第7章　图形程序设计

- ▲ Swing概述
- ▲ 创建框架
- ▲ 框架定位
- ▲ 在组件中显示信息
- ▲ 2D图形
- ▲ 颜色
- ▲ 字体
- ▲ 图像

到目前为止，我们编写的程序都是通过键盘接收输入，在控制台屏幕上显示结果。绝大多数用户并不喜欢这种交互方式。现代的程序早已不采用这种操作方法了，网络程序更是如此。从本章开始，将介绍如何编写使用图形用户界面（GUI）的Java程序。本章主要讲述如何编写定义屏幕上的窗口大小和位置的程序；如何在窗口中采用多种字体显示文本；如何显示图像等。这些都是需要掌握的编程技能，在后续各章中，将会使用这些技术编写一些很有趣味的程序。

随后的两章，将介绍如何处理敲击键盘，点击鼠标这样的事件，以及如何在应用程序中添加菜单和按钮这样的界面元素。学习完这三章之后，读者就应该能够掌握编写独立运行的图形应用程序的要素了。有关更加复杂的图形程序设计技巧请参看卷II。

另外，如果只希望用Java编写服务器端的应用程序，并且对GUI 编程又不感兴趣，那么就可以跳过这几个章节。

7.1　Swing概述

在Java 1.0刚刚出现的时候，包含了一个用于基本GUI程序设计的类库，Sun将它称为抽象窗口工具箱（Abstract Window Toolkit，AWT）。基本AWT库采用将处理用户界面元素的任务委派给每个目标平台（Windows、Solaris、Macintosh等）的本地GUI工具箱的方式，由本地GUI工具箱负责用户界面元素的创建和动作。例如，如果使用最初的AWT在Java窗口中放置一个文本框，就会有一个低层的“对等体”文本框，用来真正地处理文本输入。从理论上说，结果程序可以运行在任何平台上，但观感（look and feel）的效果却依赖于目标平台，因此，Sun公司的口号是“一次编写，随处使用”。

对于简单的应用程序来说，基于对等体方法的效果还是不错的。但是，要想编写依赖于本地用户界面元素的高质量、可移植的图形库就会暴露出一些缺陷。例如，菜单、滚动条和文本域这些用户界面元素，在不同的平台上，操作行为存在着一些微妙的差别。因此，要想给予用户一致的、可预见性的界面操作方式是相当困难的。而且，有些图形环境（如X11/Motif）并没有像Windows或Macintosh这样丰富的用户界面组件集合。这也就将基于对等体的可移植库限制在了“最小公分母”的范围内。其结果使AWT构建的GUI应用程序看起来没有Windows或Macintosh应用程序显示的那么漂亮，也没有提供那些平台用户所认知的功能。更加糟糕的是，

在不同平台上的AWT用户界面库中存在着不同的bug。研发人员必须在每一个平台上测试应用程序，因此人们嘲弄地将AWT称为“一次编写，随处调试”。

在1996年，Netscape创建了一种称为IFC（Internet Foundation Class）的GUI库，它采用了与AWT完全不同的工作方式。它将按钮、菜单这样的用户界面元素绘制在空白窗口上，而对等体只需要创建和绘制窗口。因此，Netscape的IFC组件在程序运行的所有平台上的外观和动作都一样。Sun与Netscape合作完善了这种方式，创建了一个名为Swing的用户界面库。Swing可作为Java 1.1的扩展部分使用，现已成为Java SE 1.2标准库的一部分。

就像Duke Ellington所说的那样：“如果没有Swing，Java图形界面就没有任何意义”。现在，Swing是不对等基于GUI工具箱的正式名字。它已是Java基础类库（Java Foundation Class，JFC）的一部分。完整的JFC十分庞大，其中包含的内容远远大于Swing GUI工具箱。JFC特性不仅仅包含了Swing组件，而且还包含了一个可访问的API、一个2D API和一个可拖拽的API。

注释：Swing没有完全替代AWT，而是基于AWT架构之上。Swing仅仅提供了能力更加强大的用户界面组件。尤其在采用Swing编写的程序中，还需要使用基本的AWT处理事件。从现在开始，Swing是指“被绘制的”用户界面类；AWT是指像事件处理这样的窗口工具箱的低层机制。

当然，在用户屏幕上显示基于Swing用户界面的元素要比显示AWT的基于对等体组件的速度慢一些。鉴于以往的经验，对于任何一台现代的计算机来说，微小的速度差别无妨大碍。另外，由于下列几点无法抗拒的原因，人们选择Swing：

- Swing拥有一个丰富、便捷的用户界面元素集合。
- Swing对低层平台依赖的很少，因此与平台相关的bug很少。
- Swing给予不同平台的用户一致的感觉。

不过，上面第三点存在着一个潜在的问题：如果在所有平台上用户界面元素看起来都一样，那么它们就有可能与本地控件不一样，而这些平台的用户对此可能并不熟悉。

Swing采用了一种很巧妙的方式来解决这个问题。在程序员编写Swing程序时，可以为程序指定专门的“观感”。例如，图7-1和图7-2展示了同一个程序在Windows和Motif平台下运行的观感。

此外，Sun开发了一种称为Metal的独立于平台的观感。现在，市场上人们将它称为“Java观感”。不过，绝大多数程序员还继续沿用Metal这个术语，在本书中也将这样称呼。

有些人批评Metal有点笨重，而在Java SE 5.0中看起来却焕然一新（参见图7-3）。现在，Metal外观支持多种主题，每一种主题的颜色和字体都有微小的变化。默认的主题叫做Ocean。

在Java SE 6中，Sun改进了对Windows和GTK本地观感的支持。Swing应用程序将会支持色彩主题的个性化设置，逼真地表现着动态按钮和变得十分时尚的滚动条。

有些用户更希望Java应用程序采用本地平台的观感，而不采用Metal或第三方的观感。在第8章中你将会看到，让用户随意地选择喜欢的观感是非常容易的。

注释：虽然在这本书中没有用很多篇幅介绍有关设定“观感”的方式，但Java程序员能够扩展一个已存在的观感，甚至设计一个全新的观感。设计Swing组件的绘制方式是一个很

繁琐的过程。有些程序员已经尝试过一些这样的工作，尤其是将Java移植到信息亭终端和手持设备这样的非传统平台上。有关一些有趣的观感的实现请参看http://www.javootoo.com。Java SE 5.0引入了一种被称为Synth的新观感方式，使用它处理比较容易。在Synth中，可以通过提供图像文件和XML描述符来定义一种新的观感，而不需要编写任何代码。

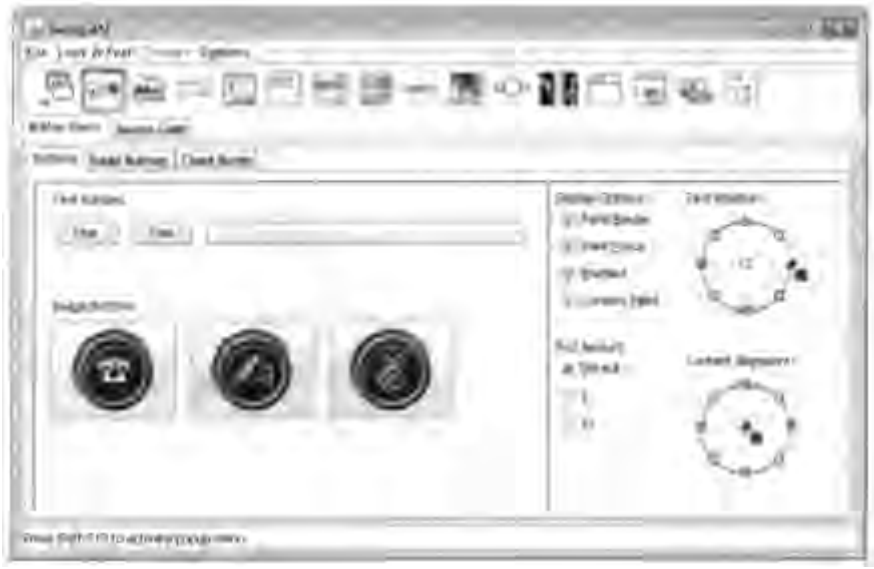

图7-1 Swing的Window观感

图7-2 Swing的GTK观感

✔ 注释：绝大多数Java用户界面程序设计都采用Swing，但有一个特别的例外。Eclipse集成开发环境使用了一种与AWT类似，且被称为SWT的图形工具箱，它可以映射到不同平台的本地组件上。有关SWT的描述可以在网站http://www.eclipse.org/ articles/找到。

最后，给大家一个忠告，如果使用过Visual Baisc或C#编写Microsoft Windows应用程序，就应该了解这些产品提供的图形布局工具和资源编辑器带来的便利。这些工具可以用来设计应

用程序的外观，然后生成大部分（有时是全部）GUI代码。尽管也有一些Java程序设计的GUI构造器，但要想有效地使用这些工具，需要知道如何手工地创建用户界面。本章剩余的部分将介绍关于显示一个窗口及绘制内容的基本知识。

图7-3 Metal观感的Ocean主题

7.2 创建框架

在Java中，顶层窗口（就是没有包含在其他窗口中的窗口）被称为框架（frame）。在AWT库中有一个称为Frame的类，用于描述顶层窗口。这个类的Swing版本名为JFrame，它扩展于Frame类。JFrame是极少数几个不绘制在画布上的Swing组件之一。因此，它的修饰部件（按钮、标题栏、图标等）由用户的窗口系统绘制，而不是由Swing绘制。

