第4章

外观（Facade）模式

面向对象编程的最大优势，在于它能够防止应用程序成为混乱纠缠的小块程序。理想状态下，面向对象系统应该是将其他可复用的工具类中的行为组织在一起的最小的类。工具包或子系统的开发人员通常会将设计良好的类组织为包，而不是提供应用程序。Java 类库中的包通常可以当做工具包（toolkits），以保证我们能够随意地开发各种特定领域的应用程序。

伴随工具包复用性而来的问题是：面向对象子系统中类的多样性可能导致选择太多，以至于希望使用该工具包的开发人员变得茫然失措。诸如 Eclipse 这样的集成开发环境（IDE），可以将开发人员从包的错综复杂中解脱出来，然而，IDE 有时却会产生大量开发人员不需要的代码。

简化工具包的另一途径是使用外观（facade）模式。只需少量代码，就能提供典型的、无修饰用法的类库中的类。一个外观就是一个类，它包含的功能介于工具包与完整的应用程序之间，为工具包或子系统的类提供了简单的用法。

外观模式的意图是为子系统提供一个接口，便于它的使用。

外观类、工具类和示例类

外观类可能全是静态方法，在 UML 中，这样的类被称为 utility（工具）。稍后，我们将介

绍一个全为静态方法的 UI（用户界面）类，尽管这样做可能会影响到将来子类对这些方法的重写。

示例类（Demo）则用于展示如何使用类或子系统。以本例而言，示例类提供了和外观类相同的价值。

挑战 4.1

写出示例类和外观类的两点区别。

答案参见第 302 页

`javax.swing` 包包含了 `JOptionPane`，以便弹出一个标准对话框。如下代码会反复显示对话框，直到用户单击了 Yes 按钮，如图 4.1 所示。

```
package app.facade;
import javax.swing.*;
import java.awt.Font;

public class ShowOptionPane {
    public static void main(String[] args) {
        Font font = new Font("Dialog", Font.PLAIN, 18);
        UIManager.put("Button.font", font);
        UIManager.put("Label.font", font);

        int option;
        do {
            option = JOptionPane.showConfirmDialog(
                        null,
                        "Had enough?",
                        "A Stubborn Dialog",
                        JOptionPane.YES_NO_OPTION);
        } while (option == JOptionPane.NO_OPTION);
    }
}
```

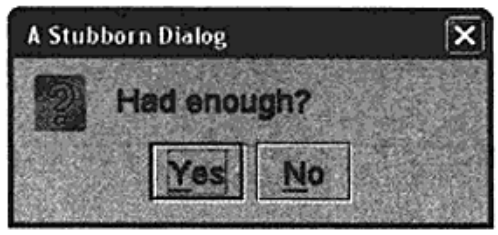

图 4.1　JOptionPane 类使显示这样的对话框变得很容易

挑战 4.2

JOptionPane 类让对话框的显示变得简单，那么，该类是一个外观类、工具类还是示例类？给出答案，并阐述你的理由。

答案参见第 303 页

挑战 4.3

Java 类库中很少有外观类，为什么？

答案参见第 303 页

重构到外观模式

外观模式常见于一些常规的应用程序开发。如果根据关注点将代码分解为不同的类，就可以提取一个类，它的主要职责是为子系统提供简便的访问方式，从而完成对系统的重构。考虑 Oozinoz 公司早期的一个例子，当时并没有 GUI 开发的规范。假设你读到程序员写的一段程序，用于展示一颗哑弹的飞行路径，如图 4.2 所示。

设计的焰火弹会在高空爆炸，发出绚烂的光芒。但是偶尔，某个焰火弹却不会爆炸（即哑弹），而是缓慢坠落到地面。与火箭不同，焰火弹是没有动力的。倘若忽略风和空气的阻力，一颗哑弹的飞行路径就是一条简单的抛物线。图 4.3 展示了执行 ShowFlight.main()后的一个窗口截图。

