JHotDraw 让你成为程序设计的毕加索

JHotDraw 让你成为程序设计的毕加索

——使用可高度定制化的 GUI 框架来简化图形应用程序的开发

软件开发人员总是希望能又快又好的进行应用软件的开发。而使用应用框架正是一种能在减少开发时间的同时又能提高软件质量的方法。应用框架被设计用于重用；它们能够提供预制的组件作为你所要开发的系统的构件，并提供设计模式作为系统架构的蓝图。

很多 Java 程序员都在频繁的使用某些框架，不过他们自己可能并没有意识到这一点。JFC Swing 可以被看作是一种用于生成通用的 GUI 程序的简单框架。虽然 Swing 可以在许多应用中派上用场，但是对于那些基于 GUI 的应用程序来说，它缺乏清楚的结构。在这方面更专业的一个框架是 JHotDraw，它主要用于绘制那些技术类和结构化的图像方面的程序。这类图像包括网络布局图和 pert 图。而且 JHotDraw 为开发这些图像的编辑器提供了很好的支持。在这个应用领域，JHotDraw 证明了应用框架的威力和用处。在具体的讨论 JHotDraw 之前，我们首先来谈论下框架和其体现的概念。

设计模式和框架的理论

当开发者使用框架的时候，他们并不是仅仅重用代码，而且重用了一个原型程序的设计和架构。通常，框架需要经过一些调整来满足问题领域的需要。框架并不像代码库一样仅仅只是提供组件，它还提供了用于组合这些组件的结构、组件之间的相互操作的方式，还常常包括应用程序的基本结构。这种结构并不像从类库中得到的那些结构那样被动，通过调用一些用户定义的组件，它常常可以颠覆控制整个控制流程。

开发者在软件设计时经常面对的问题常常是那些在某些典型的特定情况下重复出现的问题。使用设计模式，就是在面对这些问题时使用一些已经被证明正确可行的解决方案。设计模式描述了它所要解决的问题，问题的上下文，和可重用的解决方案。通常对于所要解决的问题，我们会给它有一个名字，我们可以用这个名字和其他开发者交流该解决方案。

应用框架通常会依赖于设计模式来帮助获得一个灵活的通用的应用程序设计。设计模式通过间接和抽象等手段，使得开发者可以很方便的把自己的类和组件添加到系统中去。

在应用框架的帮助下，开发者可以及时的完成应用程序的开发，应用程序可以按照用户的需要进行定制，而且开发者还可以享受到成熟框架带来的健壮性和稳定性。不过，这也是有代价的——你必须要学习和理解如何和应用框架进行交互，甚至你还要学习应用框架的缺点。大部分的框架都具有很高层次的抽象，是很复杂的软件产品。理解一个框架可能很困难，而对框架进行调试就更是艰巨了。虽然框架提供了某些进行定制的机制，但是这也可能给你带来了某些限制或者某些技术上的特殊要求，就算你要执行的功能只是稍微脱离该框架的范围，这个缺点也可能会特别突出。

在你使用应用框架之前，很重要的一点就是要明白这些事情：它的好处和不足、它的目标应用领域、它的组件和结构、开发流程、基础的设计模式和编程技术。

JHotDraw 的描述

与 JFC Swing 不同，JHotDraw 为一个基于 GUI 的编辑器提供了下面这些东西：整合在一个工具调色板中的各种工具、不同的视图、用户自定义的图形元素、还有对图像保存、载入和打印的支持。我们可以通过继承或者组合某个组件的方式来对应用框架进行定制。

除了 the main drawing window 之外，JHotDraw 对其他 windows 的支持甚少，比如它就不支持文本编辑器。不过你在有了一些 JHotDraw 的结构的知识之后，应该能很容易的扩展