警告：绝大多数Swing组件类都以“J”开头，例如，JButton、JFrame等。在Java中有Button和Frame这样的类，但它们属于AWT组件。如果偶然地忘记书写“J”，程序仍然可以进行编译和运行，但是将Swing和AWT组件混合在一起使用将会导致视觉和行为的不一致。

在本节中，将介绍有关Swing的JFrame的常用方法。例7-1给出了一个在屏幕中显示一个空框架的简单程序，如图7-4所示。

例7-1 SimpleFrameTest.java

```
1. import javax.swing.*;
2.
3. /**
4.  * @version 1.32 2007-06-12
5.  * @author Cay Horstmann
6.  */
7. public class SimpleFrameTest
8. {
9.    public static void main(String[] args)
10.   {
11.      SimpleFrame frame = new SimpleFrame();
12.      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13.      frame.setVisible(true);
```

```
14.    }
15. }
16.
17. class SimpleFrame extends JFrame
18. {
19.    public SimpleFrame()
20.    {
21.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
22.    }
23.
24.    public static final int DEFAULT_WIDTH = 300;
25.    public static final int DEFAULT_HEIGHT = 200;
26. }
```

下面逐行地讲解一下这个程序。

图7-4 最简单的可见框架

Swing类位于javax.swing包中。包名javax表示这是一个Java扩展包，而不是核心包。Swing类实际上是对Java 1.1的扩展。从1.2版本开始，在每个Java SE实现中都包含它。

在默认情况下，框架的大小为0×0，这种框架没有什么实际意义。这里定义了一个子类SimpleFrame，它的构造器将框架大小设置为300×200像素。这是SimpleFrame和JFrame之间惟一的差别。

在SimpleFrameTest类的main方法中，由创建一个SmpleFrame对象开始程序的运行。

在每个Swing程序中，有两点技术需要强调。

首先，所有的Swing组件必须由事件调度线程（event dispatch thread）进行配置，线程将鼠标点击和键盘敲击控制转移到用户接口组件。下面的代码片断是事件调度线程中的执行代码：

```
EventQueue.invokeLater(new Runnable()
   {
      public void run()
      {
         statements
      }
   });
```

这一内容将在14章中详细讨论。现在，只需要简单地将其看作启动一个Swing程序的神奇代码。

注释：许多Swing程序并没有在事件调度线程中初始化用户接口。在主线程中完成初始化是通常采用的方式。遗憾的是，由于Swing组件十分复杂，Sun的程序员无法保证这种方式的安全性。虽然发生错误的概率非常小，但任何人不愿意成为遭遇这个问题的不幸者之一。即使代码看起来有些神秘，也最好能够保证其正确性。

接下来，定义一个用户关闭这个框架时的响应动作。对于这个程序而言，只让程序简单地退出即可。选择这个响应动作的语句是

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

在包含多个框架的程序中，不能在用户关闭其中的一个框架时就让程序退出。在默认情况

下，用户关闭窗口时只是将框架隐藏起来，而程序并没有终止（在最后一个框架不可见之后，程序再终止，这样处理比较合适，而Swing却不是这样工作的）。

简单地构造框架是不会自动地显示出来的，框架起初是不可见的。这就给程序员了一个机会，可以在框架第一次显示之前往其中添加组件。为了显示框架，main方法需要调用框架的setVisible方法。

注释：在Java SE 5.0以前的版本中，可以使用JFrame类从超类Window继承show方法。Window类的超类是Component，其中也有一个show方法。在Java SE 1.2中不提倡使用Component.show。如果想要显示一个组件，建议调用setVisible(true)。然而，Java SE 1.4以前的版本，并没有反对使用Window.show方法。事实上，这个方法很实用，它可以让窗口可见，且置于其他窗口的前面。遗憾的是，由于不提倡使用它，随之也失去了这一好处，Java SE 5.0也不赞成使用show显示窗口。

在初始化语句结束后，main方法退出。需要注意，退出main并没有终止程序，终止的只是主线程。事件调度线程保持程序处于激活状态，直到关闭框架或调用System.exit方法终止程序。

图7-4中显示的是运行例7-1的结果，它只是一个很枯燥的顶层窗口。在这个图中看到的标题栏和外框装饰（比如，重置窗口大小的拐角）都是由操作系统绘制的，而不是Swing库。在Windows，GTK或Mac下运行同样的程序，将会得到不同的框架装饰。Swing库负责绘制框架内的所有内容。在这个程序中，只用默认的背景色填充了框架。

注释：在Java SE 1.4中，可以调用frame.setUndecorated(true) 关闭所有框架装饰。

7.3 框架定位

JFrame类本身只包含若干个改变框架外观的方法。然而，通过继承从JFrame的各个超类中继承了许多用于处理框架大小和位置的方法。其中最重要的有下面几个：

- setLocation和setBounds方法用于设置框架的位置。
- setIconImage用于告诉窗口系统在标题栏、任务切换窗口等位置显示哪个图标。
- setTitle用于改变标题栏的文字。
- setResizable利用一个boolean值确定框架的大小是否允许用户改变。

图7-5给出了JFrame类的继承层次。

提示：本节API注解给出了一些非常重要的用于设置框架适当观感的方法。其中一些定义在JFrame类中，而另一些来自于JFrame的各个超类。因此，可能需要查阅API文档，以便确定是否存在能够实现某个特定目的的方法。遗憾的是，在文档中查阅一个类继承的方法是一件比较令人烦恼的事情。对于子类来说，API文档只解释了覆盖的方法。例如，可以应用于JFrame类对象的toFront方法，由于它是从Window类继承而来的，所以JFrame文档没有对它进行解释。如果认为应该有一个能够完成某项操作的方法，而在处理的类文档中又没有解释，就应该查看这个类的超类API文档。每页API上面都有一个对超类的超链接，继承方法被列在新方法和覆盖方法的汇总下面。

正像API注解中所说的，对Component类（是所有GUI对象的祖先）和Window类（是Frame类的超类）需要仔细地研究一下，从中找到缩放和改变框架的方法。例如，在Component类中的setLocation方法是重定位组件的一个方法。如果调用

```
setLocation(x, y)
```

则窗口将放置在左上角水平x像素，垂直y像素的位置，坐标（0，0）位于屏幕的左上角。同样地，Component中的setBounds方法可以实现一步重定位组件（特别是JFrame）大小和位置的操作，例如：

```
setBounds(x, y, width, height)
```

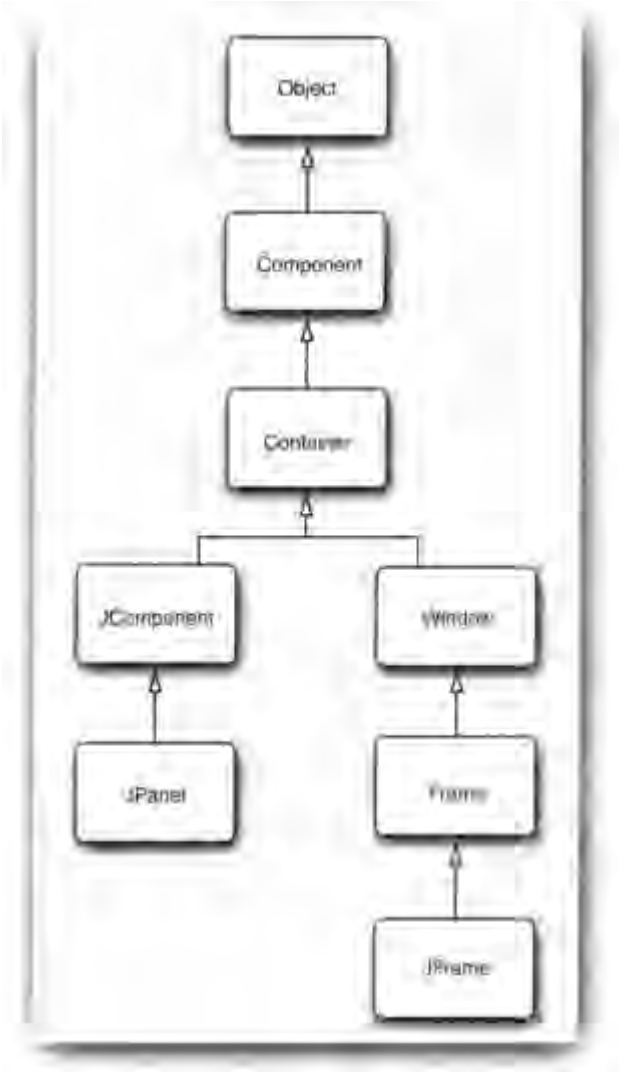

图7-5 AWT和Swing中框架和组件类的继承层次

可以让窗口系统控制窗口的位置，如果在显示窗口之前调用

```
setLoationByPlatform(true);
```

窗口系统会拾取窗口的位置（而不是大小），通常是距最后一个显示窗口很少偏移量的位置。

注释：对于框架来说，setLocation和setBounds中的坐标均相对于整个屏幕。在第9章中将会看到，在容器中包含的组件所指的坐标均相对于容器。

7.4 框架属性

组件类的很多方法是以获取/设置这一对操作形式出现的，例如，Frame类的下列方法：

```
public String getTitle()
public void setTitle(String title)
```

