第15章
Introduction to Java Programming, 8E

图　形

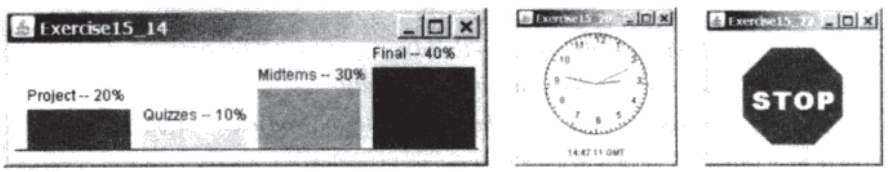
学习目标

- 描述GUI组件中的Java坐标系（15.2节）。
- 使用Graphics类中的方法画图（15.3节）。
- 覆盖paintComponent方法在GUI组件上绘画（15.3节）。
- 使用面板作画布来绘画（15.3节）。
- 绘制字符串、直线、矩形、椭圆、弧形和多边形等（15.4、15.6～15.7节）。
- 使用FontMetrics获取字体属性并且了解如何将消息居中（15.8节）。
- 在GUI组件中显示一个图像（15.11节）。
- 开发可重用的GUI组件FigurePanel、MessagePanel、StillClock和ImageViewer（15.5、15.9、15.10、15.12节）

15.1 引言

假设希望画出如图15-1所示的像条形图、时钟或停止符号这样的图形，如何才能做到呢？

本章描述如何使用Graphics类中的方法绘制字符串、直线、矩形、椭圆、弧形、多边形和图像，以及如何开发可重用的GUI组件。



a)　　b)　　c)

图15-1　可以使用Graphics类中的绘图方法绘制图形

15.2 图形坐标系

要绘图，首先需要确定在哪里画。每个组件都有自己的坐标系，原点（0，0）在组件的左上角。*x*坐标向右增加，*y*坐标向下增加。注意，Java的坐标系不同于传统的坐标系，如图15-2所示。

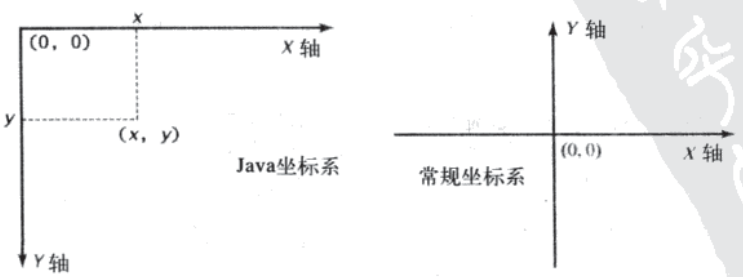


图15-2　Java图形坐标系的计量单位是像素，点（0，0）在它的左上角

在父组件c2（例如面板）中的组件c1（例如按钮）的左上角位置可以使用c1.getX()和c1.getY()进行定位。如图15-3所示，(x1, y1) = (c1.getX(), c1.getY())，(x2, y2) = (c2.getX(), c2.getY()) 以及 (x3, y3) = (c3.getX(), c3.getY())。

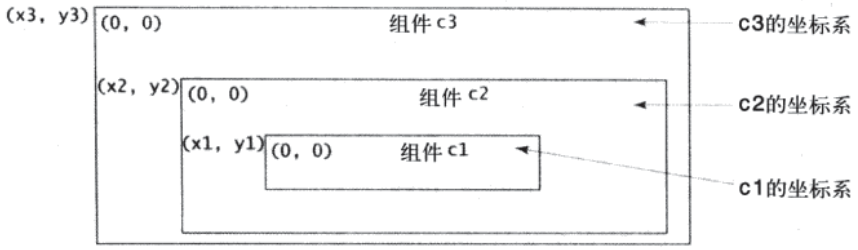


图15-3 每个GUI组件都有自己的坐标系

15.3 Graphics类

Graphics类中提供了绘制字符串、直线、矩形、椭圆、弧形、多边形和折线段的方法，如图15-4所示。

java.awt.Graphics	
+setColor(color: Color): void	为后续绘图设置新的颜色
+setFont(font: Font): void	为后续绘图设置新的字体
+drawString(s: String, x: int, y: int): void	绘制从点（x, y）开始的字符串
+drawLine(x1: int, y1: int, x2: int, y2: int): void	绘制从（x1, y1）到（x2, y2）的一条直线
+drawRect(x: int, y: int, w: int, h: int): void	绘制一个左上角在点（x, y）处、宽为w、高为h的矩形
+fillRect(x: int, y: int, w: int, h: int): void	绘制一个左上角在点（x, y）处、宽为w、高为h的填充矩形
+drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void	绘制一个圆弧宽度为aw、高度为ah的圆角矩形
+fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void	绘制一个圆弧宽度为aw、高度为ah的填充圆角矩形
+draw3DRect(x: int, y: int, w: int, h: int, raised: boolean): void	绘制一个从表面凸起或者凹进的3D矩形
+fill3DRect(x: int, y: int, w: int, h: int, raised: boolean): void	绘制一个从表面凸起或者凹进的填充3D矩形
+drawOval(x: int, y: int, w: int, h: int): void	绘制一个椭圆，椭圆的外接矩形由参数x、y、w和h确定
+fillOval(x: int, y: int, w: int, h: int): void	绘制一个填充椭圆，填充椭圆的外接矩形由参数x、y、w和h确定
+drawArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void	绘制一个圆弧，该圆弧是外接矩形由参数x、y、w和h确定的椭圆的一部分
+fillArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void	绘制一个填充圆弧，该圆弧是外接矩形由参数x、y、w和h确定的椭圆的一部分
+drawPolygon(xPoints: int[], yPoints: int[], nPoints: int): void	绘制一个x坐标和y坐标构成的数组所定义的闭合多边形。每一对坐标（x[i], y[i]）表示一个点
+fillPolygon(xPoints: int[], yPoints: int[], nPoints: int): void	绘制一个x坐标和y坐标构成的数组所定义的填充多边形。每一对坐标（x[i], y[i]）表示一个点
+drawPolygon(g: Polygon): void	绘制一个由Polygon对象定义的闭合多边形
+fillPolygon(g: Polygon): void	绘制一个由Polygon对象定义的填充多边形
+drawPolyline(xPoints: int[], yPoints: int[], nPoints: int): void	绘制一个x坐标和y坐标构成的数组所定义的折线。每一对坐标（x[i], y[i]）表示一个点

图15-4 Graphics类包含用于绘制字符串和图形的方法

可以将GUI组件看作一张纸，而将Graphics看作铅笔或画刷。可以应用Graphics类中的方法在GUI组件上进行绘画。

Graphics类是一个提供与设备无关的图形界面的抽象类，它可以在不同平台的屏幕上显示图形和图像。任何时候需要显示组件（例如，按钮、标签和面板）时，JVM都会自动在本地平台上为该组件创建一个Graphics对象，然后传递这个对象来调用paintComponent方法来显示图画。

paintComponent方法的签名如下所示：

```
protected void paintComponent(Graphics g)
```

这个定义在JComponent类中的方法是在第一次显示组件或重新显示组件的时候调用的。

为了在组件上绘图，需要定义一个扩展JPanel的类，并且覆盖它的paintComponent方法来表明绘制什么。程序清单15-1给出在面板上绘制一条线和一个字符串的例子，如图15-5所示。

程序清单15-1 TestPaintComponent.java

```
 1 import javax.swing.*;
 2 import java.awt.Graphics;
 3
 4 public class TestPaintComponent extends JFrame {
 5   public TestPaintComponent() {
 6     add(new NewPanel());
 7   }
 8
 9   public static void main(String[] args) {
10     TestPaintComponent frame = new TestPaintComponent();
11     frame.setTitle("TestPaintComponent");
12     frame.setSize(200, 100);
13     frame.setLocationRelativeTo(null); // Center the frame
14     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15     frame.setVisible(true);
16   }
17 }
18
19 class NewPanel extends JPanel {
20   protected void paintComponent(Graphics g) {
21     super.paintComponent(g);
22     g.drawLine(0, 0, 50, 50);
23     g.drawString("Banner", 0, 40);
24   }
25 }
```

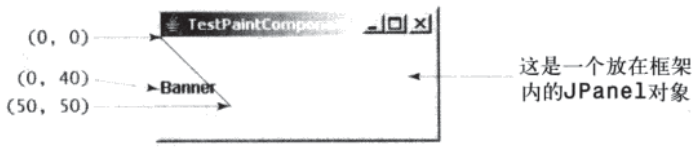


图15-5 在面板上绘制一条线和一个字符串

当第一次显示组件或者任何时候需要重新显示组件时，都会自动调用方法paintComponent来绘制图像。调用super.paintComponent(g)（第21行）来调用父类中定义的paintComponent方法。必须要确保在显示新的图画之前清空视图区域。第22行调用drawLine方法来绘制一条从（0，0）到（50，50）的直线。第23行调用drawString方法来绘制一个字符串。

所有的绘图方法都有表明在什么地方绘制对象的参数。在Java中所有的计量单位都是像素。字符串“Banner”绘制在位置（0，40）。

JVM调用paintComponent在组件上进行绘画。用户永远都不要直接调用paintComponent。因为这个原因，将paintComponent的可见性设置为protected就足够了。

面板是不可见的，它们用作一个小型的容器，这个容器将组件进行分组获得所需的布局。JPanel的另一个重要应用是绘画。可以在任何一个Swing GUI组件上绘画，但是，通常应该使用JPanel作为画布在其上绘图。如果如下所示在第19行使用JLabel替换JPanel，那会发生什么呢？

```
class NewPanel extends JLabel {
```

程序仍可以工作，但是不推荐这样做，因为JLabel设计为创建一个标签，而不是为了绘画。为保持统一性，本书将通过JPanel的子类来定义画布类。

提示 一些教科书通过JComponent的子类来定义画布类。这里的问题是如果要在画布上设置背景色，就必须写绘制背景色的代码。一个简单的setBackground(Color color)方法是不能在JComponent中设置背景色的。