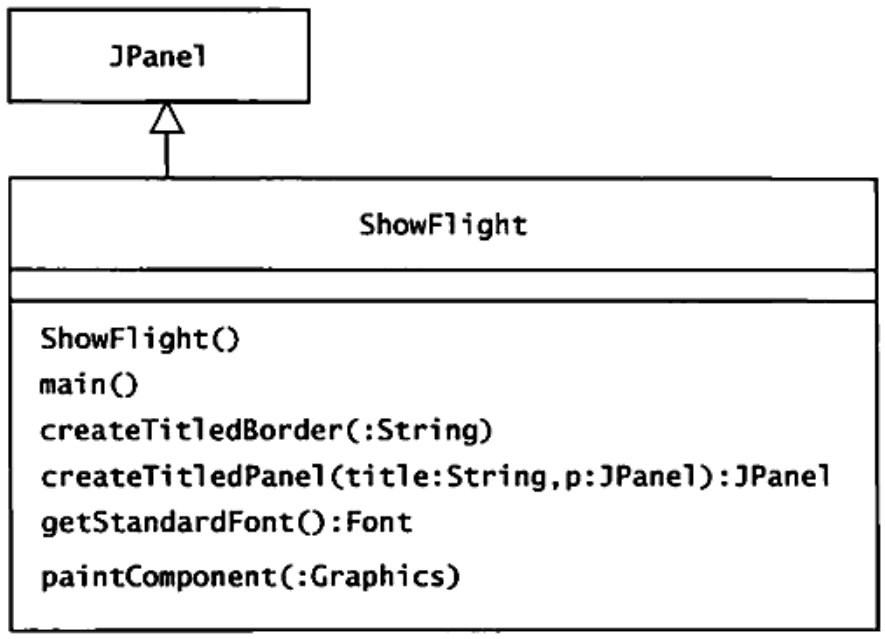

图 4.2　ShowFlight 类显示了哑弹的飞行路径

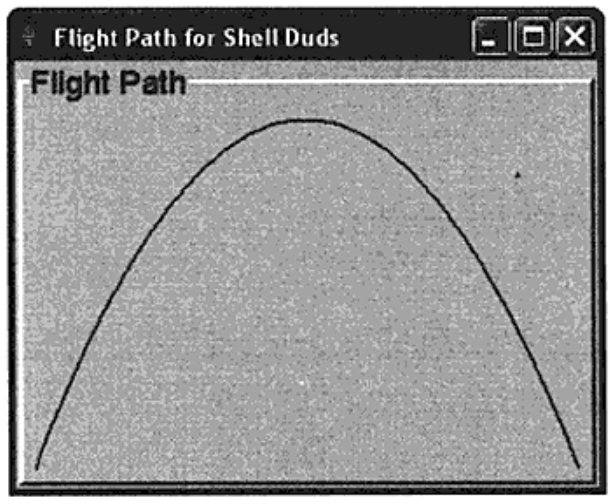

图 4.3　ShowFlight 应用展示了一颗哑弹将在何处落地

ShowFlight 类存在一个问题：它混杂了三个功能。它的首要功能是为飞行路径提供一个面板，第二个功能是作为一个完整的应用程序，需要将飞行路径的面板显示在一个具有标题的边框内，最后一个功能是计算飞行路径的抛物线。ShowFlight 类定义了 paintComponent()方法执行计算，代码如下。

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g); // 绘制背景
    int nPoint = 101;
    double w = getWidth() - 1;
    double h = getHeight() - 1;
```

```
        int[] x = new int[nPoint];
        int[] y = new int[nPoint];
        for (int i = 0; i < nPoint; i++) {
            // t从0到1
            double t = ((double) i) / (nPoint - 1);
            // x从0到w
            x[i] = (int) (t * w);
            // 当t=0和1时，y为h；当t=0.5时，y为0
            y[i] = (int) (4 * h * (t - .5) * (t - .5));
        }
        g.drawPolyline(x, y, nPoint);
    }
```

如需了解代码中如何建立哑弹轨迹坐标的x和y值，请参考下页关于“参数方程”的介绍。

这里无须类的构造函数。它使用静态工具方法去包装面板的标题，并定义了标准字体。

