第19章

备忘录（Memento）模式

有时候，你想要创建的对象已经在系统中存在。例如，当用户执行撤销操作，使系统回滚到之前的某一状态，或重新执行之前搁置的工作时，就会出现这种情形。

备忘录模式的意图是为对象状态提供存储和恢复功能。

经典范例：使用备忘录模式执行撤销操作

第17章的抽象工厂模式介绍了一个可视化的应用，用户可以通过一个工厂对象使用材料进行操作建模实验。假定撤销按钮的功能尚未实现，我们就可以运用备忘录模式。

事实上，备忘录就是存储状态信息的对象。在可视化应用程序中，我们需要保存的是应用程序的状态。无论何时添加或移动机器，用户都可以通过单击 Undo 按钮取消操作。为了给可视化应用程序添加撤销功能，需要考虑如何在一个备忘录中捕获应用程序的状态。同时，我们还必须决定何时去捕获状态，如何恢复程序到之前捕获到的状态。当应用程序启动后，会出现如图 19.1 所示的界面。

程序最初启动时是一个空白状态。任何时刻只要程序处在这种空白状态下，Undo 按钮都是禁用的。经过执行一些添加或拖曳操作，可视化应用程序变成如图 19.2 所示的样子。

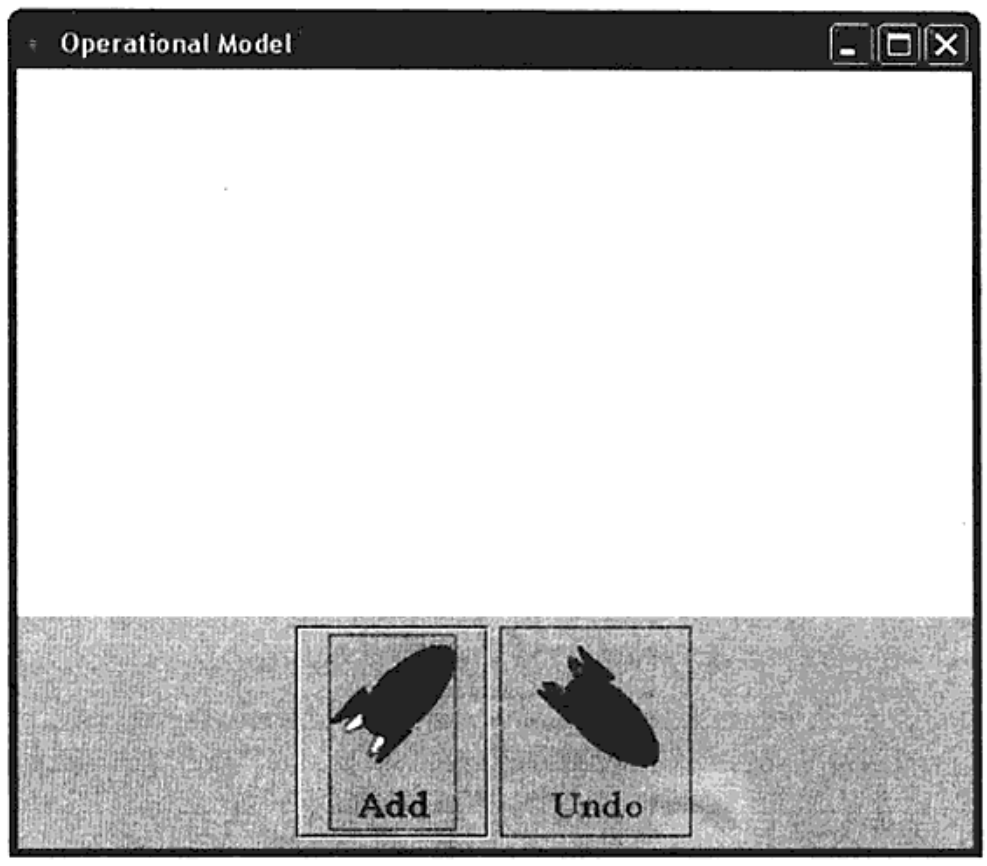