15.4 绘制字符串、直线、矩形和椭圆

方法drawString(String s,int x,int y)绘制以点（x，y）为起点的字符串，如图15-6a所示。

方法drawLine(int x1,int y1,int x2,int y2)绘制从点（x1，y1）到点（x2，y2）的直线段，如图15-6b所示。

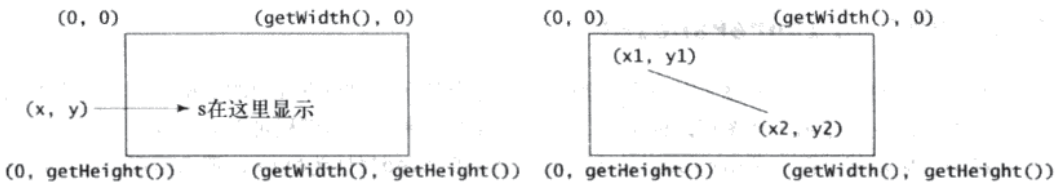


图15-6 a）方法drawString（s，x，y）绘制一个从点（x，y）开始的字符串；
b）方法drawString（x1，y1，x2，y2）绘制在两个定点之间的线段

Java提供六种绘制空心矩形或填充颜色的矩形的方法。可以绘制或填充直角矩形、圆角矩形或三维矩形。

drawRect(int x,int y,int w,int h)方法绘制一个直角矩形，而fillRect(int x, int y, int w,int h)方法绘制一个填充颜色的矩形。参数x和y表示这个矩形的左上角，w和h表示这个矩形的宽和高（参见图15-7）。

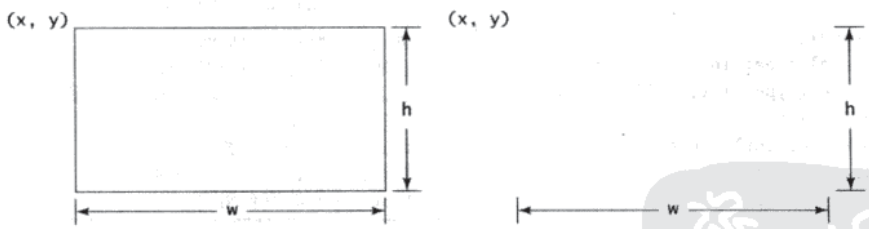


图15-7 a）方法drawRect（x，y，w，h）绘制一个矩形；
b）方法fillRect（x，y，w，h)绘制填充颜色的矩形

方法drawRoundRect(int x,int y,int w,int h,int aw,int ah)绘制一个圆角矩形，而方法fillRoundRect(int x,int y,int w,int h,int aw,int ah)绘制一个填充颜色的圆角矩形。参数x、y、w和h的含义与drawRect方法中的参数含义一样，参数aw指角上圆弧的水平直径，ah指角上圆弧的垂直直径（参见图15-8a）。换句话说，aw和ah是每个角上四分之一椭圆的宽和高。

方法draw3DRect(int x,int y,int w,int h,boolean raised)绘制一个三维矩形，方法fill3DRect(int x,int y,int w,int h,boolean raised)绘制一个填充颜色的三维矩形。参数x、y、w和h的含义与drawRect方法的参数含义相同。最后一个参数是布尔值，表示矩形是从表面凸起还是凹进。

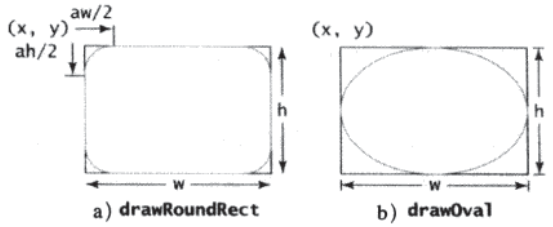


图15-8 a）方法drawRoundRect(x,y,w,h,aw,ah)绘制一个圆角矩形；
b）方法drawOval(x,y,w,h)绘制一个基于它的外接矩形的椭圆

根据要绘制的是空心椭圆还是填充椭圆，可以使用drawOval(int x,int y,int w,int h)方法或者fillOval(int x,int y,int w,int h)方法。椭圆是根据它的外接矩形绘制的。参数x和y表示外接矩形左上角的坐标，w和h分别表示外接矩形的宽和高，如图15-8b所示。

15.5 实例学习：FigurePanel类

这个例子开发可以显示各种图形的非常有用的类。这个类允许用户设置图的类型、确定是否填充该图形以及是否在面板上显示这个图形。该类的UML图如图15-9所示。该面板可以显示直线段、矩形、圆角矩形和椭圆。显示哪个图形是由type属性决定的。如果filled属性为true，那么矩形、圆角矩形和椭圆在面板上都是填充颜色的。

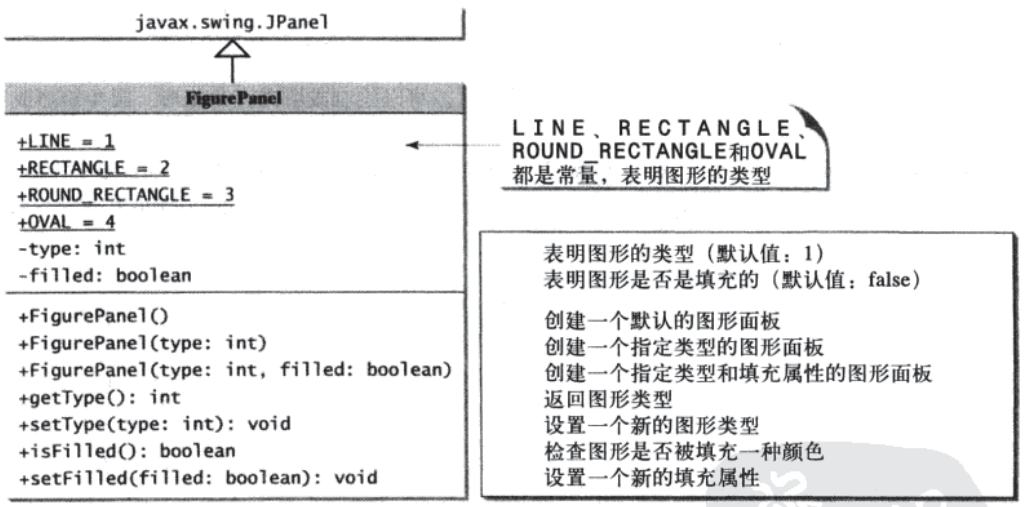


图15-9 FigurePanel在面板上显示各种类型的图形

这个UML图作为FigurePanel类的合约。使用者可以在不知道这个类是如何实现的情况下使用它。我们从编写程序清单15-2中的程序开始介绍，该程序使用这个类显示6个图形面板，如图15-10所示。

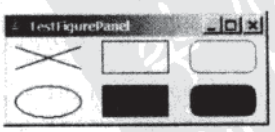


图15-10 创建六个FigurePanel对象显示六个图形

程序清单15-3实现FigurePanel类。四个常量——LINE、RECTANGLE、ROUND_RECTANGLE和OVAL，都是在第6～9行声明的。这四种图形都是依照type属性绘制的（第37行）。setColor方法（第39、44、53、62行）为图设置新的颜色。

程序清单15-2 **TestFigurePanel.java**

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class TestFigurePanel extends JFrame {
5   public TestFigurePanel() {
6     setLayout(new GridLayout(2, 3, 5, 5));
7     add(new FigurePanel(FigurePanel.LINE));
8     add(new FigurePanel(FigurePanel.RECTANGLE));
9     add(new FigurePanel(FigurePanel.ROUND_RECTANGLE));
10    add(new FigurePanel(FigurePanel.OVAL));
11    add(new FigurePanel(FigurePanel.RECTANGLE, true));
12    add(new FigurePanel(FigurePanel.ROUND_RECTANGLE, true));
13  }
14
15  public static void main(String[] args) {
16    TestFigurePanel frame = new TestFigurePanel();
17    frame.setSize(400, 200);
18    frame.setTitle("TestFigurePanel");
19    frame.setLocationRelativeTo(null); // Center the frame
20    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21    frame.setVisible(true);
22  }
23 }
```

程序清单15-3 **FigurePanel.java**

```
1 import java.awt.*;
2 import javax.swing.JPanel;
3
4 public class FigurePanel extends JPanel {
5   // Define constants
6   public static final int LINE = 1;
7   public static final int RECTANGLE = 2;
8   public static final int ROUND_RECTANGLE = 3;
9   public static final int OVAL = 4;
10
11  private int type = 1;
12  private boolean filled = false;
13
14  /** Construct a default FigurePanel */
15  public FigurePanel() {
16  }
17
18  /** Construct a FigurePanel with the specified type */
19  public FigurePanel(int type) {
20    this.type = type;
21  }
22
23  /** Construct a FigurePanel with the specified type and filled */
24  public FigurePanel(int type, boolean filled) {
25    this.type = type;
26    this.filled = filled;
27  }
28
29  /** Draw a figure on the panel */
30  protected void paintComponent(Graphics g) {
31    super.paintComponent(g);
32
33    // Get the appropriate size for the figure
34    int width = getWidth();
35    int height = getHeight();
36
37    switch (type) {
38      case LINE: // Display two cross lines
```