```
public static TitledBorder createTitledBorder(String title){
    TitledBorder tb = BorderFactory.createTitledBorder(
        BorderFactory.createBevelBorder(BevelBorder.RAISED),
        title,
        TitledBorder.LEFT,
        TitledBorder.TOP);
    tb.setTitleColor(Color.black);
    tb.setTitleFont(getStandardFont());
    return tb;
}

public static JPanel createTitledPanel(
    String title, JPanel in) {
    JPanel out = new JPanel();
    out.add(in);
    out.setBorder(createTitledBorder(title));
    return out;
}

public static Font getStandardFont() {
    return new Font("Dialog", Font.PLAIN, 18);
}
```

注意，createTitledPanel()方法把提供的控件放进一个斜边框里，以提供一个小的间隙

(padding)，保证飞行曲线不碰到容器的边缘。Main()方法也为包含了应用程序控件的表单对象增加了间隙。

```
public static void main(String[] args) {
    ShowFlight flight = new ShowFlight();
    flight.setPreferredSize(new Dimension(300, 200));
    JPanel panel = createTitledPanel("Flight Path", flight);

    JFrame frame = new JFrame("Flight Path for Shell Duds");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(panel);

    frame.pack();
    frame.setVisible(true);
}
```

图 4.3 展示了该程序的执行过程。

参数方程

当你需要画一条曲线时，很难把 y 值表示为 x 值的函数。参数方程可以让你把 x 和 y 定义成其他参数的函数。特别的，你可以将曲线的绘制时间表示为 0～1 之间的参数 t，并且可以将 x 和 y 定义成参数 t 的函数。

例如，你想要哑弹的飞行抛物线穿过 Graphics 对象，假设该对象的宽度是 w，那么关于 x 的参数方程可以简单表示为：

$$x = w * t$$

注意，t 的变化范围是 0～1，x 的变化范围是 0～w。

抛物线的 y 值随着 t 的平方值变化而变化，方向越往下，y 值就会随之而递增。对于抛物线，当 t=0.5 时，y 应该为 0。

因此我们能够得到如下方程：

$$y = k * (t - .5) * (t - .5)$$

这里，k 代表一个指定的常量。该方程规定了当 t=0.5 时，y=0，并且当 t=0 或 t=1 时，y=h，也就是显示区域的高度。经过一些代数运算，可以推导出如下完整的关于 y 的方程：

```
y = 4 * h * (t - .5) * (t - .5)
```

图 4.3 显示了方程式绘出的抛物线。

参数方程的另一个优点是，我们可以用它绘制出一个给定 x 值多个 y 值的曲线。比如绘制一个圆。半径为 1 的圆的方程为：

```
x² + y² = r²
```

或者

```
y = +- sqrt (r²- x²)
```

如果每个 x 值对应两个 y 值，情况会更加复杂。要正确地调整 Graphics 对象的高和宽，并进行绘制，则较为困难。因此，可以利用极坐标来简化绘制圆的功能。

```
x = r * cos(theta)
y = r * sin(theta)
```

这两个参数方程将 x 和 y 表示成新参数 *theta* 的函数。*Theta* 表示绘制圆的过程中扫过的弧度，范围是 0~2*pi。可以设置一个圆的半径，使它能塞在高度为 h，宽度为 w 的图形对象中。下面是一些在 Graphics 对象中绘制圆的参数方程：

```
theta = 2 * pi * t
r = min(w, h)/2
x = w/2 + r * cos(theta)
y = h/2 - r * sin(theta)
```

将这些方程转换为代码后，就可以产生如图 4.4 所示的圆（可以在 oozinoz.com 获得生成这一显示效果的 ShowCircle 应用程序代码）。

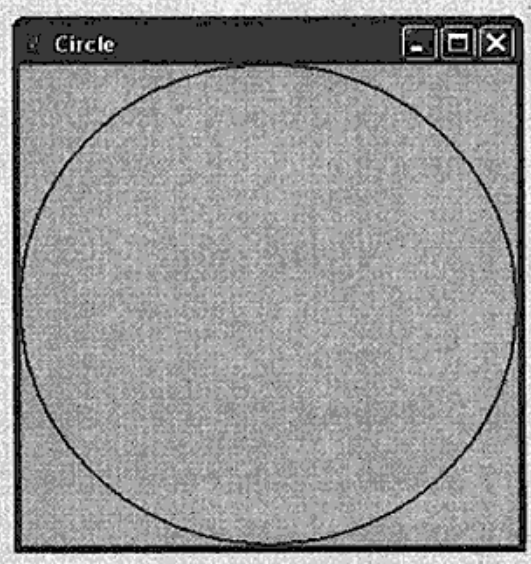

图 4.4　当一个 x 值对应多个 y 值时，参数方程可以简化对曲线的建模

下面绘制圆的代码是对数学公式的直接翻译。由于像素在水平方向和垂直方向的范围分别是 0 到 width-1 和 0 到 height-1，因此我们在代码中相应减小了 Graphics 对象的高和宽。