这样的一个获取/设置对被称为一种属性。属性包含属性名和类型。将get或set之后的第一个字母改为小写字母就可以得到相应的属性名。例如，Frame类有一个名为title且类型为String的属性。

从概念上讲，title是框架的一个属性。当设置这个属性时，希望这个标题能够改变用户屏幕上的显示。当获取这个属性时，希望能够返回已经设置的属性值。

我们并不清楚（也不关心）Frame类是如何实现这个属性的。或许只是简单的利用对等框架存储标题。或许有一个实例域：

```
private String title; // not required for property
```

如果类没有匹配的实例域，我们将不清楚（也不关心）如何实现获取和设置方法。或许只是读、写实例域，或许还执行了很多其他的操作。例如，当标题发生变化时，通知给窗口系统。

针对get/set约定有一个例外：对于类型为boolean的属性，获取类方法由is开头。例如，下面两个方法定义了locationByPlatform属性：

```
public boolean isLocationByPlatform()
public void setLocationByPlatform(boolean b)
```

有关属性的详细内容，请参看卷II第8章。

注释：许多程序设计语言（特别是，Visual Basic和C#）已经内置了对属性的支持。在Java未来的版本中，也有可能提供对属性的支持。

7.5 决定框架大小

请注意：如果没有明确地指定框架的大小，所有框架的默认值为0×0像素。为了让示例程序尽可能地简单，这里将框架的大小重置为大多数情况下都可以接受的显示尺寸。然而，对于专业应用程序来说，应该检查屏幕的分辨率，并根据其分辨率编写代码重置框架的大小，如在膝上型电脑的屏幕上，正常显示的窗口在高分辨率屏幕上可能会变成一张邮票的大小。

为了得到屏幕的大小，需要按照下列步骤操作。调用Toolkit类的静态方法getDefaultToolkit得到一个Toolkit对象（Toolkit类包含很多与本地窗口系统打交道的方法）。然后，调用getScreenSize方法，这个方法以Dimension对象的形式返回屏幕的大小。Dimension对象同时用公有实例变量width和height保存着屏幕的宽度和高度。下面是相关的代码：

```
Toolkit kit = Toolkit.getDefaultToolkit();
Dimension screenSize = kit.getScreenSize();
int screenWidth = screenSize.width;
int screenHeight = screenSize.height;
```

下面，将框架大小设定为上面取值的50%，然后，告知窗口系统定位框架：

```
setSize(screenWidth / 2, screenHeight / 2);
setLocationByPlatform(true);
```

另外，还提供一个图标。由于图像的描述与系统有关，所以需要再次使用工具箱加载图像。然后，将这个图像设置为框架的图标。

```
Image img = kit.getImage("icon.gif");
setIconImage(img);
```

对于不同的操作系统，所看到的图标显示位置有可能不同。例如，在Windows中，图标显示在窗口的左上角，按下ALT+TAB，可以在活动任务的列表中看到相应程序的图标。

例7-2是完整的程序。当运行程序时，请注意看“ Core Java ”图标。

下面是为了处理框架给予的一些提示：

- 如果框架中只包含标准的组件，如按钮和文本框，那么可以通过调用pack方法设置框架大小。框架将被设置为刚好能够放置所有组件的大小。在通常情况下，将程序的主框架尺寸设置为最大。正如Java SE 1.4，可以通过调用下列方法将框架设置为最大。

```
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
```

- 牢记用户定位应用程序的框架位置、重置框架大小，并且在应用程序再次启动时恢复这些内容是一个不错的想法。在第10章中将会介绍如何运用API的参数选择达到这个目的。
- 如果编写一个使用多个显示屏幕的应用程序，应该利用GraphicsEnvironment和GraphicsDevice类获得显示屏幕的大小。
- GraphicsDevice类允许在全屏模式下执行应用程序。

例7-2 SizedFrameTest.java

```
1. import java.awt.*;
2.
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.32 2007-04-14
7.  * @author Cay Horstmann
8.  */
9. public class SizedFrameTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                SizedFrame frame = new SizedFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. class SizedFrame extends JFrame
```

```
{
   public SizedFrame()
   {
      // get screen dimensions

      Toolkit kit = Toolkit.getDefaultToolkit();
      Dimension screenSize = kit.getScreenSize();
      int screenHeight = screenSize.height;
      int screenWidth = screenSize.width;

      // set frame width, height and let platform pick screen location

      setSize(screenWidth / 2, screenHeight / 2);
      setLocationByPlatform(true);

      // set frame icon and title

      Image img = kit.getImage("icon.gif");
      setIconImage(img);
      setTitle("SizedFrame");
   }
}
```

API java.awt.Component 1.0

- `boolean isVisible()`
- `void setVisible(boolean b)`

  获取或设置visible属性。组件最初是可见的，但JFrame这样的顶层组件例外。
- `void setSize(int width, int height) 1.1`

  使用给定的宽度和高度，重新设置组件的大小。
- `void setLocation(int x,int y)  1.1`

  将组件移到一个新的位置上。如果这个组件不是顶层组件，x和y坐标（或者p.x和p.y）是容器坐标；否则是屏幕坐标（例如：aJFrame）。
- `void setBounds(int x, int y, int width, int height)  1.1`

  移动并重新设置组件的大小。
- `Dimension getSize() 1.1`
- `void setSize(Dimension d) 1.1`

  获取或设置当前组件的size属性。

API java.awt.Window 1.0

- `void toFront()`

  将这个窗口显示在其他窗口前面。
- `void toBack()`

  将这个窗口移到桌面窗口栈的后面，并重新排列所有的可见窗口。
- `boolean isLocationByPlatform() 5.0`

- `void setLocationByPlatform(boolean b) 5.0`
  获取或设置locationByPlatform属性。这个属性在窗口显示之前被设置，由平台选择一个合适的位置。

API `java.awt.Frame 1.0`

- `boolean isResizable()`
- `void setResizable(boolean b)`
  获取或设置resizable属性。这个属性设置后，用户可以重新设置框架的大小。
- `String getTitle()`
- `void setTitle(String s)`
  获取或设置title属性，这个属性确定框架标题栏中的文字。
- `Image getIconImage()`
- `void setIconImage(Image image)`
  获取或设置iconImage属性，这个属性确定框架的图标。窗口系统可能会将图标作为框架装饰或其他部位的一部分显示。
- `boolean isUndecorated() 1.4`
- `void setUndecorated(boolean b) 1.4`
  获取或设置undecorated属性。这个属性设置后，框架显示中将没有标题栏或关闭按钮这样的装饰。在框架显示之前，必须调用这个方法。
- `int getExtendedState() 1.4`
- `void setExtendedState(int state) 1.4`
  获取或设置窗口状态。状态是下列值之一

```
Frame.NORMAL
Frame.ICONIFIED
Frame.MAXIMIZED_HORIZ
Frame.MAXIMIZED_VERT
Frame.MAXIMIZED_BOTH
```

API `java.awt.Toolkit 1.0`

- `static Toolkit getDefaultToolkit()`
  返回默认的工具箱。
- `Dimension getScreenSize()`
  返回用户屏幕的尺寸。
- `Image getImage(String filename)`
  加载文件名为filename的图像。

7.6 在组件中显示信息

本节将论述如何在框架内显示信息。例如，我们不再像第3章那样，采用文本方式将“ Not a

Hello, World program ”显示在控制台窗口中，而是在框架中显示这个消息，如图7-6所示。

图7-6 一个显示消息的框架

可以将消息字符串直接绘制在框架中，但这并不是一种好的编程习惯。在Java中，框架被设计为放置组件的容器，可以将菜单栏和其他的用户界面元素放置在其中。在通常情况下，应该在另一组件上绘制信息，并将这个组件添加到框架中。

JFrame的结构相当复杂。在图7-7中给出了JFrame的结构。可以看到，在JFrame中有四层面板。其中的根面板、层级面板和玻璃面板人们并不太关心；它们是用来组织菜单栏和内容窗格以及实现观感的。Swing程序员最关心的是内容窗格（content pane）。在设计框架的时候，要使用下列代码将所有的组件添加到内容窗格中：

```
Container contentPane = frame.getContentPane();
Component c = . . .;
contentPane.add(c);
```