```
39          g.setColor(Color.BLACK);
40          g.drawLine(10, 10, width - 10, height - 10);
41          g.drawLine(width - 10, 10, 10, height - 10);
42          break;
43        case RECTANGLE: // Display a rectangle
44          g.setColor(Color.BLUE);
45          if (filled)
46            g.fillRect((int)(0.1 * width), (int)(0.1 * height),
47              (int)(0.8 * width), (int)(0.8 * height));
48          else
49            g.drawRect((int)(0.1 * width), (int)(0.1 * height),
50              (int)(0.8 * width), (int)(0.8 * height));
51          break;
52        case ROUND_RECTANGLE: // Display a round-cornered rectangle
53          g.setColor(Color.RED);
54          if (filled)
55            g.fillRoundRect((int)(0.1 * width), (int)(0.1 * height),
56              (int)(0.8 * width), (int)(0.8 * height), 20, 20);
57          else
58            g.drawRoundRect((int)(0.1 * width), (int)(0.1 * height),
59              (int)(0.8 * width), (int)(0.8 * height), 20, 20);
60          break;
61        case OVAL: // Display an oval
62          g.setColor(Color.BLACK);
63          if (filled)
64            g.fillOval((int)(0.1 * width), (int)(0.1 * height),
65              (int)(0.8 * width), (int)(0.8 * height));
66          else
67            g.drawOval((int)(0.1 * width), (int)(0.1 * height),
68              (int)(0.8 * width), (int)(0.8 * height));
69      }
70    }
71
72    /** Set a new figure type */
73    public void setType(int type) {
74      this.type = type;
75      repaint();
76    }
77
78    /** Return figure type */
79    public int getType() {
80      return type;
81    }
82
83    /** Set a new filled property */
84    public void setFilled(boolean filled) {
85      this.filled = filled;
86      repaint();
87    }
88
89    /** Check if the figure is filled */
90    public boolean isFilled() {
91      return filled;
92    }
93
94    /** Specify preferred size */
95    public Dimension getPreferredSize() {
96      return new Dimension(80, 80);
97    }
98  }
```

repaint方法（第75、86行）是在Component类中定义的。调用repaint方法会导致paintComponent方法被调用。调用repaint方法以刷新视图区域。一般情况下，如果要显示新的东西，就应该调用这个方法。

警告 永远都不要直接调用paintComponent方法。它应该在视图区域改变时由JVM调用或者由repaint方法调用。应该覆盖paintComponent方法告诉系统如何绘制视图区域，但永远都不要覆盖repaint方法。

注意 repaint方法提出更新视图区域的请求并且立即返回。它的效果是异步的，这就意味着，它由JVM决定在独立的线程上执行paintComponent方法。

在FigurePanel类中覆盖定义在Component类中的getPreferredSize()方法（第95～97行），指定合适的尺寸，以便布局管理器考虑什么时候放置一个FigurePanel对象。根据布局管理器的规则，布局管理器可能考虑也可能不考虑这一属性。例如，在FlowLayout管理器的容器中，组件可以使用希望的尺寸，但是如果放在GridLayout管理器的容器中，组件希望的尺寸可能会被忽略。最好的方法就是覆盖在JPanel子类中的getPreferredSize()方法以确定希望的尺寸，因为默认情况下JPanel的尺寸是0×0。

15.6 绘制弧形

弧形可以认为是以矩形为边界的椭圆的一部分。绘制或者填充弧形的方法如下：

```
drawArc(int x, int y, int w, int h, int startAngle, int arcAngle);
fillArc(int x, int y, int w, int h, int startAngle, int arcAngle);
```

参数x、y、w和h的含义与drawOval方法中参数的含义是一样的；参数startAngle是起始角；arcAngle是跨度角（即弧线覆盖的角）。角的单位是度，遵循通常的数学习惯（即0度指向东边，并且从东边开始沿逆时针方向旋转的角度为正角），参见图15-11。

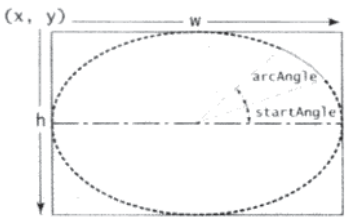


图15-11 drawArc方法绘制一个基于特定角的椭圆的弧形

程序清单15-4是一个如何绘制弧形的例子，它的输出如图15-12所示。

程序清单15-4 DrawArcs.java

```
 1 import javax.swing.JFrame;
 2 import javax.swing.JPanel;
 3 import java.awt.Graphics;
 4
 5 public class DrawArcs extends JFrame {
 6   public DrawArcs() {
 7     setTitle("DrawArcs");
 8     add(new ArcsPanel());
 9   }
10
11   /** Main method */
12   public static void main(String[] args) {
13     DrawArcs frame = new DrawArcs();
14     frame.setSize(250, 300);
15     frame.setLocationRelativeTo(null); // Center the frame
16     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17     frame.setVisible(true);
18   }
19 }
20
```

```
21 // The class for drawing arcs on a panel
22 class ArcsPanel extends JPanel {
23   // Draw four blades of a fan
24   protected void paintComponent(Graphics g) {
25     super.paintComponent(g);
26
27     int xCenter = getWidth() / 2;
28     int yCenter = getHeight() / 2;
29     int radius = (int)(Math.min(getWidth(), getHeight()) * 0.4);
30
31     int x = xCenter - radius;
32     int y = yCenter - radius;
33
34     g.fillArc(x, y, 2 * radius, 2 * radius, 0, 30);
35     g.fillArc(x, y, 2 * radius, 2 * radius, 90, 30);
36     g.fillArc(x, y, 2 * radius, 2 * radius, 180, 30);
37     g.fillArc(x, y, 2 * radius, 2 * radius, 270, 30);
38   }
39 }
```

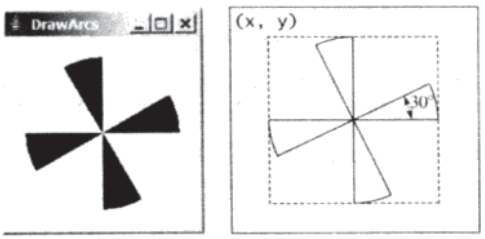
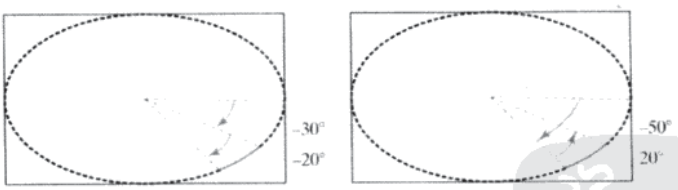


图15-12 程序绘制四个填充颜色的弧形

角度可以是负的。负起始角是从东边开始沿顺时针方向旋转，如图15-13所示。负跨度角是从起始角开始顺时针转动的角度。下面的两条语句绘制同一段弧形：

```
g.fillArc(x, y, 2 * radius, 2 * radius, -30, -20);
g.fillArc(x, y, 2 * radius, 2 * radius, -50, 20);
```

- 语句使用负的起始角-30和负的跨度角-20，如图15-13a所示。第二条语句使用负的起始角-50和正的跨度角20，如图15-13b所示。



a）起始角为-30度而跨度角为-20度　　b）起始角为-50度而跨度角为20度

图15-13 角度可以是负的

15.7 绘制多边形和折线段

为了绘制一个多边形，首先需要使用多边形类Polygon来创建一个Ploygon对象，如图15-14所示。

多边形是一个封闭的二维区域。这个区域由任意多条线段围成，每条线段都是多边形的一条边。在内部，多边形由坐标对（x，y）的序列组成，每对坐标定义多边形的一个顶点，两对相邻坐标对就是多边形一条边的两个端点。第一对点和最后一对点由封闭多边形的线段连接。

下面是用于创建Polygon对象并给多边形增加一个点的例子：

```
Polygon polygon = new Polygon();
polygon.addPoint(40, 20);
polygon.addPoint(70, 40);
polygon.addPoint(60, 80);
polygon.addPoint(45, 45);
polygon.addPoint(20, 60);
```

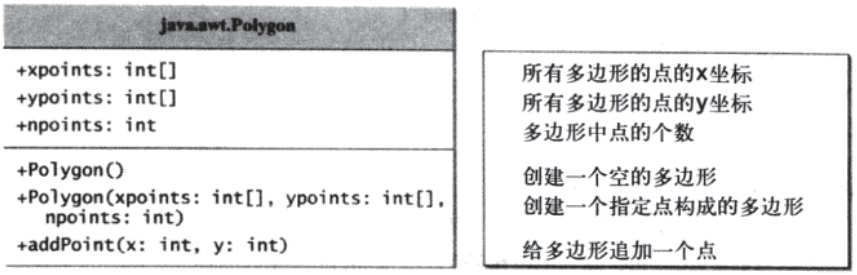


图15-14 Polygon类对多边形建模

在添加完这些点之后，xpoints是 {40, 70, 60, 45, 20}，ypoint是{20, 40, 80, 45, 60}，npoints是5。xpoint、ypoint和npoint都是Polygon中的公共数据域，这是一个不好的设计。原则上讲，所有的数据域都应该是私有的。

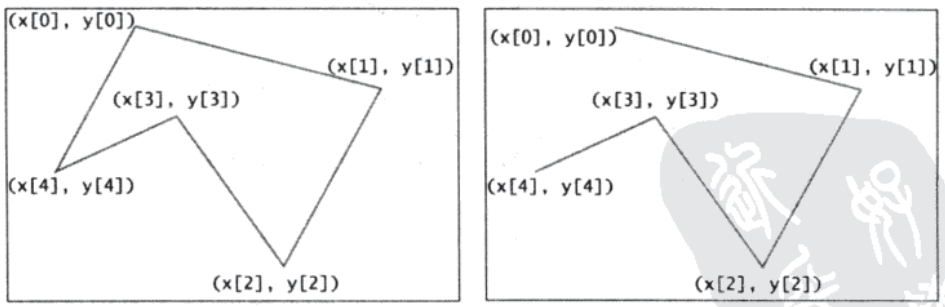
为了绘制或填充一个多边形，使用下面的Graphics类中的方法之一：

```
drawPolygon(Polygon polygon);
fillPolygon(Polygon polygon);
drawPolygon(int[] xpoints, int[] ypoints, int npoints);
fillPolygon(int[] xpoints, int[] ypoints, int npoints);
```

例如，

```
int x[] = {40, 70, 60, 45, 20};
int y[] = {20, 40, 80, 45, 60};
g.drawPolygon(x, y, x.length);
```

该绘制方法通过绘制点（x[i], y[i]）与点（x[i+1], y[i+1]）之间的线段打开多边形，其中i=0, 1, 2, ..., x.length−1；通过绘制第一个点和最后一个点之间的连线封闭多边形（参见图15-15a）。



a）多边形 b）折线段

图15-15 drawPolygon方法绘制一个多边形，drawPolyline绘制一条折线段

要绘制一条折线段，可以使用drawPolyline(int[] x, int[] y, int nPoints)方法，它绘制出由x坐标和y坐标数组定义的相互连接的一系列线段。例如，下面的代码画出了图15-15b所示的折线段。