这个框架来支持那些被忽略掉的功能。运行例子程序的时候，你就会知道用 JHotDraw 开发的典型程序是什么样子的了（译者注：在 JHotDraw 的主页上可以下载 JHotDraw 的源代码，其中包括例子程序）。比如说，JavaDraw（译者注：这是 JHotDraw 源程序中的一个例子程序）就是一个标准的绘图程序，它能让我们大约知道 JHotDraw 大概能能干什么。你可以在你解压 JHotDraw（译者注：JHotDraw 主页 www.jhotdraw.org）的所生成的目录下通过输入以下的命令启动 JavaDraw：

java CH.ifa.draw.samples.javadraw.JavaDrawApp

另外，CH.ifa.draw.samples.pert.PertApplication 也展示了一些 JHotDraw 定制化的可能性。

从软件工程的角度看，JHotDraw 也是很有趣的。JHotDraw 一开始是由 Kent Beck 和 Ward Cunningham 用 Smalltalk 开发的，它是第一个公开声言要以应用框架的设计来进行重用的开发项目。它在很早的时候就以设计模式的方式来编制文档了，所以在设计模式社区颇有影响力。Erich Gamma 和 Thomas Eggenschwiler 开发了 JHotDraw 的最初版本。

这篇文章谈论了 JHotDraw 一个新的版本——5.2（译者注：在翻译这篇文章的时候已经有了6.1beta 了），在这个版本里，原来的 AWT 组件被换成了相应的 JFC Swing 组件。它也支持一些新的 JFC Swing 特性，比如内部窗体、滚动条、工具条和弹出式菜单等。因此，JHotDraw 作为一个专用的 GUI 框架，它还是基于的是 Swing 框架提供的通用的 GUI 设施的，不过在这些设施之上，JHotDraw 又提供自己的一些特性和功能。

程序包组织

所有的 JHotDraw 类和接口都按照他们的功能放在不同的程序包中。CH.ifa.draw.framework 里存放的是核心组件所需要的大部分接口，这些接口描述了核心组件的责任、功能和内部操作。你可以在 CH.ifa.draw.standard 里找到这些接口的标准实现。在 CH.ifa.draw.figures 和 CH.ifa.draw.contrib 里你能找到一些附加的功能。桌面应用程序和 applet 程序的结构被分别定义在 CH.ifa.draw.application 和 CH.ifa.draw.applet 中。

JHotDraw 的结构

如果我们更仔细的关注这些程序包，特别是那些核心的框架包，我们就能了解 JHotDraw 的结构，然后揭示出这些组件每一部分所担当的角色了。

任何使用 JHotDraw 的应用程序都有一个用于画图的窗口。这个画图窗口——DrawWindow，是一个编辑器窗口，它是 javax.swing.JFrame 的子类。这个窗口有一个或更多个内部窗体，每个窗体都与一个 Drawing View 相连。DrawingView 是 javax.swing.JPanel 的子类，它是一个可以显示图像和接受用户输入的区域。图形的改变会传播到 DrawingView，然后 DrawingView 负责更新这些图像。图形由 figure 组成，每个 figure 又能作为其他 figure 的容器。每个 figure 有一个 handle，它用于定义访问点，决定该 figure 如何与其他 figure 交互（比如说，将这个 figure 与其他 figure 相连接）。在 DrawingView 里，你可以选中一个或者多个 figure，然后处理它们。DrawWindows 的本身有一个从工具调色板中得来的活动的工具，这个工具可以对当前 DrawingView 的图形进行操作。

JHotDraw 的典型开发过程

下面的列表里的是使用 JHotDraw 开发应用程序时经常涉及到的任务。这些任务关注于如何将 JHotDraw 整合到你的应用程序中，如何让你用本文“一个简单应用程序的问题描述”章节所提到的对象模型一起工作。