```
package app.facade;

import javax.swing.*;
import java.awt.*;

import com.oozinoz.ui.SwingFacade;

public class ShowCircle extends JPanel {
    public static void main(String[] args) {
        ShowCircle sc = new ShowCircle();
        sc.setPreferredSize(new Dimension(300, 300));
        SwingFacade.launch(sc, "Circle");
    }
protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int nPoint = 101;
        double w = getWidth() - 1;
        double h = getHeight() - 1;
        double r = Math.min(w, h) / 2.0;
        int[] x = new int[nPoint];
        int[] y = new int[nPoint];
        for (int i = 0; i < nPoint; i++) {
            double t = ((double) i) / (nPoint - 1);
            double theta = Math.PI * 2.0 * t;
            x[i] = (int) (w / 2 + r * Math.cos(theta));
            y[i] = (int) (h / 2 - r * Math.sin(theta));
        }
        g.drawPolyline(x, y, nPoint);
    }
}
```

根据 t 定义 x 和 y 函数，可以分别计算 x 和 y 的值。这比把 y 定义成 x 的函数要简单，也便于将 x 和 y 映射到 Graphics 对象坐标中。另外，当 y 是 x 的非单值函数时，参数方程也可以简化曲线的绘制。

ShowFlight类可以工作，然而根据关注点分离的原则，我们应将其重构为多个单独的类，以提高可维护性以及可重用性。假设你正在进行设计评审，并且决定做出如下改变：

- 引入一个Function类，它定义了f()方法，接收一个double值（时间值），返回一个double值（函数的值）。
- 将ShowFlight类移入PlotPanel类，改为使用Function对象来获取x和y的值。定义PlotPanel构造函数接收两个Function实例和绘图所需的点数。
- 将createTitledPanel()方法移到已存在的UI工具类中，来实现一个像当前ShowFlight类那样带有标题的面板。

挑战 4.4

完成图4.5所示的类图，用以展示将ShowFlight重构为三个类的代码：Function类、实现两个参数功能的PlotPanel类和一个UI外观类。在你的重新设计中，让类ShowFlight2为获取y值创建一个Function，并让main()方法启动程序。

答案参见第303页

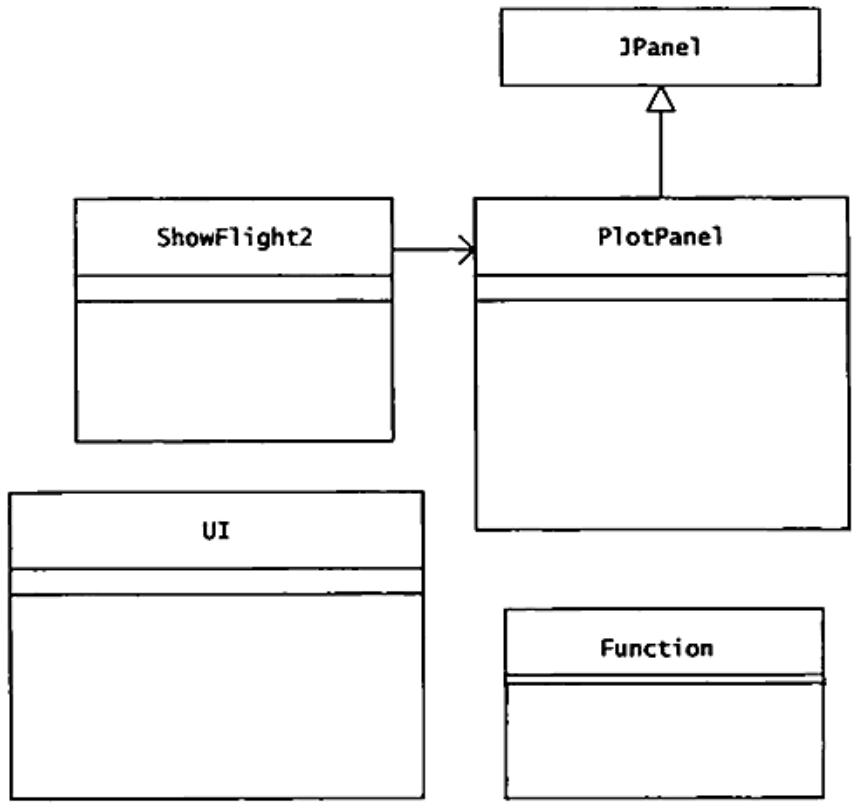

图4.5 被重构为多个类的飞行轨迹应用程序，每个类都有各自的职责

重构之后，Function 类定义了参数方程。倘若要创建一个包含 Function 类与其他类型的 com.oozinoz.function 包，则 Function.java 的核心代码可能是：