```
int x[] = {40, 70, 60, 45, 20};
int y[] = {20, 40, 80, 45, 60};
g.drawPolyline(x, y, x.length);
```

程序清单15-5是一个如何绘制六边形的例子，输出结果如图15-16所示。

程序清单15-5 DrawPolygon.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Polygon;

public class DrawPolygon extends JFrame {
  public DrawPolygon() {
    setTitle("DrawPolygon");
    add(new PolygonsPanel());
  }

  /** Main method */
  public static void main(String[] args) {
    DrawPolygon frame = new DrawPolygon();
    frame.setSize(200, 250);
    frame.setLocationRelativeTo(null); // Center the frame
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}

// Draw a polygon in the panel
class PolygonsPanel extends JPanel {
  protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    int xCenter = getWidth() / 2;
    int yCenter = getHeight() / 2;
    int radius = (int)(Math.min(getWidth(), getHeight()) * 0.4);

    // Create a Polygon object
    Polygon polygon = new Polygon();

    // Add points to the polygon in this order
    polygon.addPoint(xCenter + radius, yCenter);
    polygon.addPoint((int)(xCenter + radius *
      Math.cos(2 * Math.PI / 6)), (int)(yCenter - radius *
      Math.sin(2 * Math.PI / 6)));
    polygon.addPoint((int)(xCenter + radius *
      Math.cos(2 * 2 * Math.PI / 6)), (int)(yCenter - radius *
      Math.sin(2 * 2 * Math.PI / 6)));
    polygon.addPoint((int)(xCenter + radius *
      Math.cos(3 * 2 * Math.PI / 6)), (int)(yCenter - radius *
      Math.sin(3 * 2 * Math.PI / 6)));
    polygon.addPoint((int)(xCenter + radius *
      Math.cos(4 * 2 * Math.PI / 6)), (int)(yCenter - radius *
      Math.sin(4 * 2 * Math.PI / 6)));
    polygon.addPoint((int)(xCenter + radius *
      Math.cos(5 * 2 * Math.PI / 6)), (int)(yCenter - radius *
      Math.sin(5 * 2 * Math.PI / 6)));

    // Draw the polygon
    g.drawPolygon(polygon);
  }
}
```

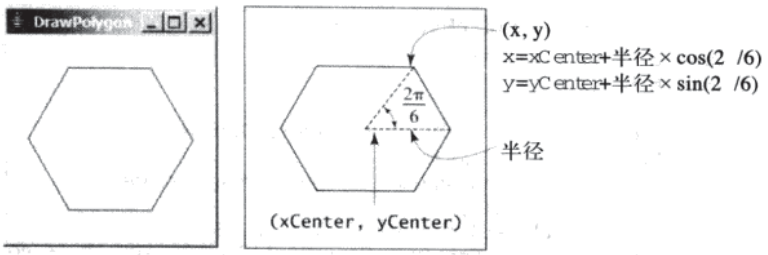


图15-16 程序使用drawPolygon方法绘制一个多边形

15.8 使用FontMetrics类居中显示字符串

可以在一个面板的任何位置显示字符串。可以居中显示它吗？要想做到这一点，需要使用FontMetrics类，对特定字体的字符串测量出确切的宽度和高度。FontMetrics可以测量给定字体的以下属性（如图15-17所示）：

- Leading（发音为ledding）是文本行之间的距离。
- Ascent表示字符从基线到上升线的距离。字体中多数字符的顶点都在上升线之下，但是也有一些可能会延伸到上升线以上。
- Descent是从基线到下沉线的距离。字体中大多数降字（例如，j、y和g）都在下沉线之上，但是也有一些可能会延伸到下沉线以下。
- Height是leading、ascent和descent的和。

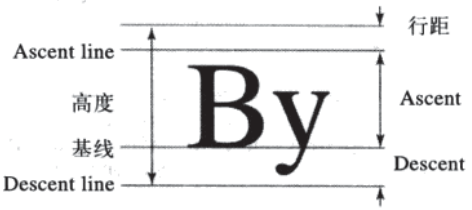


图15-17 FontMetrics类可以被用来确定给定字体的字符的字体属性

FontMetrics是一个抽象类。要得到给定字体的FontMetrics对象，可以使用定义在Graphics类中的以下getFontMetrics方法：

- **public FontMetrics getFontMetrics (Font font)**

返回指定字体的字体尺寸。

- **public FontMetrics getFontMetrics ()**

返回当前字体的字体尺寸。

可以使用下面的FontMetrics类中的实例方法得到字体属性，以及使用这种字体绘制的字符串的宽度：

```
public int getAscent() // Return the ascent
public int getDescent() // Return the descent
public int getLeading() // Return the leading
public int getHeight() // Return the height
public int stringWidth(String str) // Return the width of the string
```

程序清单15-6给出在面板中央显示消息的例子，如图15-18所示。

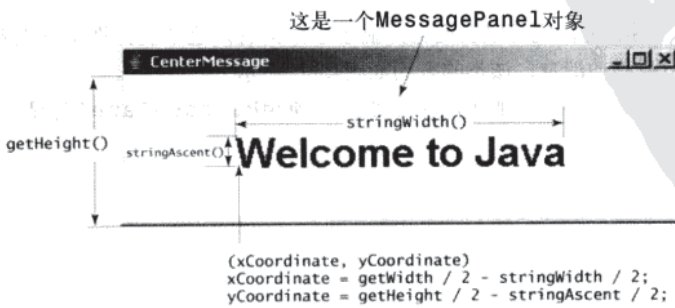


图15-18 程序使用FontMetrics类测量字符串的宽度和高度，并在框架中央显示该字符串

程序清单15-6 TestCenterMessage.java

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class TestCenterMessage extends JFrame{
5   public TestCenterMessage() {
6     CenterMessage messagePanel = new CenterMessage();
7     add(messagePanel);
8     messagePanel.setBackground(Color.WHITE);
9     messagePanel.setFont(new Font("Californian FB", Font.BOLD, 30));
10  }
11
12  /** Main method */
13  public static void main(String[] args) {
14    TestCenterMessage frame = new TestCenterMessage();
15    frame.setSize(300, 150);
16    frame.setLocationRelativeTo(null); // Center the frame
17    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18    frame.setVisible(true);
19  }
20 }
21
22 class CenterMessage extends JPanel {
23   /** Paint the message */
24   protected void paintComponent(Graphics g) {
25     super.paintComponent(g);
26
27     // Get font metrics for the current font
28     FontMetrics fm = g.getFontMetrics();
29
30     // Find the center location to display
31     int stringWidth = fm.stringWidth("Welcome to Java");
32     int stringAscent = fm.getAscent();
33
34     // Get the position of the leftmost character in the baseline
35     int xCoordinate = getWidth() / 2 - stringWidth / 2;
36     int yCoordinate = getHeight() / 2 + stringAscent / 2;
37
38     g.drawString("Welcome to Java", xCoordinate, yCoordinate);
39   }
40 }
```

在Component类中定义的方法getWidth()和getHeight()（第35～36行），分别返回组件的宽度和长度。

由于消息是centered，所以字符串的第一个字符应该放在（xCoordinate，yCoordinate）处，如图15-18所示。

15.9 实例学习：MessagePanel类

本实例开发一个有用的类，它可以在面板中显示一条消息。这个类允许用户设置消息的位置、居中放置消息、使用指定间距移动消息。该类的合约如图15-19所示。

我们从编写程序清单15-7中的测试程序开始，它使用MessagePanel类显示四个消息面板，如图15-20所示。

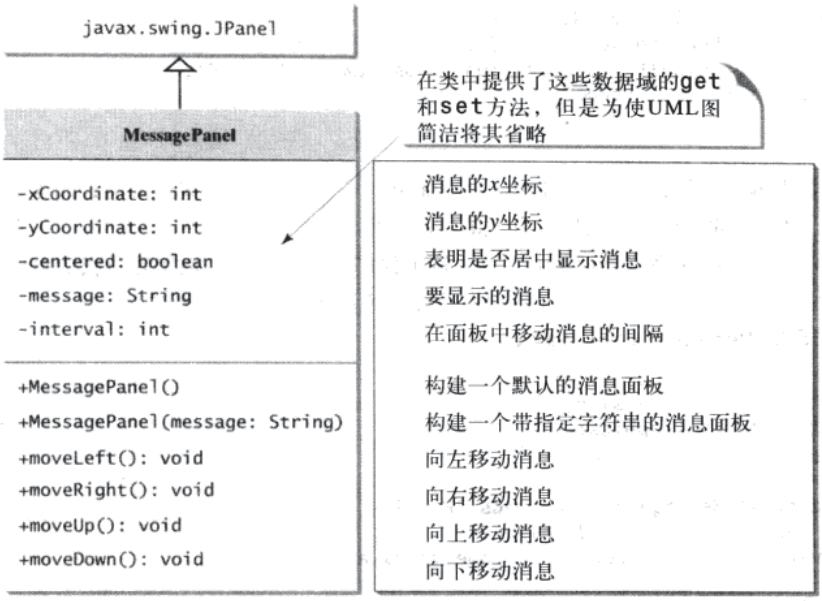


图15-19 MessagePanel类在面板上显示消息

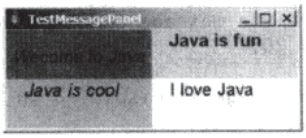


图15-20 TestMessagePanel使用MessagePanel显示四个消息面板

程序清单15-7 TestMessagePanel.java

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class TestMessagePanel extends JFrame {
5    public TestMessagePanel() {
6      MessagePanel messagePanel1 = new MessagePanel("Wecome to Java");
7      MessagePanel messagePanel2 = new MessagePanel("Java is fun");
8      MessagePanel messagePanel3 = new MessagePanel("Java is cool");
9      MessagePanel messagePanel4 = new MessagePanel("I love Java");
10     messagePanel1.setFont(new Font("SansSerif", Font.ITALIC, 20));
11     messagePanel2.setFont(new Font("Courier", Font.BOLD, 20));
12     messagePanel3.setFont(new Font("Times", Font.ITALIC, 20));
13     messagePanel4.setFont(new Font("Californian FB", Font.PLAIN, 20));
14     messagePanel1.setBackground(Color.RED);
15     messagePanel2.setBackground(Color.CYAN);
16     messagePanel3.setBackground(Color.GREEN);
17     messagePanel4.setBackground(Color.WHITE);
18     messagePanel1.setCentered(true);
19
20     setLayout(new GridLayout(2, 2));
21     add(messagePanel1);
22     add(messagePanel2);
23     add(messagePanel3);
24     add(messagePanel4);
25   }
26
27   public static void main(String[] args) {
28     TestMessagePanel frame = new TestMessagePanel();
29     frame.setSize(300, 200);
```

```
30      frame.setTitle("TestMessagePanel");
31      frame.setLocationRelativeTo(null); // Center the frame
32      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33      frame.setVisible(true);
34    }
35  }
```