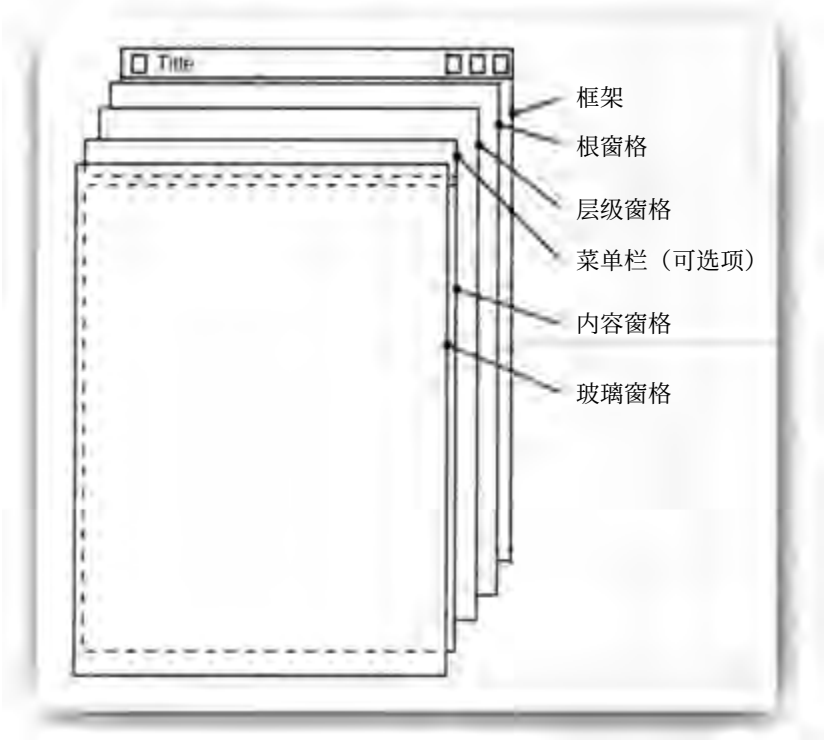

图7-7 JFrame的内部结构

在Java SE 1.4及以前的版本中，JFrame类中的add方法抛出了一个异常信息“ Do not use JFrame.add().Use JFrame.getContentPane().add instead ”。在Java SE 5.0中，JFrame.add方法不再显示这些提示信息，只是简单地调用内容窗格中的add。

因此，在Java SE 5.0中，可以直接调用

```
frame.add(c);
```

在这里，打算将一个绘制消息的组件添加到框架中。绘制一个组件，需要定义一个扩展JComponent的类，并覆盖其中的paintComponent方法。

paintComponent方法有一个Graphics类型的参数，这个参数保存着用于绘制图像和文本的设置，例如，设置的字体或当前的颜色。在Java中，所有的绘制都必须使用Graphics对象，其中包含了绘制图案、图像和文本的方法。

注释：Graphics参数与Windows中的设备环境或X11程序设计中的图形环境基本类似。

下列代码给出了如何创建一个能够进行绘制的组件：

```
class MyComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      code for drawing
   }
}
```

无论何种原因，只要窗口需要重新绘图，事件处理器就会通告组件，从而引发执行所有组件的painComponent方法。

一定不要自己调用paintComponent方法。在应用程序需要重新绘图的时候，这个方法将被自动地调用，不要人为地干预这个自动的处理过程。

何种类别的动作会触发这个自动响应过程呢？例如，在用户扩大窗口或极小化窗口，然后又恢复窗口的大小时会引发重新绘图。如果用户弹出了另外一个窗口，并且这个窗口覆盖了一个已经存在的窗口，使得覆盖的窗口不可见，则此时被覆盖的应用程序窗口被破坏，需要重新绘制（图形系统不保存下面的像素）。当然，窗口第一次显示时，需要处理一些代码，主要包含确定绘制最初元素的方式以及位置。

提示：如果需要强制刷新屏幕，就需要调用repaint方法，而不是paintComponent方法。它将引发采用相应配置的Graphics对象调用所有组件的paintComponent方法。

从上述代码片段中可以看到，paintComponent方法只有一个Graphics类型的参数。对于屏幕显示来说，Graphics对象的度量单位是像素。坐标（0，0）指出所绘制组件表面的左上角。

显示文本是一种特殊的绘图。在Graphics类中有一个drawString方法，调用的语法格式为：

```
g.drawString(text, x, y)
```

在这里，打算在原始窗口大约水平1/4，垂直1/2的位置显示字符串“Not a Hello, World program”。现在，尽管不知道应该如何度量这个字符串的大小，但可以将字符串的开始位置定义在坐标（75，100）。这就意味着字符串中的第一个字符位于从左向右75个像素，从上向下100个像素的位置（实际上，文本的基线位于像素100的位置，有关文本的度量方式将在稍后阐述）。因此，paintComponent方法的书写内容如下所示：

```
class NotHelloWorldComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
```

```
   }

   public static final int MESSAGE_X = 75;
   public static final int MESSAGE_Y = 100;
}
```

注释：有些程序员更喜欢扩展JPanel，而不是JComponent。JPanel是一个可以包含其他组件的容器（container），但同样也可以在其上面进行绘制。有一点不同之处是，面板不透明，这意味着需要在面板的边界内绘制所有的像素。最容易实现的方法是，在每个面板子类的paintComponent方法中调用super.paintComponent来用背景色绘制面板：

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

      . . . // code for drawing will go here
   }
}
```

例7-3 NotHelloWorld.java

```
1. import javax.swing.*;
2. import java.awt.*;
3.
4. /**
5.  * @version 1.32 2007-06-12
6.  * @author Cay Horstmann
7.  */
8. public class NotHelloWorld
9. {
10.    public static void main(String[] args)
11.    {
12.       EventQueue.invokeLater(new Runnable()
13.          {
14.             public void run()
15.             {
16.                NotHelloWorldFrame frame = new NotHelloWorldFrame();
17.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18.                frame.setVisible(true);
19.             }
20.          });
21.    }
22. }
23.
24. /**
25.  * A frame that contains a message panel
26.  */
27. class NotHelloWorldFrame extends JFrame
28. {
29.    public NotHelloWorldFrame()
30.    {
31.       setTitle("NotHelloWorld");
32.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

```
33.
34.       // add panel to frame
35.
36.       NotHelloWorldPanel panel = new NotHelloWorldPanel();
37.       add(panel);
38.    }
39.
40.    public static final int DEFAULT_WIDTH = 300;
41.    public static final int DEFAULT_HEIGHT = 200;
42. }
43.
44. /**
45.  * A panel that displays a message.
46.  */
47. class NotHelloWorldPanel extends JPanel
48. {
49.    public void paintComponent(Graphics g)
50.    {
51.       g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
52.    }
53.
54.    public static final int MESSAGE_X = 75;
55.    public static final int MESSAGE_Y = 100;
56. }
57.
```

API `javax.swing.JFrame` 1.2

- `Container getContentPane()`
  返回这个JFrame的内容窗格对象。
- `Component add(Component c)`
  将一个给定的组件添加到该框架的内容窗格中（在Java SE 5.0以前的版本中，这个方法将抛出一个异常）。

API `java.awt.Component` 1.0

- `void repaint()`
  “尽可能快地”重新绘制组件。
- `public void repaint(int x, int y, int width, int height)`
  “尽可能快地”重新绘制组件的一部分。

API `javax.swing.JComponent` 1.2

- `void paintComponent(Grphics g)`
  覆盖这个方法来描述应该如何绘制自己的组件。

7.7 2D图形

自从Java版本1.0以来，Graphics类就包含绘制直线、矩形和椭圆等方法。但是，这些绘制

图形的操作能力非常有限。例如，不能改变线的粗细，不能旋转这些图形。

Java SE 1.2引入了Java 2D库，这个库实现了一组功能强大的图形操作。在本章中，只介绍Java 2D库的基础部分，有关高级功能的详细内容请参看卷II的高级AWT章节。

要想使用Java 2D库绘制图形，需要获得一个Graphics2D类对象。这个类是Graphics类的子类。自从Java SE 2版本以来，paintComponent方法就会自动地获得一个Graphics2D类对象，我们只需要进行一次类型转换就可以了。如下所示：

```
public void paintComponent(Graphics g)
{
   Graphics2D g2 = (Graphics2D) g;
   . . .
}
```

Java 2D库采用面向对象的方式将几何图形组织起来。包含描述直线、矩形的椭圆的类：

```
Line2D
Rectangle2D
Ellipse2D
```

这些类全部实现了Shape接口。

注释：Java 2D库支持更加复杂的图形，例如圆弧、二次曲线、三次曲线和通用路径。有关更详细的内容请参看卷II第7章。

要想绘制图形，首先要创建一个实现了Shape接口的类的对象，然后调用Graphics2D类中的draw方法。例如，

```
Rectangle2D rect = . . .;
g2.draw(rect);
```

注释：在Java 2D库出现之前，程序员使用Grpahics类中的drawRectangle方法绘制图形。从表面上看，老式风格的方法调用起来好像更加简单一点。然而，使用Java 2D库，可以选择Java 2D库中提供的一些工具提高绘制能力。

使用Java 2D图形类或许会增加一些复杂度。在1.0的绘制方法中，采用的是整型像素坐标，而Java 2D图形采用的是浮点坐标。在很多情况下，用户可以使用更有意义的形式（例如，微米或英寸）指定图形的坐标，然后再将其转换成像素，这样做很方便。在Java 2D库中，内部的很多浮点计算都采用单精度float。毕竟，几何计算的最终目的是要设置屏幕或打印机的像素，所以单精度完全可以满足要求了。只要舍入误差限制在一个像素的范围内，视觉效果就不会受到任何影响。另外，在某些平台上，float计算的速度比较快，并且只占据double值的一半存储量。

然而，有时候程序员处理float并不太方便，这是因为Java程序设计语言在将double值转换成float值时必须进行类型转换。例如，考虑下列的语句：

```
float f = 1.2; // Error
```

这条语句无法通过编译，因为常量1.2属于double类型，而编译器不允许丢失精度。解决的方法是给浮点常量添加一个后缀F：

```
float f = 1.2F; // Ok
```

