
11      JFrame frame = new JFrame("Drawing Polygons");
12      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13
14      PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
15      frame.add(polygonsJPanel);
16      frame.setSize(280, 270);
17      frame.setVisible(true);
18   }
19 } // fim da classe DrawPolygons
```



```
// Fig. 7.22: DrawRainbow.java
// Demonstra a utilização de cores em um array.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawRainbow extends JPanel
{
   // Define as cores índigo e violeta
   final Color VIOLET = new Color( 128, 0, 128 );
   final Color INDIGO = new Color( 75, 0, 130 );

   // a utilizar no arco-íris, iniciando da parte mais interna
   // As duas entradas em branco resultam em um arco vazio no centro
   private Color colors[] =
      { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
        Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };

   // construtor
   public DrawRainbow()
   {
      setBackground( Color.WHITE ); // configura o fundo como branco
   } // fim do construtor DrawRainbow
```

```
   // desenha um arco-íris utilizando círculos concêntricos
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      int radius = 20; // raio de um arco

      // desenha o arco-íris perto da parte central inferio
      int centerX = getWidth() / 2;
      int centerY = getHeight() - 10;

      // desenha arcos preenchidos com o mais externo
      for ( int counter = colors.length; counter > 0; counter-- )
      {
         // configura a cor para o arco atual
         g.setColor( colors[ counter - 1 ] );

         // preenche o arco de 0 a 180 graus
         g.fillArc( centerX - counter * radius,
            centerY - counter * radius,
            counter * radius * 2, counter * radius * 2, 0, 180 );
      } // for final
   } // fim do método paintComponent
} // fim da classe DrawRainbow
```



Exemplo - DrawSmiley

```
// Fig. 6.16: DrawSmiley.java
// Demonstrates filled shapes.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawSmiley extends JPanel
{
   public void paintComponent(
   Graphics g )
   {
      super.paintComponent( g );

      // draw the face
      g.setColor( Color.YELLOW );
      g.fillOval( 10, 10, 200, 200 );

      // draw the eyes
      g.setColor( Color.BLACK );
      g.fillOval( 55, 65, 30, 30 );
      g.fillOval( 135, 65, 30, 30 );

     // draw the mouth
      g.fillOval( 50, 110, 120, 60 );

   // "touch up" the mouth into a smile
      g.setColor( Color.YELLOW );
      g.fillRect( 50, 110, 120, 30 );
      g.fillOval( 50, 120, 120, 40 );
   } // end method paintComponent
} // end class DrawSmiley
```

Exemplo - DrawSmiley

```
// Figura 6.12: DrawSmileyTest.java
// Aplicativo de teste que exibe um rosto sorridente.
import javax.swing.JFrame;