本节的剩余内容就是解释如何实现MessagePanel类。因为可以使用这个类，而无须知道它是如何实现的，所以，如果你希望的话，可以跳过它的实现过程。

MessagePanel类在程序清单15-8中实现。这个程序看起来很长，但实际上很简单，因为大多数方法都是get方法和set方法，并且每个方法相对来说都很短，而且易于阅读。

程序清单15-8 MessagePanel.java

```
1  import java.awt.FontMetrics;
2  import java.awt.Dimension;
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class MessagePanel extends JPanel {
7    /** The message to be displayed */
8    private String message = "Welcome to Java";
9
10   /** The x-coordinate where the message is displayed */
11   private int xCoordinate = 20;
12
13   /** The y-coordinate where the message is displayed */
14   private int yCoordinate = 20;
15
16   /** Indicate whether the message is displayed in the center */
17   private boolean centered;
18
19   /** The interval for moving the message horizontally and
20     vertically */
21   private int interval = 10;
22
23   /** Construct with default properties */
24   public MessagePanel() {
25   }
26
27   /** Construct a message panel with a specified message */
28   public MessagePanel(String message) {
29     this.message = message;
30   }
31
32   /** Return message */
33   public String getMessage() {
34     return message;
35   }
36
37   /** Set a new message */
38   public void setMessage(String message) {
39     this.message = message;
40     repaint();
41   }
42
43   /** Return xCoordinator */
44   public int getXCoordinate() {
45     return xCoordinate;
46   }
47
48   /** Set a new xCoordinator */
49   public void setXCoordinate(int x) {
50     this.xCoordinate = x;
51     repaint();
```

```
52  }
53
54  /** Return yCoordinator */
55  public int getYCoordinate() {
56    return yCoordinate;
57  }
58
59  /** Set a new yCoordinator */
60  public void setYCoordinate(int y) {
61    this.yCoordinate = y;
62    repaint();
63  }
64
65  /** Return centered */
66  public boolean isCentered() {
67    return centered;
68  }
69
70  /** Set a new centered */
71  public void setCentered(boolean centered) {
72    this.centered = centered;
73    repaint();
74  }
75
76  /** Return interval */
77  public int getInterval() {
78    return interval;
79  }
80
81  /** Set a new interval */
82  public void setInterval(int interval) {
83    this.interval = interval;
84    repaint();
85  }
86
87  /** Paint the message */
88  protected void paintComponent(Graphics g) {
89    super.paintComponent(g);
90
91    if (centered) {
92      // Get font metrics for the current font
93      FontMetrics fm = g.getFontMetrics();
94
95      // Find the center location to display
96      int stringWidth = fm.stringWidth(message);
97      int stringAscent = fm.getAscent();
98      // Get the position of the leftmost character in the baseline
99      xCoordinate = getWidth() / 2 - stringWidth / 2;
100     yCoordinate = getHeight() / 2 + stringAscent / 2;
101   }
102
103   g.drawString(message, xCoordinate, yCoordinate);
104 }
105
106 /** Move the message left */
107 public void moveLeft() {
108   xCoordinate -= interval;
109   repaint();
110 }
111
112 /** Move the message right */
113 public void moveRight() {
114   xCoordinate += interval;
115   repaint();
116 }
```

```
117
118   /** Move the message up */
119   public void moveUp() {
120     yCoordinate -= interval;
121     repaint();
122   }
123
124   /** Move the message down */
125   public void moveDown() {
126     yCoordinate += interval;
127     repaint();
128   }
129
130   /** Override get method for preferredSize */
131   public Dimension getPreferredSize() {
132     return new Dimension(200, 30);
133   }
134 }
```

如果属性centered为true（第91行），那么paintComponent方法居中显示这条消息。message在第8行初始化为“Welcome to Java”。如果它没有初始化，当使用无参构造方法创建MessagePanel时就会导致一个NullPointerException的运行时错误，因为message在第103行会是null。

警告 MessagePanel类使用属性xCoordinate和yCoordinate指定消息在面板上显示的位置。不要使用属性名x和y，因为它们已经在Component类中定义，使用方法getX()和getY()返回上一级坐标系中组件的位置。

注意 Component类具有setBackground、setForeground和setFont方法。这些方法用来设置整个组件的颜色和字体。如果想用不同的颜色和字体在同一个面板中绘制几条消息，必须使用Graphics类中的setColor和setFont方法为当前的图形设定颜色和字体。

注意 Java程序设计的一个主要特征就是类的复用。贯穿本书，我们将开发可重用的类，然后在后面重用它们。MessagePanel就是一个这样的例子，程序清单10-2中的Loan和程序清单15-3中的FigurePanel都是这样的例子。无论什么时候需要在面板中显示一条消息，MessagePanel都可以重复使用。要使类在一个广泛的应用范围内可复用，应该提供多种使用它的方式。MessagePanel类提供了许多将在本书的很多例子中使用的属性和方法。下一节给出在面板上显示一个钟表图像的有用的且可重用的类。

15.10 实例学习：StillClock类

这个实例会开发一个在面板上显示时钟的类。该类的合约如图15-21所示。

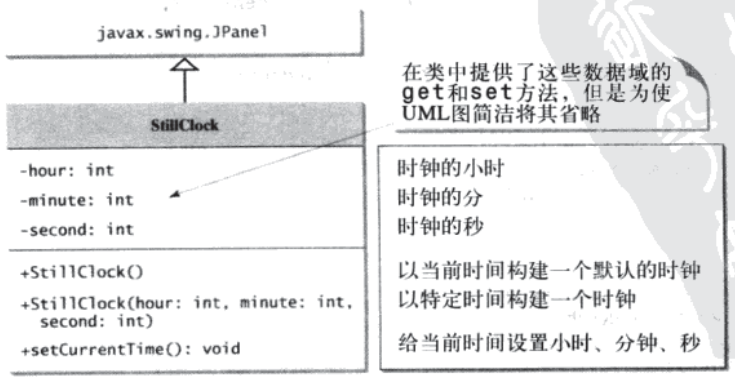


图15-21 StillClock类显示一个模拟时钟

我们首先编写程序清单15-9中的测试程序，程序使用StillClock类显示一个模拟时钟，然后使用MessagePanel类在面板中显示小时、分钟和秒，如图15-22a所示。

程序清单15-9 DisplayClock.java

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DisplayClock extends JFrame {
5   public DisplayClock() {
6     // Create an analog clock for the current time
7     StillClock clock = new StillClock();
8
9     // Display hour, minute, and second in the message panel
10    MessagePanel messagePanel = new MessagePanel(clock.getHour() +
11      ":" + clock.getMinute() + ":" + clock.getSecond());
12    messagePanel.setCentered(true);
13    messagePanel.setForeground(Color.blue);
14    messagePanel.setFont(new Font("Courier", Font.BOLD, 16));
15
16    // Add the clock and message panel to the frame
17    add(clock);
18    add(messagePanel, BorderLayout.SOUTH);
19  }
20
21  public static void main(String[] args) {
22    DisplayClock frame = new DisplayClock();
23    frame.setTitle("DisplayClock");
24    frame.setSize(300, 350);
25    frame.setLocationRelativeTo(null); // Center the frame
26    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27    frame.setVisible(true);
28  }
29 }
```

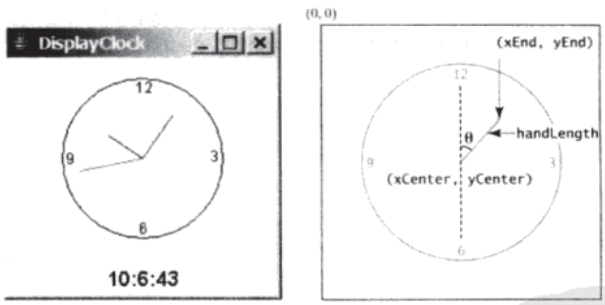


图15-22 a）程序DisplayClock显示一个给出当前时间的时钟；
b）时钟表针的端点由给定的跨度角、表针长度和中心点确定

本节的剩余内容就来解释如何实现StillClock类。因为可以使用这个类，而无须知道它是如何实现的，所以，如果你希望的话是可以跳过它的实现过程的。

为了画一个时钟，需要先画一个圆和三个表示秒、分、时的表针。画表针需要指定一条线段的两个端点。如图15-22b所示，一个端点是时钟的中心（xCenter，yCenter），而另一个端点（xEnd，yEnd）由下面的公式决定：

$$xEnd = xCenter + handLength \times \sin(\theta)$$
$$yEnd = yCenter - handLength \times \cos(\theta)$$

由于一分钟有60秒，所以秒针的角度是：

$$second \times (2\pi/60)$$

分针的位置由分钟数和秒数决定。和秒一起计算的精确分钟值是minute + second/60。例如，如果时间是3分30秒，那么总的分钟数是3.5。由于1小时有60分钟，所以分针的角度是

(minute + second/60) × (2π/60)

由于一圈被分成12小时，所以时针的角度是

(hour + minute/60 + second/(60 × 60)) × (2π/12)

为了简单起见，在计算分针角度和时针角度时可以忽略秒数，因为它们小到可以忽略不计。因此，秒针、分针和时针的端点可以根据下面的公式计算：