现在，看一下这条语句：

```
Rectangle2D r = . . .
float f = r.getWidth(); // Error
```

这条语句也无法通过编译，其原因与前面一样。由于getWidth方法的返回类型是double，所以需要进行类型转换：

```
float f = (float) r.getWidth(); // Ok
```

由于后缀和类型转换都有点麻烦，所以2D库的设计者决定为每个图形类提供两个版本：一个是为那些节省空间的程序员提供的float类型的坐标；另一个是为那些懒惰的程序员提供的double类型的坐标（本书主要采用的是第二个版本，即double类型的坐标）。

这个库的设计者选择了一种古怪且在最初看起来还有些混乱的方式进行了打包。看一下Rectangle2D类，这是一个拥有两个具体子类的抽象类，这两个具体子类也是静态内部类：

```
Rectangle2D.Float
Rectangle2D.Double
```

图7-8显示了它们之间的继承示意图。

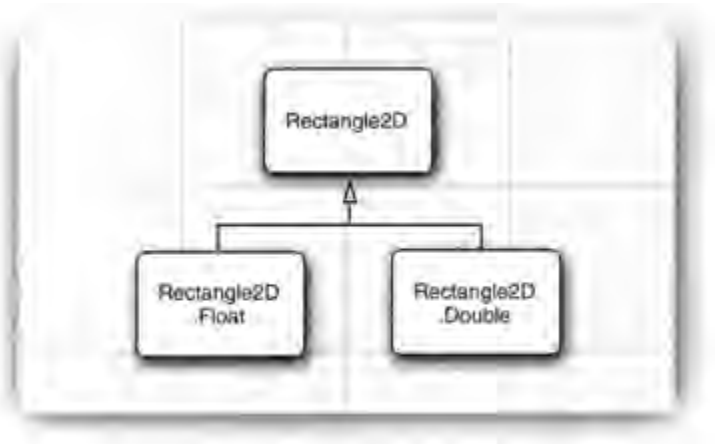

图7-8 2D矩形类

最好淡化这两个具体类是静态内部类，这样做只是为了避免使用FloatRectangle2D和DoubleRectangle2D这样的名字（有关静态内部类更详细的信息请参看第6章）。

当创建一个Rectangle2D.Float对象时，应该提供float型数值的坐标。而创建Rectangle2D.Double对象时，应该提供double型数值的坐标。

```
Rectangle2D.Float floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D.Double doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```

实际上，由于Rectangle2D.Float和Rectangle2D.Double都扩展于Rectangle2D类，并且子类只覆盖了Rectangle2D超类中的方法，所以没有必要记住图形类型。可以直接使用Rectangle2D变量保存矩形的引用。

```
Rectangle2D floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```

也就是说，只有在构造图形对象时，才需要使用烦人的内部类。

构造参数表示矩形的左上角位置、宽和高。

注释：实际上，Rectangle2D.Float类包含了一个不是由Rectangle2D继承而来的附加方法setRect(float x, float y, float h, float w)。如果将Rectangle2D.Float的引用存储在Rectangle2D变量中，那就会失去这个方法。但是，这也没有太大关系，因为在Rectangle2D中有一个参数为double类型的setRect方法。

Rectangle2D方法的参数和返回值均为double类型。例如，即使Rectangle2D.Float对象存储float类型的宽度，getWidth方法也返回一个double值。

提示：直接使用Double图形类可以避免处理float类型的值，然而如果需要创建上千个图形对象，还是应该考虑使用Float类，这样可以节省存储空间。

前面对Rectangle2D类的论述也适用于其他图形类。另外，Point2D类也有两个子类Point2D.Float和Point2D.Double。下面是构造一个点对象的方法：

```
Point2D p = new Point2D.Double(10, 20);
```

提示：Point2D类是很有用的。使用Point2D对象比使用单独的x和y更加具有面向对象的风格。许多构造器和方法都接收Point2D型参数，我们建议在可能的情况下使用Point2D对象。这样会使几何计算更容易理解。

Rectangle2D和Ellipse2D类都是由公共超类RectangularShape继承来的。无可非议，椭圆不是矩形，但它们都有着矩形边界，如图7-9所示。

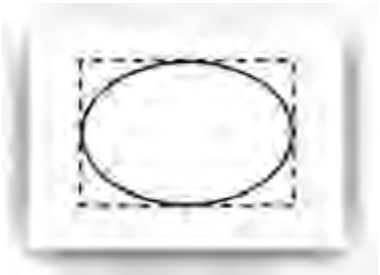

图7-9 椭圆的矩形边界

RectangularShape类定义了20多个有关图形操作的通用方法，其中比较常用的方法有getWidth、getHeight、getCenterX、getCenterY等（但在写本书时，getCenter方法还不能以Point2D对象的形式返回中心位置）。

最后，从Java 1.0遗留下来的两个类也被放置在图形类的继承层次中。它们是Rectangle和Point类，分别扩展于Rectangle2D和Point2D类，并用整型坐标存储矩形和点。

图7-10给出了图形类之间的关系。不过，省略了Double和Float子类。图中的遗留类采用填充灰色的方式标记。

Rectangle2D和Ellipse2D对象很容易构造。需要给出

- 左上角的x和y坐标；
- 宽和高。

对于椭圆，这些内容代表外接矩形。例如，

```
Ellipse2D e = new Ellipse2D.Double(150, 200, 100, 50);
```

用左上角位于（150，200）、宽100、高50的外接矩形构造一个椭圆。

然而，有时候并不知道左上角的位置。经常得到的是矩形的两个对角点，而这两个对角不一定是左上角和右下角。不能直接这样构造一个矩形：

```
Rectangle2D rect = new Rectangle2D.Double(px, py, qx - px, qy - py); // Error
```

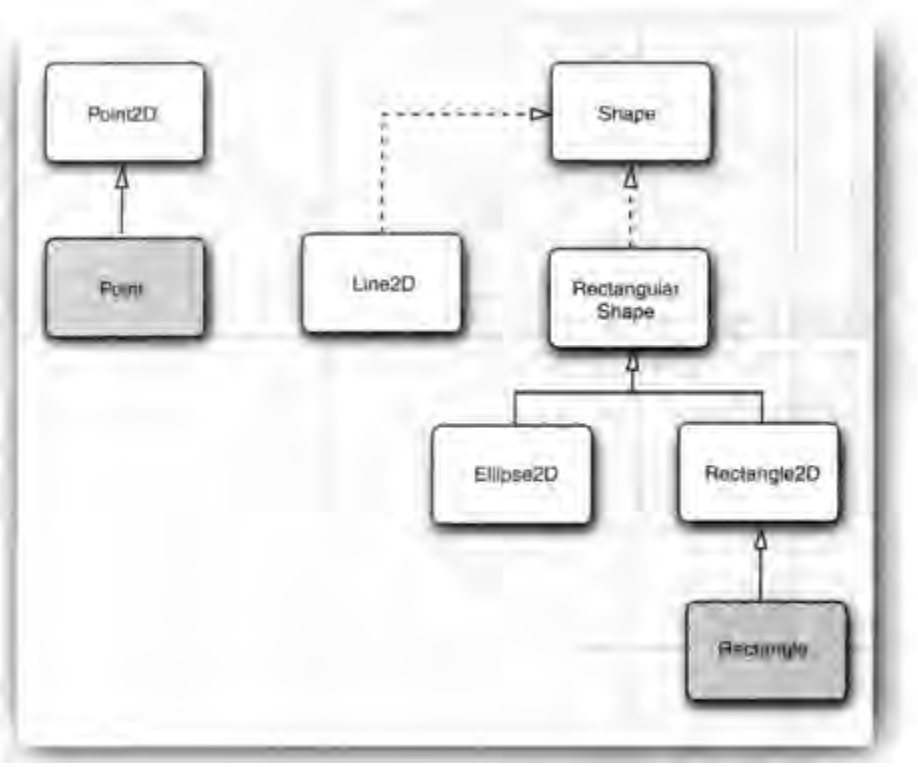

图7-10 图形类之间的关系

如果p不是左上角，两个坐标之差就为负，矩形就为空。在这种情况下，首先创建一个空矩形，然后调用setFrameFromDiagonal方法，如下所示：

```
Rectangle2D rect = new Rectangle2D.Double();
rect.setFrameFromDiagonal(px, py, qx, qy);
```

或者，如果已知的顶点分别用Point2D类型的两个对象p和q表示，就应该这样调用：

```
rect.setFrameFromDiagonal(p, q);
```

在构造椭圆（这种情况还出现在构造其他图形时）时，通常可以知道椭圆的中心、宽和高，而不是外接矩形的顶点。setFrameFromCenter方法使用中心点，但仍然要给出四个顶点中的一个。因此，通常采用下列方式构造椭圆：

```
Ellipse2D ellipse = new Ellipse2D.Double(centerX - width / 2, centerY - height / 2, width, height);
```

要想构造一条直线，需要提供起点和终点。这两个点既可以使用Point2D对象表示，也可以使用一对数值表示：

```
Line2D line = new Line2D.Double(start, end);
```