1. 为你的应用程序创建你自己的图形元素和符号。可能你经常都需要定义自己的图形元素。很幸运的，这已经提前定义好了几种图形元素：AbstractFigure、CompositeFigure、AttributeFigure。你可以通过继承这些类，然后重定义某些方法，比如 draw（），来重定义它们的行为，从而达到在图表中定制图形表示的目的。通常情况下，这些图形元素应该和你在应用程序中使用的对象保持一致，至少要有某种联系。
2. 开发你自己的工具，用于创建图形元素，并按照应用的需要来对图形元素进行加工。JHotDraw 再次带来了便利，它已经给我们准备好了一些工具：比如说创建工具、联接工具、选择工具和文字工具。通过继承这些类并重定义其中的某些方法，比如像 mouseUP（）、mouseDown（），你就可以定义你的程序的交互过程，执行你的应用程序所需要的任务——比如加工在你的应用程序中所定义的对象。
3. 生成实际的 GUI，并将其整合进你的程序中去。毫不奇怪，JHotDraw 已经有了一个基本的应用程序框架了：或者是一个基本的 DrawApplication，或者是一个支持很多内部 frame 的 MDI_DrawApplication，再或者是一个 DrawApplet。你可以通过重定义 createMenus（）、createFileMenu（）和其他的办法来定义你自己的菜单，通过重定义 createTools（）来插入新的工具。最后在运行时实例化你的应用程序然后调用 open（），一个完整的 GUI 就产生了。
4. 使用 javac 编译你的应用程序。记住，把所有 JHotDraw 需要的的程序包在调用 java 或者 javac 命令之前加入到 classpath 中去。

这些任务里有些涉及到某些设计模式的应用，本文将在迟些时候进行具体的讨论。

一个简单应用程序的问题描述

在使用一个框架之前，知道它的目标应用领域和它是如何解决在这个领域出现的问题是很重要的（如果有一个目标应用领域存在的话）。

在本文里，我们讨论一个名叫 JModeller 的简单的类图编辑器的开发过程，这可以看作是 JHotDraw 的简单应用。JModeller 帮助设计类图，将软件架构制作成文档。它支持类之间的联系、聚合、以来和继承关系。除了这些基本的设计结构和图形的保存、载入和打印功能之外，这个编辑器就没有别的更高级的特性了。

在这个类图编辑器里的为了管理类所涉及对象模型是相当简单的。在这个模型里的主要的类是 JModellerClass，它代表了一个类，有类名、属性和方法组成。如果类之间的关联、依赖和继承关系变的复杂，你可以在模型中添加一个专用的类来跟踪这些关系的属性和行为。目前为止这并不需要，所以由类本身来存储类间关系的信息。而且，有专门的图形元素用于绘制类、关联线、依赖线或者是继承线。用于开发一个类编辑器的对象模型和图形编辑器使用的用于类图设计的对象模型是不同的，读者不应混淆。图形编辑器里使用的具体的对象模型只是存储一些用于类图的信息和一些图形信息。很明显，用 JHotDraw 构建的应用程序可以很好的满足上面所叙述的要求。

使用 JHotDraw 的设计模式

现在我将向读者展示如何利用 JHotDraw 里的设计模式类开发类图编辑器。

MVC 框架

首先，请读者清楚的一点就是 JHotDraw 是基于 MVC 模型的，这个模型将应用逻辑和用户界面分离开。View 通常用于在用户界面上显示信息；而 controller 用于处理用户的交互然后将其映射到应用功能上。Model 部分作为 view 和 controller 部分的基础，由应用逻辑和数据组成。在数据上的任何变化都会通知到 View 上的。

复合设计模式

正如本文之前提到的，用于在一个类图里存储信息的对象模型是很简单的。你只需要一个简单的 JModellerClass，代码可以在 JModellerClass.java 里找到。在类图里表示类的图形

元素更有趣些。它们有许多基本图形元素不能满足的要求。所以我们创建了一个新的图形元素：GraphicalCompositeFigure；这个图形元素整合了好几种已经存在的图形元素的优点，特别是 TextFigure，它被用于显示类的名字、属性和方法。JHotDraw 已经提供了一种图形元素叫做 CompositeFigure，它可以将很多种图形元素整合在一起。

