第2章

接口型模式介绍

抽象地讲，类的接口是类允许其他类对象访问的方法与字段集。接口通常代表一种承诺，即方法需要实现接口方法名表示的操作，遵循代码注释、测试和其他文档说明。类的实现就是位于方法体中的代码。

Java 将接口概念提升为独立的结构，体现了接口（对象必须遵循的承诺）与实现（对象如何履行承诺）的分离。Java 接口允许多个类提供相同的功能，也允许一个类同时实现多个接口。

许多设计模式都使用了 Java 内建的这一特性。例如，运用适配器（Adapter）模式，通过使用一个接口类型来适配类的接口，从而满足客户的需要。若要用好 Java 基本的内建特性，就要从接口开始，确保自己掌握了 Java 特性的工作原理。

接口与抽象类

最初的 *Design Patterns* 一书总是提到抽象类的使用，而对于接口的使用只字不提。这是因为编写该书实例的 C++与 Smalltalk 语言根本就没有接口结构。由于 Java 接口与抽象类非常相似，这一点并不影响 Java 开发人员对该书的理解。

挑战 2.1

写出在 Java 中抽象类和接口的三点区别。

答案参见第 297 页

如果不能使用接口，完全可以像 C++那样使用抽象类。然而，作为一种独立的结构，接口在 *n* 层应用程序开发过程中的地位举足轻重。

让我们考虑一个火箭仿真类所必须实现的接口定义。工程师设计了许多不同的火箭，包括固体燃料火箭和液体燃料火箭，这两种火箭有着完全不同的弹道学原理。不管火箭如何组成，对火箭的仿真都必须提供预期推力与质量等数据。下面是 Oozinoz 定义的火箭仿真接口：

```
package com.oozinoz.simulation;

public interface RocketSim {
    abstract double getMass();
    public double getThrust();
    void setSimTime(double t);
}
```

挑战 2.2

下面的描述哪些是正确的？

A. 尽管只有 getMass()方法被显式声明为抽象，但 RocketSim 接口的三个方法都是抽象方法。

B. 尽管只有 getThrust()方法被显式声明为公开，但 RocketSim 接口的三个方法都是公开方法。

C. 接口虽然被声明为“公有接口”，但即使省略 public 关键字，接口仍然是公有的。

D. 可以创建另外一个接口，例如 RocketSimSolid，去扩展 RocketSim 接口。

E. 每个接口必须至少包含一个方法。

F. 接口可以声明实例字段，实现该接口的类也必须声明该字段。

G. 虽然不能实例化接口，但接口仍然可以定义构造函数，并要求实现类必须提供具有相同签名的构造函数。

答案参见第 297 页

接口与职责

Java 接口的优势在于它限制了对象之间的协作，这种约束其实提供了更大的自由。即使实现接口的类的实现发生了巨大变化，接口的客户端仍然可以不受影响。

开发人员在创建 `RocketSim` 接口的实现类时，有责任编写 `getMass()`和 `getThrust()`方法，返回火箭的性能指标。换言之，开发人员必须遵循这些方法制订的契约。

有时候，接口指定的方法并不要求必须为调用者提供服务。在某些情况下，实现类可以忽略这些调用，为实现的方法提供一个空的方法体。

挑战 2.3

请列举一个接口，它包含的方法并不要求实现该接口的类必须返回值，或者代表调用者执行若干操作。

答案参见第 298 页

如果创建的接口指定了一系列用于通知的方法，则可以考虑提供桩（stub），即提供空实现的接口实现类。开发者通过实现桩的子类，重写那些对应用程序具有重要意义的接口方法。`java.awt.event` 中的 `WindowAdapter` 就是这样一个例子，如图 2.1 所示。（若要进一步了解 UML，请参见附录 D）。`WindowAdapter` 类实现了 `WindowListener` 接口的所有方法，但这些实现都是空的，方法不含任何语句。

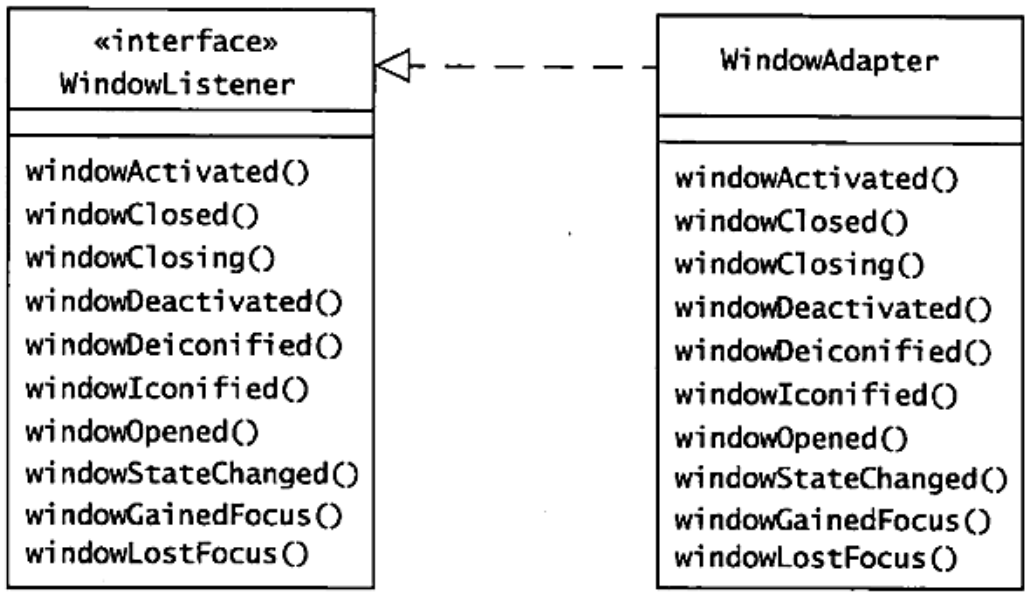

图 2.1　WindowAdapter 类注册监听器，可以方便地监听 window 事件，从而使我们忽略自己并不关心的事件

除了声明方法外，接口也可以声明常量。在下面的例子中，ClassificationConstants 声明了两个常量[译注1]，实现该接口的类可以访问它们。

```
public interface ClassificationConstants {
    static final int CONSUMER = 1;
    static final int DISPLAY = 2;
}
```

接口与抽象类还有一点关键区别：虽然类只能声明扩展一个类，却可以声明实现多个接口。

小结

接口的威力在于它描述了在类协作中它所期望与不期望的行为。接口与抽象类很相似，定义行为却并不提供实现。

熟练掌握接口的概念与细节，有助于我们更好地运用 Java 接口。这种强有力的结构是许多健壮设计与设计模式的核心。

译注1：在 JDK 1.5 引入 enum 类型后，通常不建议在接口中声明常量，而是尽量将具有分组类别的常量定义为 enum 类型。

超越普通接口

如果能恰如其分地运用 Java 接口，就可以简化并完善我们的设计。有时候，接口的设计会超出我们对普通接口的定义与使用。表 2.1 展示了不同场景下应使用的模式。

表 2.1　不同场景下应使用的模式

如果你期望	可运用模式
• 适配类的接口以匹配客户端期待的接口	适配器模式
• 为一组类提供一个简单接口	外观模式
• 为单个对象与复合对象提供统一的接口	合成模式
• 解除抽象与实现之间的耦合，使得二者能够独立演化	桥接模式

每个设计模式都旨在解决不同场景的问题。面向接口的模式适用于需要对一个类或一组类的方法进行定义或重定义的场景。例如，某个类实现了我们所需要的服务，但它的方法名称却与客户端的期望不符，这就需要运用适配器模式。