或者

```
Line2D line = new Line2D.Double(startX, startY, endX, endY);
```

例7-4中的程序绘制了一个矩形；这个矩形的内接椭圆；矩形的对角线以及以矩形中心为圆点的圆。图7-11显示了结果。

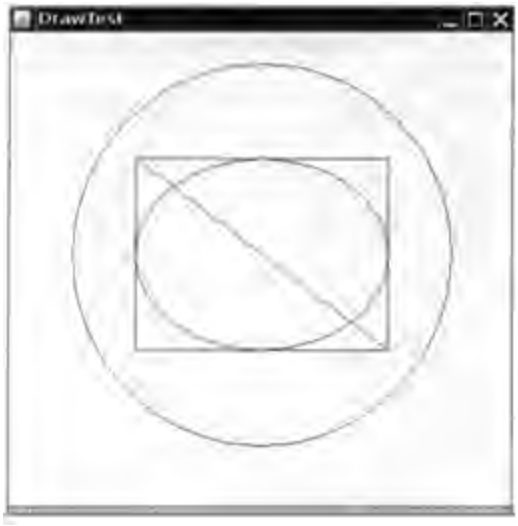

图7-11 绘制几何图形

例7-4 DrawTest.java

```
1. import java.awt.*;
2. import java.awt.geom.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.32 2007-04-14
7.  * @author Cay Horstmann
8.  */
9. public class DrawTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                DrawFrame frame = new DrawFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame that contains a panel with drawings
27.  */
28. class DrawFrame extends JFrame
29. {
30.    public DrawFrame()
31.    {
32.       setTitle("DrawTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
```

```
35.       // add panel to frame
36.
37.       DrawComponent component = new DrawComponent();
38.       add(component);
39.    }
40.
41.    public static final int DEFAULT_WIDTH = 400;
42.    public static final int DEFAULT_HEIGHT = 400;
43. }
44.
45. /**
46.  * A component that displays rectangles and ellipses.
47.  */
48. class DrawComponent extends JComponent
49. {
50.    public void paintComponent(Graphics g)
51.    {
52.       Graphics2D g2 = (Graphics2D) g;
53.
54.       // draw a rectangle
55.
56.       double leftX = 100;
57.       double topY = 100;
58.       double width = 200;
59.       double height = 150;
60.
61.       Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
62.       g2.draw(rect);
63.
64.       // draw the enclosed ellipse
65.
66.       Ellipse2D ellipse = new Ellipse2D.Double();
67.       ellipse.setFrame(rect);
68.       g2.draw(ellipse);
69.
70.       // draw a diagonal line
71.
72.       g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
73.
74.       // draw a circle with the same center
75.
76.       double centerX = rect.getCenterX();
77.       double centerY = rect.getCenterY();
78.       double radius = 150;
79.
80.       Ellipse2D circle = new Ellipse2D.Double();
81.       circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
82.       g2.draw(circle);
83.    }
84. }
```

API `java.awt.geom.RectangularShape` 1.2

- `double getCenterX()`
- `double getCenterY()`

- double getMinX()
- double getMinY()
- double getMaxX()
- double getMaxY()

返回闭合矩形的中心，以及最小、最大x和y坐标值。

- double getWidth()
- double getHeight()

返回闭合矩形的宽和高。

- double getX()
- double getY()

返回闭合矩形左上角的x和y坐标。

API java.awt.geom.Rectangle2D.Double 1.2

- Rectangle2D.Double(double x,double y,double w,double h)

利用给定的左上角、宽和高，构造一个矩形。

API java.awt.geom.Rectangle2D.Float 1.2

- Rectangle2D.Float(float x,float y,float w,float h)

利用给定的左上角、宽和高，构造一个矩形。

API java.awt.geom.Ellipse2D.Double 1.2

- Ellipse2D.Double(double x,double y,double w,double h)

利用给定的左上角、宽和高的外接矩形，构造一个椭圆。

API java.awt.geom.Point2D.Double 1.2

- Point2D.Double(double x, double y)

利用给定坐标构造一个点。

API java.awt.geom.Line2D.Double 1.2

- Line2D.Double(Point2D start, Point2D end)
- Line2D.Double(double startX, double startY, double endX, double endY)

使用给定的起点和终点，构造一条直线。

7.8 颜色

使用Graphics2D类的setPaint方法可以为图形环境上的所有后续的绘制操作选择颜色。例如：

```
g2.setPaint(Color.RED);
g2.drawString("Warning!", 100, 100);
```

只需要将调用draw替换为调用fill就可以用一种颜色填充一个封闭图形（例如：矩形或椭圆）的内部：

```
Rectangle2D rect = . . .;
g2.setPaint(Color.RED);
g2.fill(rect); // fills rect with red color
```

要想绘制多种颜色，就需要按照选择颜色、绘制图形、再选择另外一种颜色、再绘制图形的过程实施。

Color类用于定义颜色。在java.awt.Color类中提供了13个预定义的常量，它们分别表示13种标准颜色。

```
BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW
```

注释：在Java SE 1.4之前的版本中，颜色常量的名字为小写形式，例如，Color.red。这似乎有些超出寻常，因为标准编码的惯例是采用大写形式书写常量。从Java SE 1.4开始，可以采用大写的形式书写标准颜色的名字，不过，为了向后兼容，也可以用小写形式书写。

可以通过提供红、绿和蓝三色成分来创建一个Color对象，以达到定制颜色的目的。三种颜色都是用0～255（也就是一个字节）之间的整型数值表示，调用Color的构造器格式为：

```
Color(int redness, int  greenness, int blueness)
```

下面是一个定制颜色的例子：

```
g2.setPaint(new Color(0, 128, 128)); // a dull blue-green
g2.drawString("Welcome!", 75, 125);
```

注释：除了纯色以外，还可以选择更复杂的“颜料”设置，例如，改变色调（hue）或者图像。有关这方面更加详细的内容请参看卷II中的高级AWT章节。如果使用Graphics对象，而不是Graphics2D对象，就需要使用setColor方法设置颜色。

要想设置背景颜色，就需要使用Component类中的setBackground方法。Component类是JComponent类的祖先。

```
MyComponent p = new MyComponent();
p.setBackground(Color.PINK);
```

另外，还有一个setForeground方法，它是用来设定在组件上进行绘制时使用的默认颜色。

提示：从名字就可以看出，Color类中的brighter( )方法和darker( )方法的功能，它们分别加亮或变暗当前的颜色。使用brighter方法也是加亮条目的好办法。实际上，brighter( )只微微地加亮一点。要达到耀眼的效果，需要调用三次这个方法：c.brighter( ). brighter( ). brighter( )。

Java在SystemColor类中预定义了很多颜色的名字。在这个类中的常量，封装了用户系统的各个元素的颜色。例如，

```
p.setBackground(SystemColor.window)
```

它将把面板的背景颜色设定为用户桌面上所有窗口使用的默认颜色。(无论何时重新绘制窗口，

都会填充背景颜色。）当希望让绘制的用户界面元素与用户桌面上已经存在的其他元素的颜色匹配时，使用SystemColor类中的颜色非常有用。表7-1列出了系统颜色的名字和它们的含义。

表7-1 系统颜色

desktop	桌面的背景颜色	textText	文本的前景颜色
activeCaption	标题的背景颜色	textInactiveText	非活动组件的文本颜色
activeCaptionText	标题的文本颜色	textHighlight	高亮度文本的背景颜色
activeCaptionBorder	标题文本的边框颜色	textHighlightText	高亮度文本的文本颜色
inactiveCaption	非活动标题的背景颜色	control	组件的背景颜色
inactiveCaptionText	非活动标题的文本颜色	controlText	组件的文本颜色
inactiveCaptionBorder	非活动标题的边框颜色	controlLtHighlight	组件的浅高亮度颜色
Window	窗口的背景	controlHighlight	组件的高亮度颜色
windowBorder	窗口边框的颜色	controlShadow	组件的阴影颜色
windowText	窗口内的文本颜色	controlDkShadow	组件的暗阴影颜色
menu	菜单的背景颜色	scrollbar	滚动条的背景颜色
menuText	菜单的文本颜色	info	帮助区文本的颜色
text	文本的背景颜色	infoText	帮助区的文本颜色

API java.awt.Color 1.0

- `Color(int r,int g,int b)`
  创建一个颜色对象。
  参数：r 红色值（0-255）
  g 绿色值（0-255）
  b 蓝色值（0-255）
- `Color getColor()`

API java.awt.Graphics 1.0

- `void setColor(Color c)`
  获取或改变当前的颜色。所有后续的绘图操作都使用这个新颜色。
  参数：c 新颜色
- `Paint getPaint()`

API java.awt.Graphics2D 1.2

- `void setPaint(Paint p)`
  获取或设置这个图形环境的绘制属性。Color类实现了Paint接口。因此，可以使用这个方法将绘制属性设置为纯色。
- `void fill(Shape s)`
  用当前的颜料填充该图形。

API java.awt.Component 1.0

- Color getBackground()
- void setBackground(Color c)

获取或设置背景颜色。

参数：c 新背景颜色

- Color getForeground()
- void setForeground(Color c)

获取或设置前景颜色。

参数：c 新前景颜色

7.9 为文本设定特殊字体

在本章开始的“Not a Hello, World”程序中用默认字体显示了一个字符串。实际上，经常希望选用不同的字体显示文本。人们可以通过字体名（font face name）指定一种字体。字体名由“Helvetica”这样的字体家族名（font family name）和一个可选的“Bold”后缀组成。例如，“Helvetica”和“Helvetica Bold”属于“Helvetica”家族的字体。