复合模式的结构图

就如这个图形元素的名字一样，相应的设计模式就叫做复合设计模式。它和其他的一些设计模式一样，都在 Erich Gamma 的那本《*Design Patterns*》 书里有所叙述。在复合设计模式背后的思想就是在一个容器里有很多个属于相同基本类型的组件，但这个容器本身也可以被当作一个单独的组件。容器方法调用里的所有行为被容器委托到它里面的组件上。通常情况下，客户端的组件不会意识到是在和元素的组合体在打交道，它将其当作一个单独的组件。这种封装技术使得我们可以用一个组件的继承层次结构来创建一个像 CompositeFigure 这样的复合体，在这个结构中的所有组件就像是一个单元整体的样子来与别的模块交互。有趣的是，StandardDrawing 是 CompositeFigure 的子类，这意味着一副图像可以包含其他的图形元素，而它本身也是一个图形元素。

对于复合体如何把一个行为分派给它里面的组件，可以找到一个很好的例子，就是 CH.ifa.draw.standard.CompositeFigure 的 draw（）方法

```
/**
 * Draws all the contained figures
 * @see Figure#draw
 */
public void draw(Graphics g) {
    FigureEnumeration k = figures();
    while (k.hasMoreElements())
        k.nextFigure().draw(g);
}
```

通常情况下 CompositeFigure 并没有自己的图形表示，它只是简单的显示它所复合的其他 Figure。不过一个类的图形表示必须要能考虑到它的全部图形表示，并能让它的的图形容器也显示出来。GraphicalCompositeFigure 是一个 CompositeFigure，它也同样的把它自己的图形表示功能委托给一个专用的图形元素上。如果一个 AttributeFigure（或者是它的子类）被用作一个图形元素，那 GraphicalCompositeFigure 就也可以用来保存想字体信息和颜色了。

```
public class GraphicalCompositeFigure extends CompositeFigure {
    ....
    /**
     * Return the display area. This method is delegated to the encapsulated presentation figure.
     */
    public Rectangle displayBox() {
        return getPresentationFigure().displayBox();
    }
     /**
      * Draw the figure. This method is delegated to the encapsulated presentation figure.
      */
     public void draw(Graphics g) {
        getPresentationFigure().draw(g);
        super.draw(g);
```

```
    }
    ....
}
```

现在你所需要做的就是创建一个 ClassFigure 类，它从 GraphicalCompositeFigure 里继承了所有需要的行为。ClassFigure 类里包含一个 TextFigure 用于存放类名，还有其他的用于存放属性和方法的不同图形元素。一个在构造器里的 CH.ifa.draw.figures.RectangleFigure 被用于 draw a box around the whole container figure.

```
public class ClassFigure extends GraphicalCompositeFigure {
    private JModellerClass          myClass;
    private GraphicalCompositeFigure myAttributesFigure;
    private GraphicalCompositeFigure myMethodsFigure;
    ...
    public ClassFigure() {
        this(new RectangleFigure());
    }
    public ClassFigure(Figure newPresentationFigure) {
        super(newPresentationFigure);
    }
    /**
     * Hook method called to initialize a ClassFigure.
     * It is called from the superclass' constructor and the clone() method.
     */
    protected void initialize() {
        // start with an empty Composite
        removeAll();
        // create a new Model object associated with this View figure
        setModellerClass(new JModellerClass());
        // create a TextFigure responsible for the class name
        setClassNameFigure(new TextFigure() {
            public void setText(String newText) {
                super.setText(newText);
                getModellerClass().setName(newText);
                update();
            }
        });
        // add the TextFigure to a Composite
        GraphicalCompositeFigure nameFigure = new GraphicalCompositeFigure(new
SeparatorFigure());
        nameFigure.add(getClassNameFigure());
        ...
        add(nameFigure);
        // create a figure responsible for maintaining attributes
        setAttributesFigure(new GraphicalCompositeFigure(new SeparatorFigure()));
        ...
```

```
        add(getAttributesFigure());
        // create a figure responsible for maintaining methods
        setMethodsFigure(new GraphicalCompositeFigure(new SeparatorFigure()));
        ...
        add(getMethodsFigure());
        setAttribute(Figure.POPUP_MENU, createPopupMenu());
        super.initialize();
    }
    ...
}
```