public class DrawSmileyTest
{
      public static void main(String[] args)
      {
            DrawSmiley panel = new DrawSmiley();
            JFrame janela = new JFrame();
            janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            janela.add(panel);
            janela.setSize(230, 250);
            janela.setVisible(true);
      }
 } // fim da classe DrawSmileyTest
```

API Java2D

- Para acessar as funcionalidades da biblioteca **Graphics2D** devemos obter sua referencia a partir de um contexto gráfico.

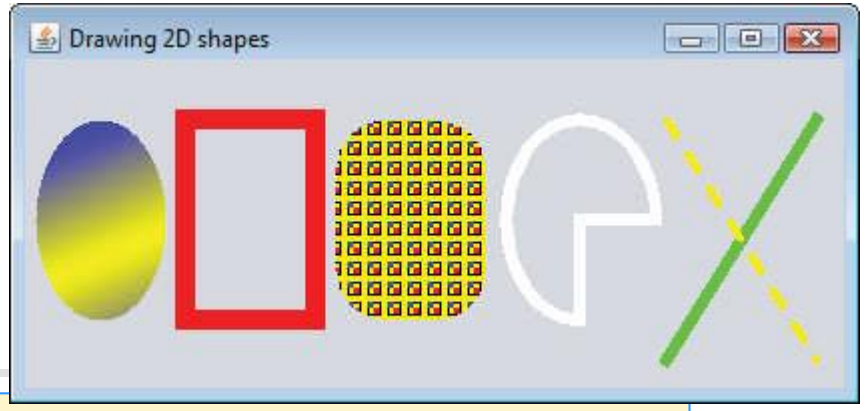
- `Graphics2D g2d = ( Graphics2D ) g`

- Métodos de Graphics2D
  - `g2d.translate(x, y)`
  - `g2d.rotate(radianos)`
  - `g2d.setPaint( Color )`
  - `g2d.draw(obj)`
  - `g2d.fill(obj)`
  - `g2d.setStroke(..)`

Java2D **Shapes**

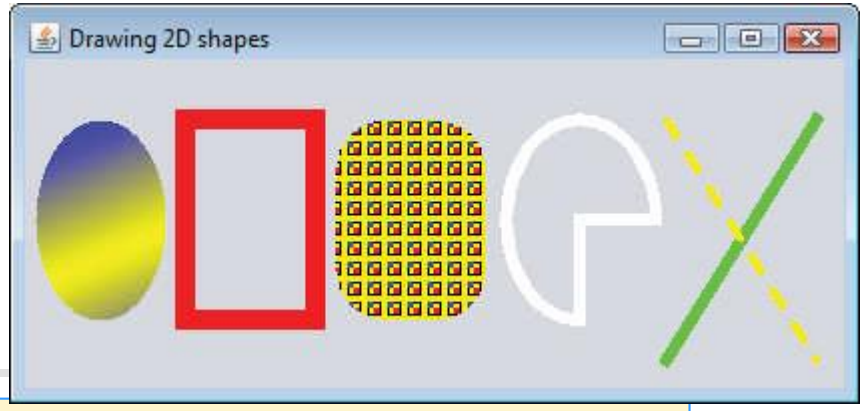
```
1 // Figura 13.29: ShapesJPanel.java
2 // Demonstrando algumas formas 2D Java.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.BasicStroke;
6 import java.awt.GradientPaint;
7 import java.awt.TexturePaint;
8 import java.awt.Rectangle;
9 import java.awt.Graphics2D;
10 import 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 {
```

Java2D **Shapes**



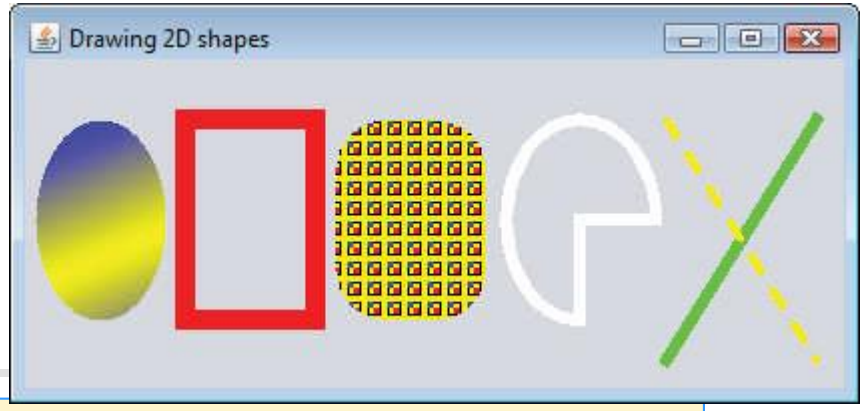

```
20    // desenha formas com Java 2D API
21    @Override
22    public void paintComponent(Graphics g)
23    {
24       super.paintComponent(g);
25       Graphics2D g2d = (Graphics2D) g; // casting g para Graphics2D
26
27       // desenha oval 2D preenchida com um gradiente azul-amarelo
28       g2d.setPaint(new GradientPaint(5, 30, Color.BLUE, 35, 100,
29                                      Color.YELLOW, true));
30       g2d.fill(new Ellipse2D.Double(5, 30, 65, 100));
31
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));
36
37       // desenha retâng. arred. 2D com um fundo armazenado em buffer
38       BufferedImage buffImage = new BufferedImage(10, 10,
39                                 BufferedImage.TYPE_INT_RGB);
```

Java2D **Shapes**



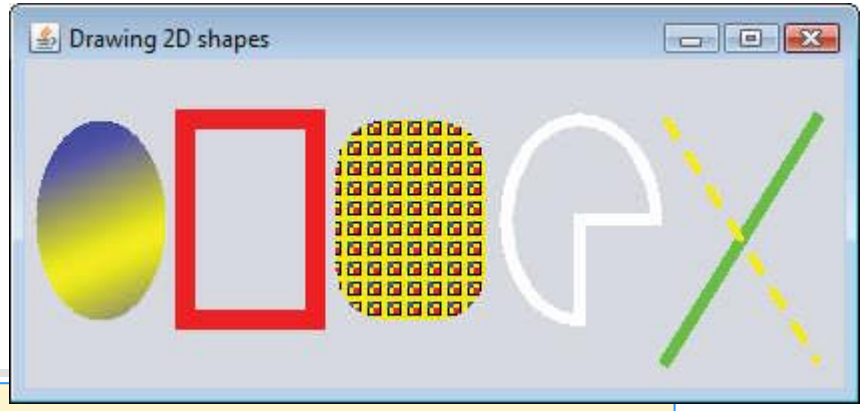