要想知道某台特定计算机上允许使用的字体，就需要调用GraphicsEnvironment类中的getAvailableFontFamilyNames方法。这个方法将返回一个字符型数组，其中包含了所有可用的字体名。GraphicsEnvironment类描述了用户系统的图形环境，为了得到这个类的对象，需要调用静态的getLocalGraphicsEnvironment方法。下面这个程序将打印出系统上的所有字体名：

```
import java.awt.*;

public class ListFonts
{
   public static void main(String[] args)
   {
      String[] fontNames = GraphicsEnvironment
         .getLocalGraphicsEnvironment()
         .getAvailableFontFamilyNames();
      for (String fontName : fontNames)
         System.out.println(fontName);
   }
}
```

在某个系统上，输出的结果为：

```
Abadi MT Condensed Light
Arial
Arial Black
Arial Narrow
Arioso
Baskerville
Binner Gothic
. . .
```

后面还有70种左右的字体。

字体名可以商标化，字体设计在一些权限内可以版权化。因此，字体的分发需要向字体的

创始者付版税。当然，像名牌香水有廉价仿制品一样，字体也有外观相似的。例如，Helvetica的仿制品就是Windows中称为Arial的字体。

为了创建一个公共基准，AWT定义了五个逻辑（logical）字体名：

```
SansSerif
Serif
Monospaced
Dialog
DialogInput
```

这些字体将被映射到客户机上的实际字体。例如，在Windows系统中，SansSerif将被映射到Arial上。

另外，Sun JDK包含3种字体，它们是“Lucida Sans”,“Lucida Bright”和“Lucida Sans Typewriter”。

要想使用某种字体绘制字符，必须首先利用指定的字体名、字体风格和字体大小来创建一个Font类对象。下面是构造一个Font对象的例子：

```
Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
```

第三个参数是以点数目计算的字体大小。点数目是排版中普遍使用的表示字体大小的单位，每英寸包含72个点。

在Font构造器中，提供字体名的位置也可以给出逻辑字体名称。另外，利用Font构造器的第二个参数可以指定字体的风格（常规、加粗、斜体或加粗斜体），下面是几个字体风格的值：

```
Font.PLAIN
Font.BOLD
Font.ITALIC
Font.BOLD + Font.ITALIC
```

注释：字体映射定义在Java安装的jre/lib子目录中的fontconfig.properties文件中。有关这个文件的详细内容请参看http://java.sun.com/javase/6/docs/guide/intl/fontconfig.html。

可以读取TrueType或PostScriot Type 1格式的字体文件。这需要一个字体输入流——通常从磁盘文件或者URL读取（有关流的更详细信息请参看卷II第1章）。然后调用静态方法Font.createFont：

```
URL url = new URL("http://www.fonts.com/Wingbats.ttf");
InputStream in = url.openStream();
Font f1 = Font.createFont(Font.TRUETYPE_FONT, in);
```

上面定义的字体为常规字体，大小为1。可以使用deriveFont方法得到希望大小的字体：

```
Font f = f1.deriveFont(14.0F);
```

警告：deriveFont方法有两个重载版本。一个（有一个float参数）设置字体的大小；另一个（有一个int参数）设置字体风格。所以f.deriveFont(14)设置的是字体风格，而不是大小（其结果为斜体，因为14的二进制表示的是ITALIC，而不是BOLD）。

Java字体包含了通用的ASCII字符和符号。例如，如果用Dialog字体打印字符‘\u2297’，那么就会看到⊗字符。只有在Unicode字符集中定义的符号才能够使用。

下面这段代码将使用系统中14号加粗的标准sans serif字体显示字符串“ Hello,World ”：

```
Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
g2.setFont(sansbold14);
String message = "Hello, World!";
g2.drawString(message, 75, 100);
```

接下来，将字符串绘制在面板的中央，而不是任意位置。因此，需要知道字符串占据的宽和高的像素数量。这两个值取决于下面三个因素：

- 使用的字体（在前面列举的例子中为sans serif，加粗，14号）；
- 字符串（在前面列举的例子中为 ” Hello,World ”）；
- 绘制字体的设备（在前面列举的例子中为用户屏幕）。

要想得到屏幕设备字体属性的描述对象，需要调用Graphics2D类中的getFontRenderContext方法。它将返回一个FontRenderContext类对象。可以直接将这个对象传递给Font类的getStringBounds方法：

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);
```

getStringBounds方法将返回包围字符串的矩形。

为了解释这个矩形的大小，需要清楚几个排版的相关术语。如图7-12所示。基线（baseline）是一条虚构的线，例如，字母“ e ”所在的底线。上坡度（ascent）是从基线到坡顶（ascenter）的距离。例如，“ b ”和“ k ”以及大写字母的上面部分。下坡度（descent）是从基线到坡底（descenter）的距离，坡底是“ p ”和“ g ”这种字母的底线。

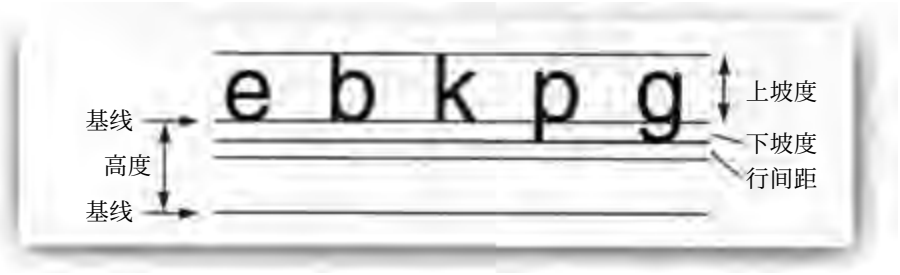

图7-12 排版术语解释

行间距（leading）是某一行的坡底与其下一行的坡顶之间的空隙（这个术语源于排字机分隔行的引导带）。字体的高度是连续两个基线之间的距离，它等于下坡度+行间距+上坡度。

getStringBounds方法返回的矩形宽度是字符串水平方向的宽度。矩形的高度是上坡度、下坡度、行间距的总和。这个矩形始于字符串的基线，矩形顶部的y坐标为负值。因此，可以采用下面的方法获得字符串的宽度、高度和上坡度：

```
double stringWidth = bounds.getWidth();
double stringHeight = bounds.getHeight();
double ascent = -bounds.getY();
```

如果需要知道下坡度或行间距，可以使用Font类的getLineMetrics方法。这个方法将返回一个LineMetrics类对象，获得下坡度和行间距的方法是：

```
LineMetrics metrics = f.getLineMetrics(message, context);
```

```
float descent = metrics.getDescent();
float leading = metrics.getLeading();
```

下面这段代码使用了所有这些信息，将字符串显示在包围它的组件中央：

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);

// (x,y) = top left corner of text
double x = (getWidth() - bounds.getWidth()) / 2;
double y = (getHeight() - bounds.getHeight()) / 2;

// add ascent to y to reach the baseline
double ascent = -bounds.getY();
double baseY = y + ascent;
g2.drawString(message, (int) x, (int) baseY);
```

为了能够获得中央的位置，可以使用getWidth( )得到组件的宽度。使用bounds.getWidth( )得到字符串的宽度。前者减去后者就是两侧应该剩余的空间。因此，每侧剩余的空间应该是这个差值的一半。高度也是一样。

注释：如果需要在paintComponent方法外部计算布局图的尺度，不能从Graphics2D对象得到字体绘制环境。换作调用JComponent类的getFontMetrics方法，而后紧接着调用getFontRenderContext:

```
FontRenderContext context = getFontMetrics(f).getFontRenderContext();
```

为了说明位置是正确的，示例程序绘制了基线和包围这个字符串的矩形。图7-13给出了屏幕显示结果。例7-5是程序清单。

图7-13 绘制基线和字符串边框

例7-5 FontTest.java

```
1. import java.awt.*;
2. import java.awt.font.*;
3. import java.awt.geom.*;
4. import javax.swing.*;
5.
6. /**
7.  * @version 1.33 2007-04-14
8.  * @author Cay Horstmann
9.  */
10. public class FontTest
11. {
```

```
12.    public static void main(String[] args)
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                FontFrame frame = new FontFrame();
19.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                frame.setVisible(true);
21.             }
22.          });
23.    }
24. }
25.
26. /**
27.  * A frame with a text message component
28.  */
29. class FontFrame extends JFrame
30. {
31.    public FontFrame()
32.    {
33.       setTitle("FontTest");
34.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.
36.       // add component to frame
37.
38.       FontComponent component = new FontComponent();
39.       add(component);
40.    }
41.
42.    public static final int DEFAULT_WIDTH = 300;
43.    public static final int DEFAULT_HEIGHT = 200;
44. }
45.
46. /**
47.  * A component that shows a centered message in a box.
48.  */
49. class FontComponent extends JComponent
50. {
51.    public void paintComponent(Graphics g)
52.    {
53.       Graphics2D g2 = (Graphics2D) g;
54.
55.       String message = "Hello, World!";
56.
57.       Font f = new Font("Serif", Font.BOLD, 36);
58.       g2.setFont(f);
59.
60.       // measure the size of the message
61.
62.       FontRenderContext context = g2.getFontRenderContext();
63.       Rectangle2D bounds = f.getStringBounds(message, context);
64.
65.       // set (x,y) = top-left corner of text
66.
67.       double x = (getWidth() - bounds.getWidth()) / 2;
68.       double y = (getHeight() - bounds.getHeight()) / 2;
```

```
69.
70.       // add ascent to y to reach the baseline
71.
72.       double ascent = -bounds.getY();
73.       double baseY = y + ascent;
74.
75.       // draw the message
76.
77.       g2.drawString(message, (int) x, (int) baseY);
78.
79.       g2.setPaint(Color.LIGHT_GRAY);
80.
81.       // draw the baseline
82.
83.       g2.draw(new Line2D.Double(x, baseY, x + bounds.getWidth(), baseY));
84.
85.       // draw the enclosing rectangle
86.
87.       Rectangle2D rect = new Rectangle2D.Double(x, y, bounds.getWidth(), bounds.getHeight());
88.       g2.draw(rect);
89.    }
90. }
```