图 19.1　当可视化应用程序启动后，工作面板是空的，Undo 按钮处于禁用状态

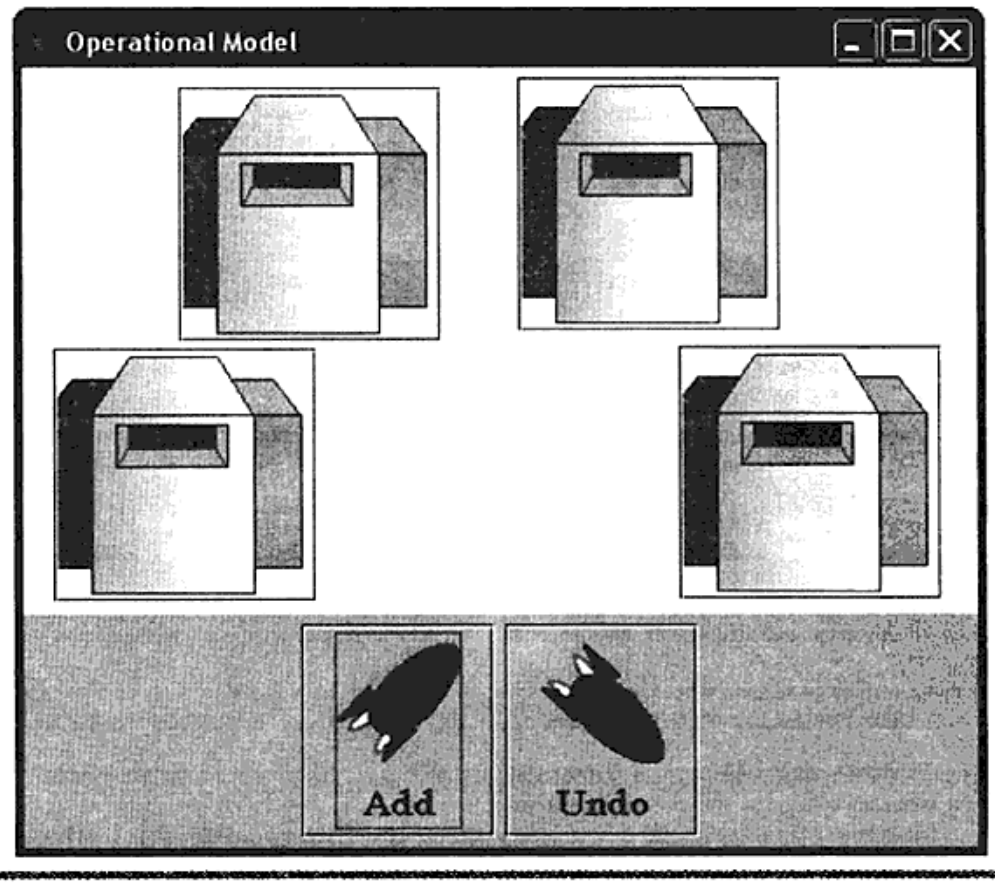

图 19.2　用户可以在可视化应用程序中添加和重新布置机器

在备忘录中，我们需要捕获的状态由用户放置的机器位置信息组成。可以使用栈来保存这些备忘录，每当用户单击一次 Undo 按钮就弹出一个备忘录。

- 每当用户在可视化应用程序中添加或移动机器时，应用程序就为这个仿真工厂创建一个备忘录，并把它放入一个栈中。
- 每当用户单击 Undo 按钮时，应用程序就将栈顶部的备忘录弹出，然后将仿真对象恢复为该备忘录所记录的状态。

当可视化应用程序启动后，程序会向空栈压入一个初始的空备忘录，而这个备忘录永远也不会被弹出，以确保栈顶总有一个有效的备忘录。一旦栈仅包含一个备忘录，Undo 按钮就会被禁用。

可以在单个类中编写这些代码，但是，我们希望能够添加一些可以支持操作建模或其他用户期望用到的功能。如果仍然在一个类中集中这些职责，就会导致最终的应用程序变得非常庞大。因此，使用 MVC 设计是比较明智的选择。图 19.3 说明了如何将工厂建模功能转移到单独的类中。