策略设计模式

用于表示 ClassFigure 类的图形元素只负责本元素的绘制；这个图形元素并不知道怎么放置 ClassFigure，也不知道 ClassFigure 中的该怎么安排各个子组件之间的图形显示。实际上，图形表示和布局管理的算法是相互独立的。结果，布局算法被从 ClassFigure 中分离出来，并被封装成一个外部的类，不过它对这个 ClassFigure 有广泛的访问权限，并能对其进行控制。如果我们要放置一个 ClassFigure，这项任务会被委托给 CH.ifa.draw.contrib.FigureLayoutStrategy 类来完成，这个类有遍历组件所有子组件并且放置他们的逻辑步骤。……

布局管理器通过布局表来执行其算法

状态设计模式

在 JHotDraw 里，工具调色板里有很多工具，你可以用他们来选择、加工或者创建一个图形元素。在某些时候，你可能还需要定义自己的工具来完成别的功能。正如我们所看到的，一个 ClassFigure 包含很多个 TextFigure 用于保存类名、属性和方法的信息。不过不幸的是，CH.ifa.draw.standard.SelectionTool 只能激活被选中的容器，它并不包含任何 TextFigures。如果你想通过双击某个 ClassFigure 来编辑它所包含的 TextFigures，那可以使用 CH.ifa.draw.contrib.CustomSelectionTool。这个类能够实现你的目的，而且还会帮你处理图形元素的弹出窗口。我们在这个编辑器里使用的选择工具就是源于这个类，然后重定义了它的 handleMouseDoubleClick 和 handleMouseClick 方法。如果发生了双击的时间，这个工具会激活一个 CH.ifa.draw.figures.TextTool 用于负责对文本的编辑。

```
public class DelegationSelectionTool extends CustomSelectionTool {
    private TextTool myTextTool;
    public DelegationSelectionTool(DrawingView view) {
        super(view);
        setTextTool(new TextTool(view, new TextFigure()));
    }
    protected void handleMouseDoubleClick(MouseEvent e, int x, int y) {
        Figure figure = drawing().findFigureInside(e.getX(), e.getY());
        if ((figure != null) && (figure instanceof TextFigure)) {
            getTextTool().activate();
            getTextTool().mouseDown(e, x, y);
        }
    }
    protected void handleMouseClick(MouseEvent e, int x, int y) {
```

```
        deactivate();
    }
    public void deactivate() {
        super.deactivate();
        if (getTextTool().isActivated()) {
            getTextTool().deactivate();
        }
    }
     ...
}
```

在画板上进行操作的选择工具是属于 drawing view 的。因此，drawing view 总是有一个处于活动状态的工具。当改变工具的时候，它的行为和与用户交互的方式也改变了。换句话说，工具就是在 drawing view 的上下文环境中的一种状态。在理想状态下，drawing view 应该和实际的工具是相互独立的，所以工具之间应该是可以相互替换，我们也不需要用复杂的 if/switch 语句来区分这些工具。在一个状态设计模式里，你使用代表状态的对象来讲状态和状态的上下文区分开，而这个代表状态的对象定义了一个上下文可以使用的接口。因此，在一个以 drawing view 为上下文的环境里，引入一个 DelegationSelectionTool 工具，只是引入了一个另一个新的状态而已。Drawing view 还是像往常一样接受用户的输入，不过将这些输入委托给工具来处理。工具知道该如何来处理用户的输入，它会相应的执行某些任务。

工具决定了 StandardDrawingView 操作的状态

状态模式看起来有点像是策略模式，它们都将行为委托给某个特定的对象，不过它们的目的是不一样的。策略模式将对象和算法之间解耦，使得算法可以被重用。而状态模式将内在的行为分离和提取出来，使得其可以容易的扩展和相互替换。