```
40
41          // obtém Graphics2D de buffImage e desenha nela
42          Graphics2D gg = buffImage.createGraphics();
43          gg.setColor(Color.YELLOW);
44          gg.fillRect(0, 0, 10, 10);
45          gg.setColor(Color.BLACK);
46          gg.drawRect(1, 1, 6, 6);
47          gg.setColor(Color.BLUE);
48          gg.fillRect(1, 1, 3, 3);
49          gg.setColor(Color.RED);
50          gg.fillRect(4, 4, 3, 3); // desenha um retângulo preenchido
51
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, 100, 50, 50));
57
```

Java2D **Shapes**

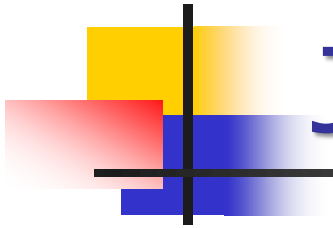



```
58        // Desenha arco 2D em forma de torta em branco
59        g2d.setPaint(Color.WHITE);
60        g2d.setStroke(new BasicStroke(6.0f));
61        g2d.draw(
62              new Arc2D.Double(240, 30, 75, 100, 0, 270, Arc2D.PIE));
63
64        // Desenha linhas 2D em verde e amarelo
65        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     }
75  } // fim da classe ShapesJPanel
```

Java2D Shapes




```
1 // Figura 13.30: Shapes.java
2 // Testando ShapesJPanel.
3 import javax.swing.JFrame;
5 public class Shapes
6 {
7     // executa o aplicativo
8     public static void main(String[] args)
9     {
10        // cria 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);
18        frame.setSize(425, 200);
19        frame.setVisible(true);
20    }
21 } // fim da classe Shapes
```



Java2D **Shapes**

- **`java.awt.geom`**
  - **`Ellipse2D.Double`**
  - **`Rectangle2D.Double`**
  - **`RoundRectangle2D.Double`**
  - **`Arc2D.Double`**
  - **`Line2D.Double`**

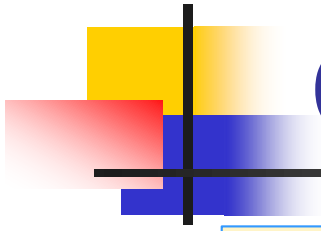
API Java2D

- `GradientPaint( xi, yi, CorI, xf, yf, CorF, cicl )`

- `BufferedImage(lagura, altura, BufferedImage.TYPE_INT_RGB )`
- `buffImage.createGraphics();`

- `TexturePaint(buffImage, new Rectangle( lagura, altura ) )`

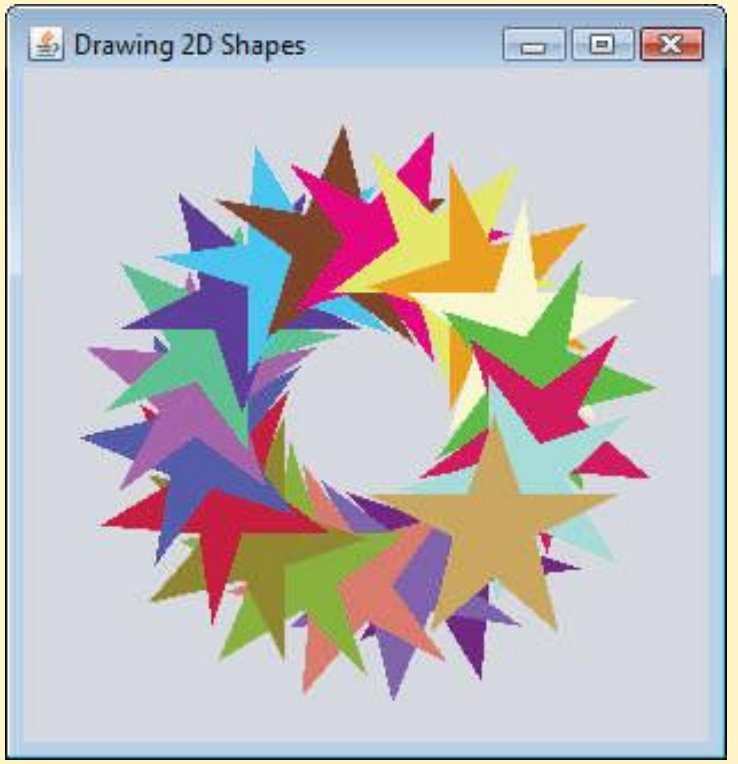
- `BasicStroke( 4, BasicStroke.CAP_ROUND,`
  `BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );`



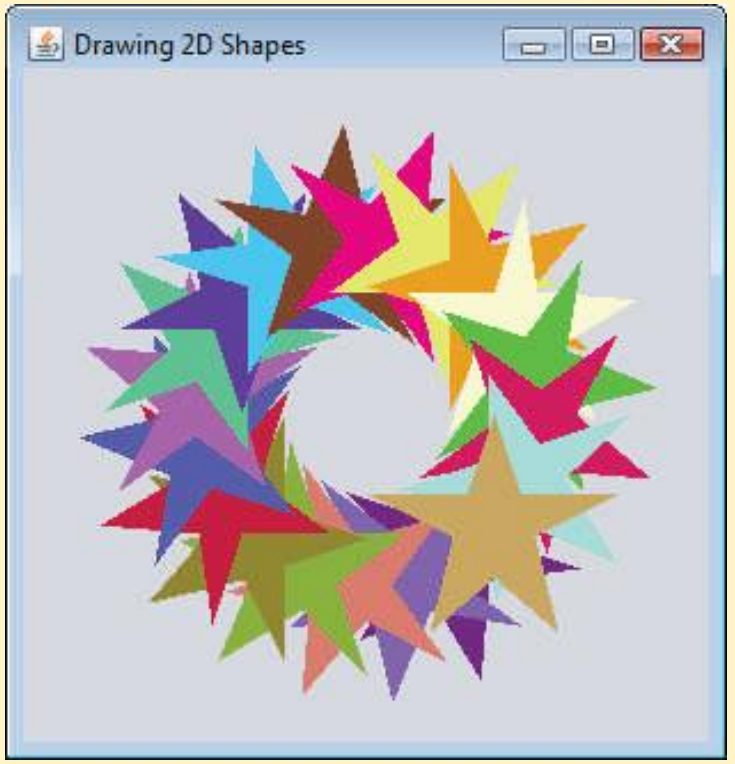
Classe GeneralPath

- Podemos construir uma *Shape* personalizada com a classe **GeneralPath;**
- Utilizamos os métodos:
  - moveTo(x , y)
  - lineTo (x , y)
  - curveTo(x1, y1, x2, y2, x3, y3)
  - quadTo(x1, y1, x2, y2)
  - closePath()