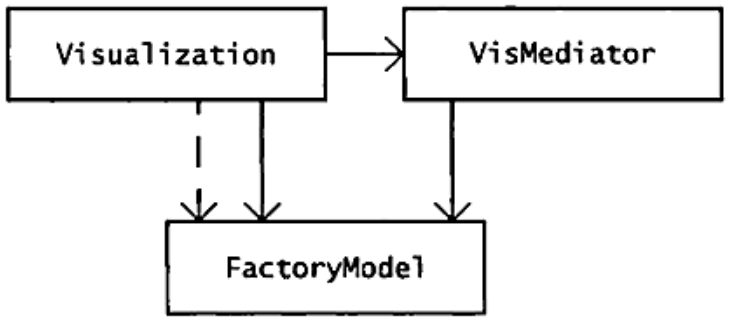

图 19.3　这个设计把应用程序的功能分到单独的类中，包括建模工厂、提供 GUI 元素以及处理用户的单击操作

这样的设计思路首先要求集中开发一个 FactoryModel 类，它没有任何 GUI 控件，和 GUI 类也没有任何依赖关系。

FactoryModel 类是我们设计的核心。它负责维护机器的当前配置信息以及之前配置的备忘录信息。

每当客户端代码要求工厂添加或移动一个机器时，工厂都会创建一个副本——一个备忘录，用来记录当前机器的位置信息并将其压入备忘录栈中。在本例中，我们并不需要特殊的备忘录类。每个备忘录仅仅是位置的列表：特定时间的机器位置列表。

工厂模型必须提供事件用来支持客户对工厂状态变化的关注。这就使得可视化应用程序 GUI 能够将用户所做的修改通知给模型。假设我们希望工厂允许客户注册添加机器和拖动机器等事件。图 19.4 所示的是 FactoryModel 类的设计说明。

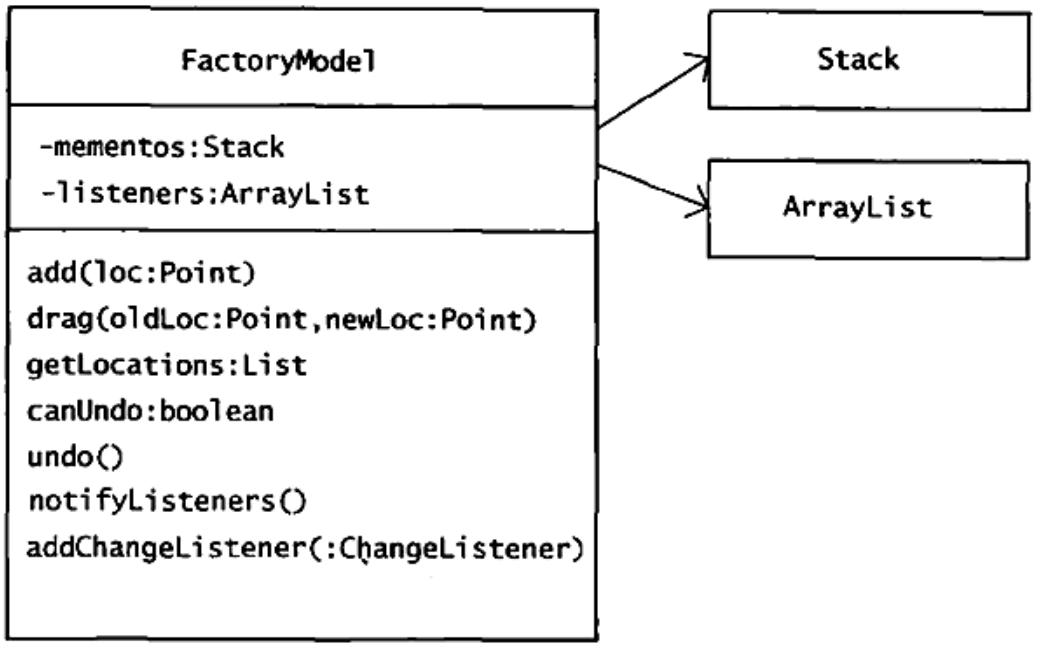

图 19.4 FactoryModel 类维护一个存储工厂配置信息的栈，并允许客户记录工厂的变更信息

图 19.4 的设计思路要求 FactoryModel 类为客户提供注册多个事件的能力。

比如，考虑添加机器事件。任何已注册的 ChangeListener 对象都会接收到这个变更操作。

```
package com.oozinoz.visualization;
// ...
public class FactoryModel {
    private Stack mementos;

    private ArrayList listeners = new ArrayList();

    public FactoryModel() {
        mementos = new Stack();
        mementos.push(new ArrayList());
    }
    //...
}
```