模板设计模式

类图显示类，而类之间是有关系的。AssociationLineConnection 元素代表了两个类之间的线性联系，它可以变成直接的联系或者是聚合关系。InheritanceLineConnection 用一条从子类指向父类的带箭头的直线表示一种继承关系。CH.ifa.draw.figures.LineConnection 类通过提供 connectEnd() and disconnectEnd()来实现模板方法。

在 LineConnection 中的模板方法设计模式和钩子方法

模板方法又是一种设计模式，它定义了一个总是要执行的命令序列。在这个命令序列中，另外一些方法总是在某个确定的时候被调用。于是子类可以在不改变总体的行为的情况下添加和具体，应用相关的指令了。因此 AssociationLineConnection 和 InheritanceLineConnection 只需要对这些需要的钩子函数进行重定义了——handleConnect() 和 handleDisconnect()，从而可以建立起两个类之间的关系。下面的代码演示了这点：

```
public class InheritanceLineConnection extends LineConnection {
    ...
    /**
     * Hook method to plug in application behavior into
     * a template method. This method is called when a
     * connection between two objects has been established.
     */
    protected void handleConnect(Figure start, Figure end) {
```

```
        super.handleConnect(start, end);
        JModellerClass startClass = ((ClassFigure)start).getModellerClass();
        JModellerClass endClass = ((ClassFigure)end).getModellerClass();
        startClass.addSuperclass(endClass);
    }
    ...
}
```

在建立连接之前，模板方法必须要执行一系列的步骤。首先调用 canConnect 测试两个图形元素是否可以连接，然后设置开始和结束端的图形元素，最后调用钩子方法 handleConnect

模板方法调用钩子方法 handleConnect（）

另一个模板方法的例子可以在 CH.ifa.draw.standard.AttributeFigure 里找到，它其中的 draw（）方法定义了绘图的程序，在这个方法当中会调用 drawBackground()和 drawFrame（）方法，而这两个方法会被诸如 CH.ifa.draw.figures.RectangleFigure 之类的类重定义。

工厂方法

工厂方法是《*Design Patterns*》一书里记述的另一种设计模式。你可以将工厂方法想像成一个在构造过程中的特殊的钩子方法。这个设计模式让你可以创建定制化的组件。这个方法在 JHotDraw 里被广泛的使用，特别是在创建如菜单和工具等用户界面组件的时候。你可以在 CH.ifa.draw.application.DrawApplication 里找到大量的工厂方法，它们有诸如 createTools（）或者是 createMenus()之类的方法，或者，更进一步的，叫做 createFileMenu(), createEditMenu()等等名字。根据你想定制的粒度，你可以将合适的创建方法进行改变。为了插入用于创建类或者用于类之间的关联和继承关系的工具，你可以在你的应用程序里重定义 createToo 方法。

```
public class JModellerApplication extends MDI_DrawApplication {
    ...
    public JModellerApplication() {
        super("JModeller - Class Diagram Editor");
    }
    /**
     * Create the tools for the toolbar. The tools are
     * a selection tool, a tool to create a new class and
     * two tools to create association and inheritance
     * relationships between classes.
     *
     * @param    palette toolbar to which the tools should be added
     */
    protected void createTools(JToolBar palette) {
        super.createTools(palette);
        Tool tool = new ConnectedTextTool(view(), new TextFigure());
        palette.add(createToolButton(IMAGES+"ATEXT", "Label", tool));

        tool = new CreationTool(view(), new ClassFigure());
        palette.add(createToolButton(DIAGRAM_IMAGES+"CLASS", "New Class", tool));
        tool = new ConnectionTool(view(), new AssociationLineConnection());
```

```
        palette.add(createToolButton(IMAGES+"LINE", "Association Tool", tool));
        tool = new ConnectionTool(view(), new DependencyLineConnection());
        palette.add(createToolButton(DIAGRAM_IMAGES+"DEPENDENCY", "Dependency
Tool", tool));
        tool = new ConnectionTool(view(), new InheritanceLineConnection());
        palette.add(createToolButton(DIAGRAM_IMAGES+"INHERITANCE", "Inheritance
Tool", tool));
    }
    ...
}
```