```
// Fig. 13.31: Shapes2JPanel.java
// Demonstrando um caminho geral.
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.GeneralPath;
import java.util.Random;
import javax.swing.JPanel;

public class Shapes2JPanel extends JPanel
{
   // desenha caminhos gerais
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );          // chama o paintComponent da superclasse
      Random random = new Random();       // obtém o gerador de números aleatórios

      int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
      int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };

      Graphics2D g2d = ( Graphics2D ) g;
      GeneralPath star = new GeneralPath(); // cria o objeto GeneralPath
```



```
      // configura a coordenada inicial do General Path
      star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );

      // cria a estrela -- isso não desenha a estrela
      for ( int count = 1; count < xPoints.length; count++ )
         star.lineTo( xPoints[ count ], yPoints[ count ] );

      star.closePath(); // fecha a forma

      g2d.translate( 200, 200 ); // translada a origem para

      // gira em torno da origem e desenha estrelas em cores
      for ( int count = 1; count <= 20; count++ )
      {
         g2d.rotate( Math.PI / 10.0 ); // rotaciona o sistema de coordenadas

         // configura cores aleatórias
         g2d.setColor( new Color( random.nextInt( 256 ),
            random.nextInt( 256 ), random.nextInt( 256 ) ) );

         g2d.fill( star );              // desenha estrela preenchida
      } // for final
   } // fim do método paintComponent
} // fim da classe Shapes2JPanel
```



Timer

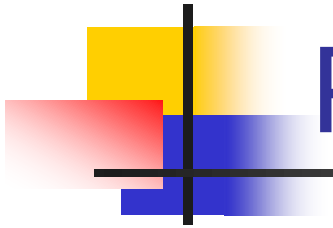
- Classe **Timer** cria um contador que uma vez iniciado gera um evento de ação em intervalos de tempo regulares.

- Exemplo

```
Timer t = new Timer (1000, new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {  repaint(); }
   } );
t.start();
```

Exercício

- Criar um programa em Java para desenhar círculos concêntricos usando duas cores alternadas como o desenho da figura abaixo:



Referencias

- Java How to program 3, 4, 5 , 6 , 7 e 10 ed.

Deitel e Deitel

- Sun

http://java.sun.com