一开始，构造函数将工厂的初始配置设置为一个空列表。类中的其他方法维护机器配置备忘录的栈，并在改变发生时触发相应的事件。例如，要给当前配置添加一个机器，客户可以调

用下面的方法：

```
publit void add(Point location) {
    List oldLocs = (LiSt) mementos.peek();
    List newLocs = new ArrayList(oldLocs);
    newLocs.add(0, location);
    mementos.push(newLocs);
    notifyListeners();
}
```

这段代码创建了一个机器位置的新列表，并将其压入工厂模型所维护的备忘录栈中。略有不同的是，这段代码确保新机器位于列表的第一位。新机器（位置信息）应该出现在其他机器（信息）的前面，这是因为工厂平面图中的机器位置可能重合。

在接收到工厂模型的事件时，注册了变化通知的客户可以通过完全重绘来更新工厂模型的视图。getLocations()方法始终会提供工厂模型的最新配置，代码如下所示：

```
public List getLocations() {
    return (List) mementos.peek():
}
```

FactoryModel 类的 undo()方法允许客户把机器位置模型恢复为以前的版本。执行这些代码的时候，它也会调用 notifyListeners()方法。

挑战 19.1

实现 FactoryModel 类的 undo()方法。

答案参见第 343 页

通过注册为监听者，可提供重绘客户的工厂视图的方法，这样，感兴趣的客户可以提供撤销操作。Visualization 类就是这样的一个客户。

图 19.3 所示的 MVC 设计思路使得解释用户动作的任务和维护 GUI 的任务相分离。Visualization 类创建 GUI 控件，但是不处理传递给中间者的 GUI 事件。VisMediator 类把 GUI 事件解释为工厂模型中相应的变化。当模型发生改变时，GUI 可能需要相应的更新。Visualization 类会注册 FactoryModel 类所提供的通知事件。注意它们的职责划分：

- 可视化应用程序（Visualization 对象）把工厂事件转换为 GUI 的变化。
- 调停者（VisMediator 对象）把 GUI 事件转换为工厂变化。