```
public abstract double f(double t);
```

经过重构，PlotPanel 类只拥有了一个职责：显示一对参数方程，代码如下：

```
package com.oozinoz.ui;

import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JPanel;
import com.oozinoz.function.Function;

public class PlotPanel extends JPanel {
    private int points;
    private int[] xPoints;
    private int[] yPoints;

    private Function xFunction;
    private Function yFunction;

    public PlotPanel(
            int nPoint, Function xFunc, Function yFunc) {
        points = nPoint;
        xPoints = new int[points];
        yPoints = new int[points];
        xFunction = xFunc;
        yFunction = yFunc;
        setBackground(Color.WHITE);
    }

    protected void paintComponent(Graphics graphics) {
        double w = getWidth() - 1;
        double h = getHeight() - 1;

        for (int i = 0; i < points; i++) {
            double t = ((double) i) / (points - 1);
            xPoints[i] = (int) (xFunction.f(t) * w);
```

```
                yPoints[i] = (int) (h * (1 - yFunction.f(t)));
            }

            graphics.drawPolyline(xPoints, yPoints, points);
        }
    }
```

注意，PlotPanel 类目前是 com.oozinoz.ui 包的一部分，和 UI 类处于同一位置。对 ShowFlight 类进行重构后，UI 类也拥有了 createTitledPanel()和 createTitledBorder()方法。UI 类逐步演化为外观类，使之更容易使用 Java 控件。

使用这些控件的应用程序可能是一个很小的类，该类唯一的职责就是对这些控件进行布局并显示它们。例如，类 ShowFlight2 的代码如下：

```
package app.facade;

import java.awt.Dimension;
import javax.swing.JFrame;
import com.oozinoz.function.Function;
import com.oozinoz.function.T;
import com.oozinoz.ui.PlotPanel;
import com.oozinoz.ui.UI;

public class ShowFlight2 {
    public static void main(String[] args) {
        PlotPanel p = new PlotPanel(
            101,
            new T(),
            new ShowFlight2().new YFunction());
        p.setPreferredSize(new Dimension(300, 200));

        JFrame frame = new JFrame(
            "Flight Path for Shell Duds");
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(
            UI.NORMAL.createTitledPanel("Flight Path", p));

        frame.pack();
```

```
        frame.setVisible(true);
    }


    private class YFunction extends Function {
        public YFunction() {
            super(new Function[] {});
        }

        public double f(double t) {
            // 当 t 等于 0 或 1 时，y 为 0；当 t 等于 0.5 时，y 为 1
            return 4 * t * (1 - t);
        }
    }
}
```

ShowFlight2 类为哑弹的飞行路径提供了 YFunction 类。main()方法用来布局和显示用户界面。这个类的运行结果和最初的 ShowFlight 类相同。但是，现在你拥有了一个可重用的外观类，它可以简化 Java 应用程序中图形用户界面的创建。

小结

整体而言，应该对子系统中的类进行重构，直到每个类都有一个明确的目的。这可以使你的代码更容易维护，但也可能让使用该子系统的用户变得无所适从。为了让调用这些代码的开发人员使用更方便，可以为子系统提供示例程序或者外观类。通常，示例程序可以独立运行，却无法重用，仅用于演示使用子系统的方法。外观类则是可配置、可重用的类，提供了高层次的接口，使得子系统的使用更加方便。