你必须重定义 createSelectionTool()方法来创建你自己的 createSelectionTool：

```
protected Tool createSelectionTool() {
    return new DelegationSelectionTool(view());
}
```

原型设计模式

你可能已经留意到了在 createTool 方法里，每种工具都是以工具所要产生的一个图形元素的实例来初始化的。在 JHotDraw 里，每一个创建工具都是利用一个原始的图形元素的实例来创建重复的实例的。……。在 CH.ifa.draw.standard.AbstractFigure 里定义了基本的 clone()机制，在这个方法里，运用 Java 的串行化机制对对象进行了复制。

串行化是一种很简单的方法，它将整个对象结构先写出，然后再读入。这会创建原始图形元素的深复制副本，其中包括了所有引用对象的副本。因此，原始图形元素和它的副本并不共享任何对象引用。比如说，ClassFigure 并不会串行化任何想联系的上下文菜单，这些菜单必须在反串行化的时候被初始化。

```
private void readObject(ObjectInputStream s)
    // call superclass' private readObject() indirectly
    s.defaultReadObject();
    // Create popup menu after it has been read
    setAttribute(Figure.POPUP_MENU, createPopupMenu());
}
```

而且，读取和写入图形及其所包含的图形元素也会简单的调用串行化机制。JHotDraw……。每个图形元素都实现了 CH.ifa.draw.util.Storable 接口，它们各自提供了用于如何将他们自己写入磁盘和读取出来的实现。被写入的信息包括类名和所有属性的值，这些值以 String 的形式被写入。很明显，属性值被写入的顺序是很重要的。利用类名，JHotDraw 使用 Java 的反射机制产生一个新的图形实例。在这种情况下，空参数列表的默认构造器会被调用。

Mian 方法

```
……JModellerApplication……
/**
 * Start the application by creating an instance and open
 * the editor window.
 */
public static void main(String[] args) {
    JModellerApplication window = new JModellerApplication();
```

```
    window.open();
}
```

使用 JModeller

JModeller 的使用是相当简单的。这其中只有很少的障碍需要进一步的简短解释。首先，每一个 ClassFigure 都有一个上下文菜单，它让你可以为类加入属性和方法。通过双击类的名字、属性或方法，你可以编辑上面的文本。空的属性或方法名代表将那个属性或者方法删除，空的类名则代表将整个类删除。AssociationLineConnection 类同样也有一个上下文菜单，它让你可以把关联改为聚合，可以在 bidirectional and unidirectional 联系之间转换。你可以在 ConnectedTextTool 工具的帮助下在类的关系上加上注释。通过使用合适的连接工具来点选连接线，你可以在任何两个类的关系上增加一个处理点。最后，在你编译或者运行 JModeller 的时候，你必须保证 jhotdraw.jar 在你的 classpath 里。比如说，在 Windows 下你可以通过下面这样的命令行命令来编译 JModeller：

```
javac -classpath "%CLASSPATH%;/jhotdraw.jar;." *.java
```

通过这样的命令你可以启动程序：

```
java -classpath "%CLASSPATH%;/jhotdraw.jar;." JModellerApplication
```

里必须是你解压 JHotDraw 后产生的目录。

结论

要想正确的使用一个框架，你必须先理解它。设计模式是一种可以达到高度可复用的软件架构的方法，它也非常适合用于将架构写成文档以方便别的开发者理解。JHotDraw 就是一个框架的好例子，它由一些基础的设计模式组成，比如……。了解了在这些模式之后的基本概念和这些模式在 JHotDraw 里的应用会帮助你决定如何定制和调整这个框架以满足你的应用程序的需求。虽然学习一个框架需要额外的努力，在一开始甚至会拖慢你的进度，不过像 JHotDraw 这样一个框架通常都在可以缩短开发时间的同时提高你的软件质量。当然，JModeller 只是 JHotDraw 强大能力的一个小小的例子，希望这篇文章会是你亲自动手实验体会这一点的开始。

本文来自:http://article.yeeyan.org/view/14599/4185