API `java.awt.Font 1.0`

- `Font(String name, int style, int size)`
  创建一个新字体对象。
  参数：name　字体名。不是字体名（例如，“ Helvetica Bold ”），就是逻辑字体名（例如，“ Serif ”、“ SansSerif ”）
  style　字体风格（Font.PLAIN、Font.BOLD、Font.ITALIC或Font.BOLD+Font.ITALIC）
  size　字体大小（例如，12）
- `String getFontName()`
  返回字体名，例如，“ Helvetica Bold ”。
- `String getFamily()`
  返回字体家族名，例如，“ Helvetica ”。
- `String getName()`
  如果采用逻辑字体名创建字体，将返回逻辑字体，例如，“ SansSerif ”；否则，返回字体名。
- `Rectangle2D getStringBounds(String s, FontRenderContext context) 1.2`
  返回包围这个字符串的矩形。矩形的起点为基线。矩形顶端的y坐标等于上坡度的负值。矩形的高度等于上坡度、下坡度和行间距之和。宽度等于字符串的宽度。
- `LineMetrics getLineMetrics(String s, FontRenderContext context) 1.2`
  返回测定字符串宽度的一个线性metrics对象。
- `Font deriveFont(int style)  1.2`
- `Font deriveFont(float size)  1.2`

- `Font deriveFont(int style, float size)` 1.2
  返回一个新字体，除给定大小和字体风格外，其余与原字体一样。

API `java.awt.font.LineMetrics` 1.2

- `float getAscent()`
  返回字体的上坡度——从基线到大写字母顶端的距离。
- `float getDescent()`
  返回字体的下坡度——从基线到坡底的距离。
- `float getLeading()`
  返回字体的行间距——从一行文本底端到下一行文本顶端之间的空隙。
- `float getHeight()`
  返回字体的总高度——两条文本基线之间的距离（下坡度+行间距+上坡度）。

API `java.awt.Graphics` 1.0

- `Font getFont()`
- `void setFont(Font font)`
  获取或设置当前的字体。这种字体将被应用于后续的文本绘制操作中。
  参数：font　　一种字体
- `void drawString(String str, int x, int y)`
  采用当前字体和颜色绘制一个字符串。
  参数：str　　将要绘制的字符串
  x　　字符串开始的x坐标
  y　　字符串基线的y坐标

API `java.awt.Graphics2D` 1.2

- `FontRenderContext getFontRenderContext()`
  返回这个图形文本中，指定字体特征的字体绘制环境。
- `void drawString(String str, float x, float y)`
  采用当前的字体和颜色绘制一个字符串。
  参数：str　　将要绘制的字符串
  x　　字符串开始的x坐标
  y　　字符串基线的y坐标

API `javax.swing.JComponent` 1.2

- `FontMetrics getFontMetrics(Font f)` 5.0
  获取给定字体的度量。FontMetrics类是LineMetrics类的早先版。

API java.awt.FontMetrics 1.0

- FontRenderContext getFontRenderContext() 1.2
  返回字体的字体绘制环境。

7.10 图像

到目前为止，我们已经看到了如何通过绘制直线和图形创建一个简单的图像。而对于照片这样的复杂图像来说，通常都是由扫描仪或特殊的图像处理软件生成的（正像在卷II中将看到的，逐像素地生成图像，并将结果存储到数组中也是可以的。这种方式通常用于生成不规则碎片的图像）。

一旦图像保存在本地文件或因特网的某个位置上，就可以将它们读到Java应用程序中，并在Graphics对象上进行显示。在Java SE 1.4中，读取一个图像十分简单。如果图像存储在本地文件中，就应该调用：

```
String filename = "...";
Image image = ImageIO.read(new File(filename));
```

否则，应该提供URL：

```
String urlname = "...";
Image image = ImageIO.read(new URL(urlname));
```

如果图像不可用，read方法将抛出一个IOException。在第11章中，将讨论有关异常处理的问题。而在目前的例子程序中只捕获异常，并打印出栈的轨迹。

这里的变量image包含了一个封装图像数据的对象引用。可以使用Graphics类的drawImage方法将图像显示出来。

```
public void paintComponent(Graphics g)
{
   . . .
   g.drawImage(image, x, y, null);
}
```

例7-6又前进了一步，它在一个窗口中平铺显示了一幅图像。屏幕显示的结果如图7-14所示。这里采用paintComponent方法来实现平铺显示。它的基本过程为：先在左上角显示图像的一个拷贝，然后使用copyArea将其拷贝到整个窗口：

图7-14 平铺图像的窗口

```
for (int i = 0; i * imageWidth <= getWidth(); i++)
  for (int j = 0; j * imageHeight <= getHeight(); j++)
     if (i + j > 0)
        g.copyArea(0, 0, imageWidth, imageHeight, i * imageWidth, j * imageHeight);
```

例7-6列出了图像显示程序的完整代码。

例7-6 ImageTest.java

```
1. import java.awt.*;
2. import java.io.*;
3. import javax.imageio.*;
```

```
4. import javax.swing.*;
5.
6. /**
7.  * @version 1.33 2007-04-14
8.  * @author Cay Horstmann
9.  */
10. public class ImageTest
11. {
12.    public static void main(String[] args)
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                ImageFrame frame = new ImageFrame();
19.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                frame.setVisible(true);
21.             }
22.          });
23.    }
24. }
25.
26. /**
27.  * A frame with an image component
28.  */
29. class ImageFrame extends JFrame
30. {
31.    public ImageFrame()
32.    {
33.       setTitle("ImageTest");
34.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.
36.       // add component to frame
37.
38.       ImageComponent component = new ImageComponent();
39.       add(component);
40.    }
41.
42.    public static final int DEFAULT_WIDTH = 300;
43.    public static final int DEFAULT_HEIGHT = 200;
44. }
45.
46. /**
47.  * A component that displays a tiled image
48.  */
49. class ImageComponent extends JComponent
50. {
51.    public ImageComponent()
52.    {
53.       // acquire the image
54.       try
55.       {
56.          image = ImageIO.read(new File("blue-ball.gif"));
57.       }
58.       catch (IOException e)
59.       {
60.          e.printStackTrace();
```

```
61.       }
62.    }
63.
64.    public void paintComponent(Graphics g)
65.    {
66.       if (image == null) return;
67.
68.       int imageWidth = image.getWidth(this);
69.       int imageHeight = image.getHeight(this);
70.
71.       // draw the image in the top-left corner
72.
73.       g.drawImage(image, 0, 0, null);
74.       // tile the image across the component
75.
76.       for (int i = 0; i * imageWidth <= getWidth(); i++)
77.          for (int j = 0; j * imageHeight <= getHeight(); j++)
78.             if (i + j > 0) g.copyArea(0, 0, imageWidth, imageHeight, i * imageWidth, j
79.                   * imageHeight);
80.    }
81.
82.    private Image image;
83. }
```

API `javax.imageio.ImageIO 1.4`

- `static BufferedImage read(File f)`
- `static BufferedImage read(URL u)`

从给定文件或URL上读取图像。

API `java.awt.Graphics 1.0`

- `boolean drawImage(Image img, int x, int y, ImageObserver observer)`

绘制一幅非比例图像。注意：这个调用可能会在图像还没有绘制完毕就返回。

参数：img 将要绘制的图像
x 左上角的x坐标
y 左上角的y坐标
observer 绘制进程中以通告为目的的对象（可能为null）。

- `boolean drawImage(Image img,int x,int y,int width,int height,ImageObserver observer)`

绘制一幅比例图像。系统按照比例将图像放入给定宽和高的区域。注意：这个调用可能会在图像还没有绘制完毕就返回。

参数：img 将要绘制的图像
x 左上角的x坐标
y 左上角的y坐标
width 描述图像的宽度

height　　描述图像的高度
observer　　绘制进程中以通告为目的的对象（可能为null）

- `void copyArea(int x,int y,int width,int height,int dx,int dy)`
  拷贝屏幕的一块区域。
  参数：x　　原始区域左上角的x坐标
  y　　原始区域左上角的y坐标
  width　　原始区域的宽度
  height　　原始区域的高度
  dx　　原始区域到目标区域的水平距离
  dy　　原始区域到目标区域的垂直距离

本章总结了有关Java图形编程的内容介绍。关于更高级的技术，可以参看卷II中有关2D图形和图像的操作。在下一章，读者将学习如何让程序对用户的输入进行响应。