```
xSecond = xCenter + secondHandLength × sin(second × (2π/60))
ySecond = yCenter - secondHandLength × cos(second × (2π/60))
xMinute = xCenter + minuteHandLength × sin(minute × (2π/60))
yMinute = yCenter - minuteHandLength × cos(minute × (2π/60))
xHour = xCenter + hourHandLength × sin((hour + minute/60) × (2π/60))
yHour = yCenter - hourHandLength × cos((hour + minute/60) × (2π/60))
```

StillClock类在程序清单15-10中实现。

程序清单15-10 StillClock.java

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.util.*;
4
5  public class StillClock extends JPanel {
6    private int hour;
7    private int minute;
8    private int second;
9
10   /** Construct a default clock with the current time*/
11   public StillClock() {
12     setCurrentTime();
13   }
14
15   /** Construct a clock with specified hour, minute, and second */
16   public StillClock(int hour, int minute, int second) {
17     this.hour = hour;
18     this.minute = minute;
19     this.second = second;
20   }
21
22   /** Return hour */
23   public int getHour() {
24     return hour;
25   }
26
27   /** Set a new hour */
28   public void setHour(int hour) {
29     this.hour = hour;
30     repaint();
31   }
32
33   /** Return minute */
34   public int getMinute() {
35     return minute;
36   }
37
38   /** Set a new minute */
39   public void setMinute(int minute) {
40     this.minute = minute;
41     repaint();
42   }
43
44   /** Return second */
```

```
45   public int getSecond() {
46     return second;
47   }
48
49   /** Set a new second */
50   public void setSecond(int second) {
51     this.second = second;
52     repaint();
53   }
54
55   /** Draw the clock */
56   protected void paintComponent(Graphics g) {
57     super.paintComponent(g);
58
59     // Initialize clock parameters
60     int clockRadius =
61       (int)(Math.min(getWidth(), getHeight()) * 0.8 * 0.5);
62     int xCenter = getWidth() / 2;
63     int yCenter = getHeight() / 2;
64
65     // Draw circle
66     g.setColor(Color.BLACK);
67     g.drawOval(xCenter - clockRadius, yCenter - clockRadius,
68       2 * clockRadius, 2 * clockRadius);
69     g.drawString("12", xCenter - 5, yCenter - clockRadius + 12);
70     g.drawString("9", xCenter - clockRadius + 3, yCenter + 5);
71     g.drawString("3", xCenter + clockRadius - 10, yCenter + 3);
72     g.drawString("6", xCenter - 3, yCenter + clockRadius - 3);
73
74     // Draw second hand
75     int sLength = (int)(clockRadius * 0.8);
76     int xSecond = (int)(xCenter + sLength *
77       Math.sin(second * (2 * Math.PI / 60)));
78     int ySecond = (int)(yCenter - sLength *
79       Math.cos(second * (2 * Math.PI / 60)));
80     g.setColor(Color.red);
81     g.drawLine(xCenter, yCenter, xSecond, ySecond);
82
83     // Draw minute hand
84     int mLength = (int)(clockRadius * 0.65);
85     int xMinute = (int)(xCenter + mLength *
86       Math.sin(minute * (2 * Math.PI / 60)));
87     int yMinute = (int)(yCenter - mLength *
88       Math.cos(minute * (2 * Math.PI / 60)));
89     g.setColor(Color.blue);
90     g.drawLine(xCenter, yCenter, xMinute, yMinute);
91
92     // Draw hour hand
93     int hLength = (int)(clockRadius * 0.5);
94     int xHour = (int)(xCenter + hLength *
95       Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
96     int yHour = (int)(yCenter - hLength *
97       Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
98     g.setColor(Color.green);
99     g.drawLine(xCenter, yCenter, xHour, yHour);
100  }
101
102  public void setCurrentTime() {
103    // Construct a calendar for the current date and time
104    Calendar calendar = new GregorianCalendar();
105
106    // Set current hour, minute and second
107    this.hour = calendar.get(Calendar.HOUR_OF_DAY);
108    this.minute = calendar.get(Calendar.MINUTE);
109    this.second = calendar.get(Calendar.SECOND);
```

```
110   }
111
112   public Dimension getPreferredSize() {
113     return new Dimension(200, 200);
114   }
115 }
```

程序可以使时钟的大小随着框架大小的改变而调整。每次改变框架的大小时，系统都会自动调用paintComponent方法来绘制一个新框架。paintComponent方法按照面板宽度（getWidth()）和高度（getHeight()）的比例显示时钟（Still Clock类中的第60～63行）。

15.11 显示图像

在12.10节中已经学习了如何创建图像图标，以及在标签和按钮上如何显示它们。例如，下面的语句创建一个图像图标，然后在标签上显示它：

```
ImageIcon ImageIcon = new ImageIcon("image/us.gif");
JLabel jlblImage = new JLabel(imageIcon);
```

图像图标显示一个尺寸固定的图像。为了显示大小灵活的图像，需要使用java.awt.Image类。可以使用getImage()方法从一个图像图标中创建一个图像，如下所示：

```
Image image = imageIcon.getImage();
```

使用标签作为显示图像的区域比较简单、方便，但是你对如何显示图像没有多少控制权。更加灵活的显示图像的方式就是在面板上使用Graphics类的drawImage方法。drawImage方法的四个版本如图15-23所示。

java.awt.Graphics	
+drawImage(image: Image, x: int, y: int, bgcolor: Color, observer: ImageObserver): void	在特定位置绘制图像。图像的左上角是在图像环境的坐标空间中的点（x, y）处。图像中的透明像素可以用特定颜色bgcolor来绘制。observer是指在其上显示图像的对象。如果图像大于要绘制它的面积则对它进行截取
+drawImage(image: Image, x: int, y: int, observer: ImageObserver): void	除了不要指定背景色之外，其他都和前一个方法一样
+drawImage(image: Image, x: int, y: int, width: int, height: int, observer: ImageObserver): void	绘制图像的一个可度量版本以将它填满指定矩形中所有的可用空间
+drawImage(image: Image, x: int, y: int, width: int, height: int, bgcolor: Color, observer: ImageObserver): void	它除了提供要绘制的图像后台的一个单色背景色之外，其他都和前一个方法一样

图15-23 可以应用Graphics对象上的drawImage方法以显示GUI组件中的图像

ImageObserver表明在创建一个图像时，会指定一个GUI组件接收图像信息的通知。为了使用像JPanel的Swing组件中的drawImage方法绘制图像，需要覆盖paintComponent方法以告诉组件如何在面板内显示图像。

程序清单15-11给出显示来自image/us.gif的图像的代码。文件image/us.gif（第20行）在类目录下。在第21行获取一个Image对象。drawImage方法显示填充整个面板的图像，如图15-24所示。

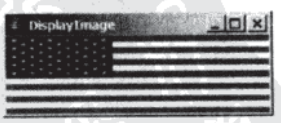


图15-24 在面板内显示一个图像

程序清单15-11 DisplayImage.java

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DisplayImage extends JFrame {
5   public DisplayImage() {
6     add(new ImagePanel());
7   }
8
```

```
 9   public static void main(String[] args) {
10     JFrame frame = new DisplayImage();
11     frame.setTitle("DisplayImage");
12     frame.setSize(300, 300);
13     frame.setLocationRelativeTo(null); // Center the frame
14     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15     frame.setVisible(true);
16   }
17 }
18
19 class ImagePanel extends JPanel {
20   private ImageIcon imageIcon = new ImageIcon("image/us.gif");
21   private Image image = imageIcon.getImage();
22
23   /** Draw image on the panel */
24   protected void paintComponent(Graphics g) {
25     super.paintComponent(g);
26
27     if (image != null)
28       g.drawImage(image, 0, 0, getWidth(), getHeight(), this);
29   }
30 }
```

15.12 实例学习：ImageViewer类

显示图像是Java程序设计的一个常用任务。这个实例学习会开发一个名为ImageViewer的可重用组件，它在面板上显示一个图像。该类包括属性image、stretched、xCoordinate和yCoordinate，它们都有相关的访问器和修改器方法，如图15-25所示。

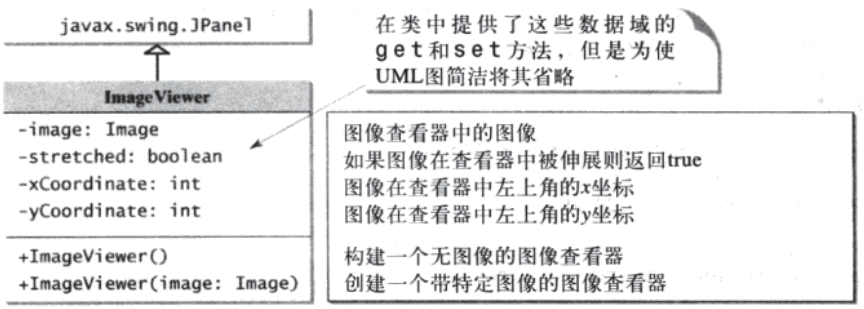


图15-25 ImageViewer类在面板上显示图像

可以使用像JLabel和JButton这样的Swing组件中的图像，但是这些图像是不可伸展的。在ImageViewer中的图像是可伸展的。

我们编写程序清单15-12中的测试程序，使用ImageViewer类显示六个图像。图15-26显示该程序的运行示例。

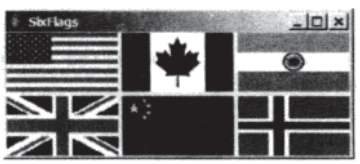


图15-26 在六个ImageViewer组件中显示六个图像

程序清单15-12 SixFlags.java

```
1 import javax.swing.*;
2 import java.awt.*;
3
```

```
 4 public class SixFlags extends JFrame {
 5   public SixFlags() {
 6     Image image1 = new ImageIcon("image/us.gif").getImage();
 7     Image image2 = new ImageIcon("image/ca.gif").getImage();
 8     Image image3 = new ImageIcon("image/india.gif").getImage();
 9     Image image4 = new ImageIcon("image/uk.gif").getImage();
10     Image image5 = new ImageIcon("image/china.gif").getImage();
11     Image image6 = new ImageIcon("image/norway.gif").getImage();
12
13     setLayout(new GridLayout(2, 0, 5, 5));
14     add(new ImageViewer(image1));
15     add(new ImageViewer(image2));
16     add(new ImageViewer(image3));
17     add(new ImageViewer(image4));
18     add(new ImageViewer(image5));
19     add(new ImageViewer(image6));
20   }
21
22   public static void main(String[] args) {
23     SixFlags frame = new SixFlags();
24     frame.setTitle("SixFlags");
25     frame.setSize(400, 320);
26     frame.setLocationRelativeTo(null); // Center the frame
27     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28     frame.setVisible(true);
29   }
30 }
```

程序清单15-13实现ImageViewer类。（注意，可以跳过这个实现）。很容易实现属性image、stretched、xCoordinate和yCoordinate的访问器和修改器方法。paintComponent方法（第26～35行）在面板上显示图像。第29行在显示图像之前确保图像不为null。第30行检查图像是否是可伸展的。

程序清单15-13 ImageViewer.java

```
 1 import java.awt.*;
 2 import javax.swing.*;
 3
 4 public class ImageViewer extends JPanel {
 5   /** Hold value of property image. */
 6   private java.awt.Image image;
 7
 8   /** Hold value of property stretched. */
 9   private boolean stretched = true;
10
11   /** Hold value of property xCoordinate. */
12   private int xCoordinate;
13
14   /** Hold value of property yCoordinate. */
15   private int yCoordinate;
16
17   /** Construct an empty image viewer */
18   public ImageViewer() {
19   }
20
21   /** Construct an image viewer for a specified Image object */
22   public ImageViewer(Image image) {
23     this.image = image;
24   }
25
26   protected void paintComponent(Graphics g) {
27     super.paintComponent(g);
28
29     if (image != null)
30       if (isStretched())
31         g.drawImage(image, xCoordinate, yCoordinate,
32           getWidth(), getHeight(), this);
```

```
33        else
34          g.drawImage(image, xCoordinate, yCoordinate, this);
35    }
36
37    /** Return value of property image */
38    public java.awt.Image getImage() {
39      return image;
40    }
41
42    /** Set a new value for property image */
43    public void setImage(java.awt.Image image) {
44      this.image = image;
45      repaint();
46    }
47
48    /** Return value of property stretched */
49    public boolean isStretched() {
50      return stretched;
51    }
52
53    /** Set a new value for property stretched */
54    public void setStretched(boolean stretched) {
55      this.stretched = stretched;
56      repaint();
57    }
58
59    /** Return value of property xCoordinate */
60    public int getXCoordinate() {
61      return xCoordinate;
62    }
63
64    /** Set a new value for property xCoordinate */
65    public void setXCoordinate(int xCoordinate) {
66      this.xCoordinate = xCoordinate;
67      repaint();
68    }
69
70    /** Return value of property yCoordinate */
71    public int getYCoordinate() {
72      return yCoordinate;
73    }
74
75    /** Set a new value for property yCoordinate */
76    public void setYCoordinate(int yCoordinate) {
77      this.yCoordinate = yCoordinate;
78      repaint();
79    }
80  }
```

本章小结

- 每个组件都有自己的坐标系，原点（0，0）在窗口的左上角，*x*坐标向右增加，*y*坐标向下增加。
- Graphics类是在不同平台的屏幕上显示图形和图像的抽象类。Graphics类在JVM本地平台上实现。使用paintComponent(g)方法在GUI组件上绘图时，这个g是特定平台的抽象Graphics类的具体子类的实例。Graphics类封装了平台细节并且可以在不考虑特定平台的情况下统一绘画。
- 为确保视图域在显示新图之前被清除，必须调用super.paintComponent(g)。用户可以通过调用定义在Component类中的repaint()方法请求再次显示组件。调用repaint()会引起JVM调用paintComponent。用户应该永远都不要直接调用paintComponent。因为这个原因，将paintComponent可见性定义为protected就足够了。

- 通常使用JPanel作为画布。为了在JPanel上绘图，创建新类扩展JPanel并且覆盖paintComponent方法以告诉面板如何绘画。
- 可以设置组件或所画物体的字体，使用字体尺寸测量字体的大小。字体和字体尺寸封装在Font类和FontMetrics类中。FontMetrics可以用来计算字符串的准确长度和宽度，这有助于测量字符串的大小，以便在正确的位置显示它。
- Component类中有setBackground、setForeground和setFont方法。这些方法都用来为整个组件设置颜色和字体。假如想用不同的颜色和字体在同一个面板中显示几条消息，必须使用Graphics类中的setColor和setFont方法设定当前绘画对象的颜色和字体。
- 为了显示一幅图像，首先创建一个图像图标，然后使用ImageIcon的getImage()方法来获取图像的一个Image对象，再使用java.awt.Graphics类中的drawImage方法绘制图像。

复习题

15.2～15.3节

15.1 假设在现有的消息下面绘制一条新消息，x和y的坐标应该增加还是减少？

15.2 为什么Graphics类是抽象的？如何创建一个Graphics对象？

15.3 描述paintComponent方法。它在哪里定义？它是如何调用的？它可以直接调用吗？程序如何调用这个方法？

15.4 为什么paintComponent方法是protected的？如果在子类中将它改为public或private，会发生什么？为什么在程序清单15-1中第21行调用super.paintComponent(g)，在程序清单15-3中第31行调用super.paintComponent(g)？

15.5 可以在任意Swing GUI组件上进行绘画吗？为什么绘画时要将面板作为画布而不是标签或按钮？

15.4～15.7节

15.6 描述绘制字符串、直线、矩形、圆角矩形、3D矩形、椭圆、弧形、多边形和折线段的方法。

15.7 描述填充矩形、圆角矩形、椭圆、弧线和多边形的方法。

15.8 如何在Graphics对象中获取和设置颜色和字体？

15.9 写出绘制下面图形的语句：

- 绘制一条从点（10，10）到（70，30）的粗直线。可以画几条相互紧挨着的线以创造出一条粗线的效果。
- 绘制/填充一个左上角在点（10，10）、宽为100而高为50的矩形。
- 绘制/填充一个宽为100而高为200的圆角矩形。角的水平直径为40，垂直直径为20。
- 绘制/填充一个半径为30的圆。
- 绘制/填充一个宽为50而高为100的椭圆。
- 绘制一个半径为50的上半圆。
- 绘制/填充一个由下面几个点构成的多边形：（20，40）、（30，50）、（40，90）、（90，10）、（10，30）。

15.8～15.10节

15.10 如何找出字体的行距、上升线、下沉线以及高？如何找出Graphics对象中以像素为单位的字符串的准确长度？

15.11 如果在程序清单15-8中的第8行没有初始化message，那么当使用它的无参构造方法创建MessagePanel时会发生什么？

15.12 下面的程序是要在面板上显示一条消息，但是却什么都没有显示。在第2行和第14行都有问题，纠正这些错误。

```
1 public class TestDrawMessage extends javax.swing.JFrame {
2   public void TestDrawMessage() {
3     add(new DrawMessage());
4   }
5
6   public static void main(String[] args) {
7     javax.swing.JFrame frame = new TestDrawMessage();
8     frame.setSize(100, 200);
9     frame.setVisible(true);
10  }
11 }
12
13 class DrawMessage extends javax.swing.JPanel {
14   protected void PaintComponent(java.awt.Graphics g) {
15     super.paintComponent(g);
16     g.drawString("Welcome to Java", 20, 20);
17   }
18 }
```

15.11～15.12节

15.13 如何由ImageIcon对象创建一个Image对象？

15.14 如何由Image对象创建一个ImageIcon对象？

15.15 描述Graphics类中的drawImage方法。

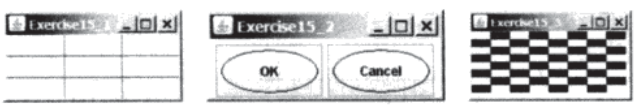
15.16 解释在JLabel和JPanel中显示图像的不同之处。

15.17 哪个包会包括ImageIcon，哪个包会包括Image？

编程练习题

15.2～15.7节

*15.1 （显示一个3×3的网格）编写程序，显示一个3×3的网格，如图15-27a所示。使用红色画垂直线，使用蓝色画水平线。



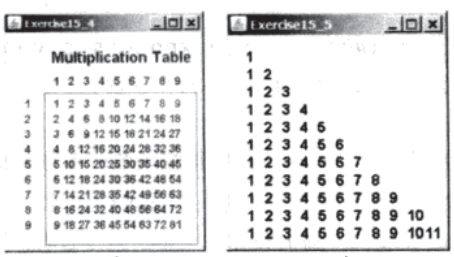
a) b) c)

图15-27 a）练习题15.1显示一个网格；b）练习题15.2显示两个OvalButton对象；c）练习题15.3显示一个棋盘

**15.2 （创建一个自定制的按钮类）扩展JButton类，开发一个名为OvalButton的自定制按钮类，将按钮上的文本显示在椭圆中。图15-27b显示使用OvalButton类创建的两个按钮。

*15.3 （显示一个棋盘）练习题12.10显示一个棋盘，每一个黑格和白格都是一个JButton。改写程序，使用Graphics类中的绘图方法，在JPanel上绘制一个棋盘，如图15-27c所示。

*15.4 （显示一个乘法表）编写程序，使用绘图方法在面板中显示一个乘法表，如图15-28a所示。

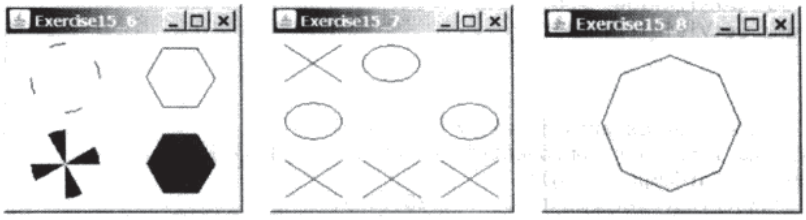


a) b)

图15-28 a）练习题15.4显示一个乘法表；b）练习题15.5将数字显示为一个三角形式

**15.5 （用三角形式显示数字）编写程序，将数字显示成三角形式，如图15-28b所示。改变窗口大小时，为了适应窗口，行数会随窗口的大小而变化。

**15.6 （改进FigurePanel）程序清单15-3中的FigurePanel类可以显示直线、矩形、圆角矩形和椭圆。在类中添加合适的新代码来显示弧形和多边形。使用新的FigurePanel类编写测试程序，显示如图15-29a所示的图形。



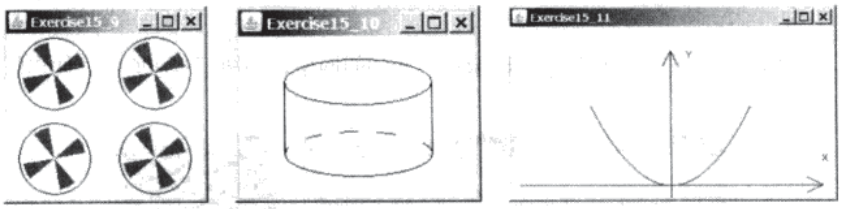
a) b) c)

图15-29 a）在GridLayout框架中显示四种几何图案的面板；b）在井字游戏的棋格随机显示X、O或者空白；c）练习题15.8绘制一个八边形

**15.7 （显示一个井字游戏的棋盘）创建一个自定制面板，它可以显示X、O或者空白。显示什么是重画面板时随机决定的。使用Math.random()方法产生整数0、1或2，对应于面板上显示X、O或者空白。创建一个包含九个自定制面板的框架，如图15-29b所示。

**15.8 （绘制一个八边形）编写一个绘制八边形的程序，如图15-29c所示。

*15.9 （创建四个扇形图）编写程序，使用两行两列的GridLayout框架放置四个扇形图，如图15-30a所示。



a) b) c)

图15-30 a）练习题15.9绘制四个扇形；b）练习题15.10绘制一个圆柱体；c）练习题15.11绘制函数$f(x) = x^2$的图形

*15.10 （创建一个圆柱体）编写程序，绘制一个圆柱体，如图15-30b所示。

**15.11 （绘图表示平方的函数）编写程序，绘制函数$f(x) = x^2$的图形（参见图15-30c）。

提示 使用下面的循环在多边形p中加入点：

```
double scaleFactor = 0.1;

for (int x = -100; x <= 100; x++) {
  p.addPoint(x + 200, 200 - (int)(scaleFactor * x * x));
}
```

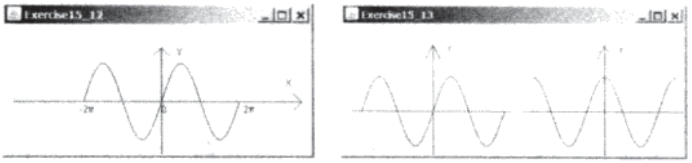
对Graphics对象g使用g.drawPolyline(p.xpoints,p.ypoints,p.npoints)来连接点。p.xpoints返回x坐标的数组，p.ypoints返回y坐标的数组，p.npoints返回Polygon对象p的点的个数。

**15.12 （画出正弦函数的图形）编写一个程序，画出正弦函数的图形，如图15-31a所示。

提示 π的统一码是\u03c0。要显示-2π，使用g.drawString("-2\u03c0",x,y)。对于像sin(x)的三角函数，x是弧度。使用下面的循环将点添加到多边形p中：

```
for (int x = -100; x <= 100; x++) {
  p.addPoint(x + 200,
    100 - (int)(50 * Math.sin((x / 100.0) * 2 * Math.PI)));
}
```

-2π位于点（100，100）处，坐标轴的中心位于点（200，100）处，而2π位于点（300，100）处。使用Graphics类中的drawPolyline方法来连接这些点。



a) b)

图15-31 a）练习题15.12绘制函数$f(x) = \sin(x)$的图形；b）练习题15.13绘制正弦和余弦函数

**15.13 （使用抽象方法绘制函数图形）编写一个抽象类，绘制函数图形。这个类的定义如下：

```
public abstract class AbstractDrawFunction extends JPanel {
  /** Polygon to hold the points */
  private Polygon p = new Polygon();

  protected AbstractDrawFunction () {
    drawFunction();
  }

  /** Return the y-coordinate */
  abstract double f(double x);

  /** Obtain points for x-coordinates 100, 101, ..., 300 */
  public void drawFunction() {
    for (int x = -100; x <= 100; x++) {
      p.addPoint(x + 200, 200 - (int)f(x));
    }
  }

  /** Implement paintComponent to draw axes, labels, and
   *  connecting points
   */
  protected void paintComponent(Graphics g) {
    // To be completed by you
  }
}
```

使用下面的函数测试这个类：

```
f(x) = x²;
f(x) = sin(x);
f(x) = cos(x);
f(x) = tan(x);
f(x) = cos(x) + 5sin(x);
f(x) = 5cos(x) + sin(x);
f(x) = log(x) + x²;
```

对每个函数，扩展AbstractDrawFuntion类来创建一个新类，并实现f方法。图15-31b显示所绘出的正弦函数和余弦函数的图形。

**15.14 （显示一个条形图）编写程序，使用条形图显示作业、平时测验、期中考试和期末考试占总成绩的百分比，如图15-1a所示。假设作业占20%用红色显示，平时测验占10%用蓝色显示，期中考试占30%用绿色显示，期末考试占40%用橙色显示。

**15.15 （显示一个饼图）编写程序，使用饼图显示作业、平时测验、期中考试和期末考试占总成绩的

百分比，如图15-32a所示。假设作业占20%用红色显示，平时测验占10%用蓝色显示，期中考试占30%用绿色显示，期末考试占40%用橙色显示。

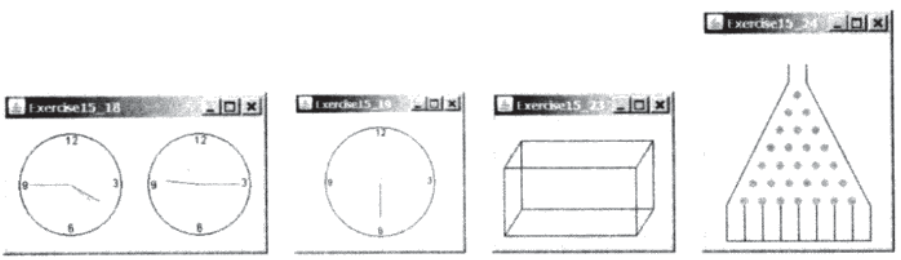
15.16 （获得字体信息）编写程序，在面板上显示消息“Java is fun”。将面板的字体设为TimesRoman、粗体、20像素。将字体的行距、上升线、下沉线、高度和字符串宽度显示为面板的工具提示文本，如图15-32b所示。

a) b) c)

图15-32 a）练习题15.15使用饼图显示作业、平时测验、期中考试和期末考试占总成绩的百分比；b）练习题15.16在工具提示文本中显示字体属性；c）练习题15.17绘制出刽子手游戏的框架

15.17 （游戏：刽子手）编写程序显示流行的刽子手游戏的图像，如图15-32c所示。

15.18 （使用StillClock类）编写程序显示两个时钟。第一个时钟的小时、分钟和秒值为4、20、45，而第二个时钟的小时、分钟和秒值为22、46、15，如图15-33a所示。



a) b) c) d)

图15-33 a）练习题15.18显示两个时钟；b）练习题15.19显示一个任意小时和分钟值的时钟；c）练习题15.23显示一个长方体；d）练习题15.24模拟一个豆机

*15.19 （任意时间）用三个新的Boolean属性hourHandVisible、minuteHandVisible和secondHandVisible以及和它们相关的访问器和修改器方法修改StillClock类。可以使用set方法指定指针为可见的或者不可见的。编写测试程序只显示小时指针和分钟指针。小时和分钟的值是随机产生的。小时值在0到11之间，而分钟值是0或者30，如图15-33b所示。

**15.20 （画一个精细的时钟）修改15.12节中的StillClock类，绘制一个将小时和分钟分得更精细的时钟，如图15-1b所示。

**15.21 （显示一个带图像的井字游戏棋盘）改写练习题12.7，在JPanel上显示图像而不是在JLabel上显示图像图标。

**15.22 （显示STOP信号）编写程序显示STOP信号，如图15-1c所示。八边形是红色的而信号是白色的。

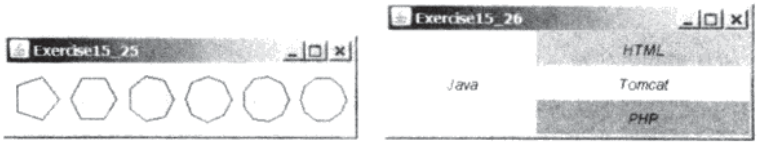
提示 参见程序清单15-5以及程序清单15-6。

15.23 （显示一个长方体）编写一个程序，显示一个长方体，如图15-33c所示。这个立方体可以随着框架的增加和减小而扩张或者收缩。

**15.24 （游戏：豆机）编写一个程序，显示练习题6.21中介绍过的豆机。豆机应该放在可改变大小的面

板的中央，如图15-33d所示。

**15.25 （几何方面：显示一个正n边形）创建一个名为RegularPolygonPanel的JPanel的子类，来绘制一个n条边的正多边形。这个类包括了名为numberOfSides的属性，它表示多边形边的个数。多边形放在面板的中心位置。多边形大小和面板的大小成比例。从RegularPolygonPanel创建一个五边形、六边形、七边形、八边形、九边形和十边形，然后在框架中显示它们，如图15-34a所示。



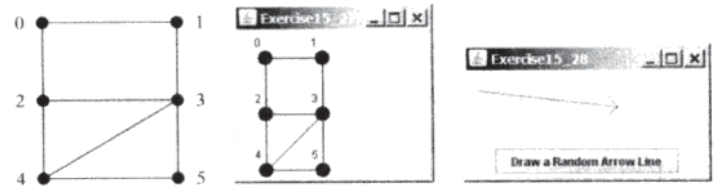
a) b)

图15-34 a）练习题15.25显示几个n边形；b）练习题15.26使用MessagePanel显示4个字符串

15.26 （使用MessagePanel类）编写程序显示四条消息，如图15-35b所示。

**15.27 （显示一个图形）一个图形包括一些端点以及连接这些点的线。编写程序从文件中读取一个图形，然后在面板上显示它。文件的第一行包括表明端点个数的数字（n）。这些点都被标上0，1，...，n-1。每一个连接线的形式都是u x y v1，v2，...，这种形式描述点u的位置在（x，y）处并且有和它相连的两条线（u，v1）、（u，v2）等。图15-35a给出一个图形的文件的例子，然后在面板上显示这个图形，如图15-35b所示。

```
File
6
0 30 30 1 2
1 90 30 0 3
2 30 90 0 3 4
3 90 90 1 2 4 5
4 30 150 2 3 5
5 90 150 3 4
```



a) b) c)

图15-35 a）~b）程序读取关于图形的信息，然后显示它；c）程序显示一条带箭头的直线

**15.28 （绘制一条带箭头的直线）编写一个静态方法，使用下面的方法头绘制一条从起点到终点的带箭头的直线：

```
public static void drawArrowLine(int x1, int y1,
  int x2, int y2, Graphics g)
```

编写一个测试程序，当点击Draw Random Arrow Line按钮时，程序会随机绘制一条带箭头的直线，如图15-35c所示。