图 19.5 说明了这三个类更详细的协作关系。

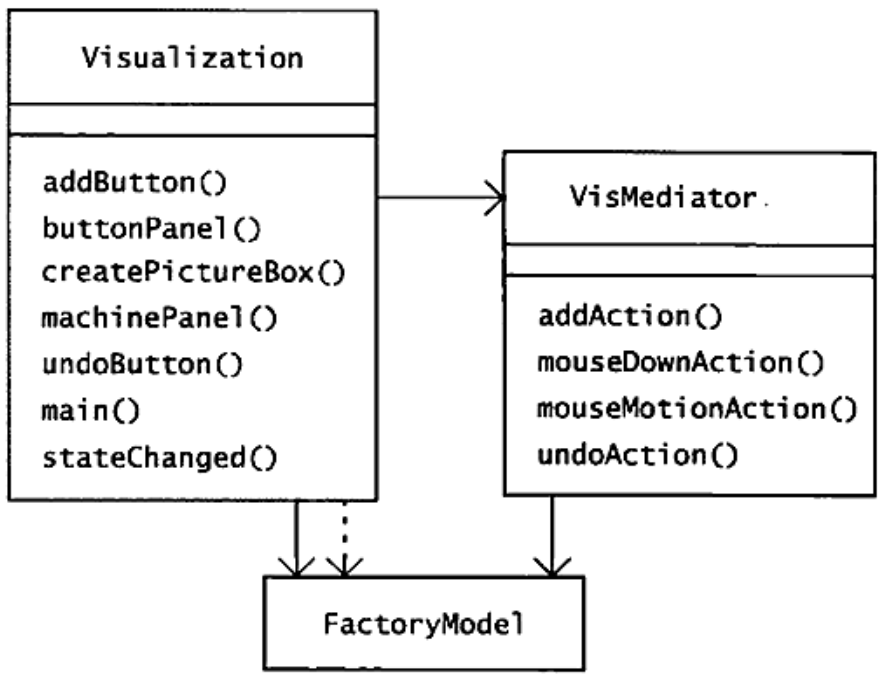

图 19.5 VisMediator 类将 GUI 事件解释成工厂模型的变化，Visualization 类处理工厂事件以更新 GUI

假定在拖曳一张机器照片时，用户碰巧将它放置在一个错误的位置，此时可以单击 Undo 按钮。为了能够处理这个单击任务，Visualization 类注册了一个调停者对象，用以通知按钮事件。Visualization 类中的 Undo 按钮的代码如下所示：

```
protected JButton undoButton() {
    if (undoButton == null) {
        undoButton = ui.createButtonCancel();
        undoButton.setText("Undo");
        undoButton.setEnabled(false);
        undoButton.addActionListener(mediator.undoAction());
    }
    return undoButton;
}
```

这段代码把处理单击操作的任务交给了调停者。调停者再通知请求变化的工厂模型。使用如下代码将执行撤销请求，以恢复工厂模型的变化：

```
private void undo(ActionEvent e) {
```

```
        factoryModel.undo();
    }
```

方法中的 factoryModel 变量是 Visualization 类创建的 FactoryModel 的一个实例，同时，它会把工厂模型传入 VisMediator 类的构造函数。我们已经检查了 FactoryModel 类的 pop()命令。图 19.6 显示了当用户单击 Undo 按钮后消息的流动过程。

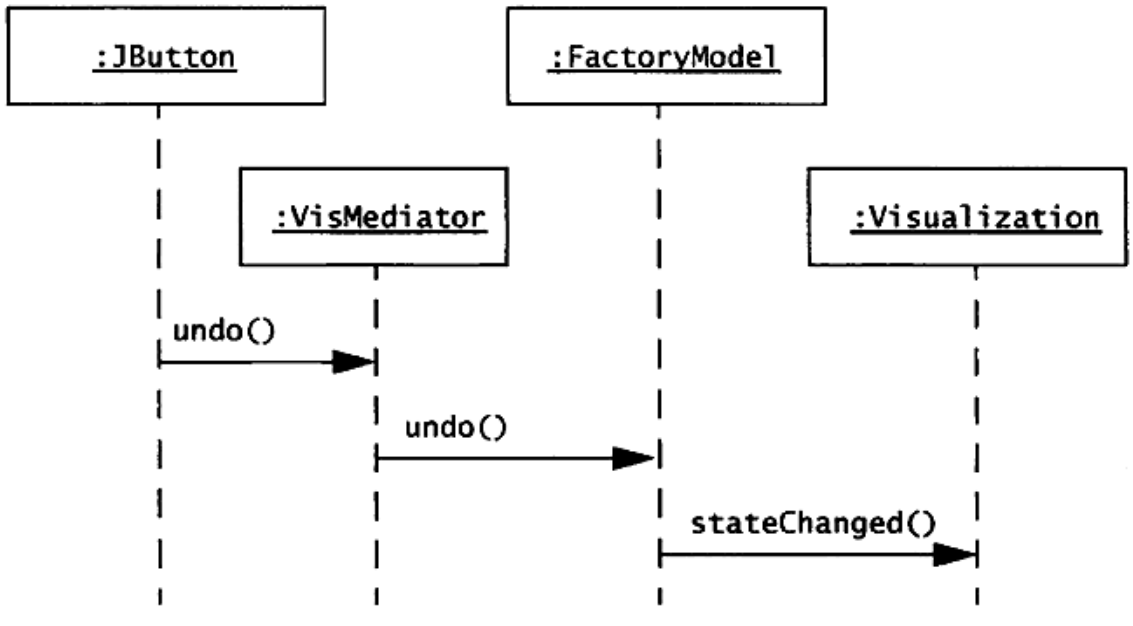

图 19.6　用户单击 Undo 按钮后消息的流动过程

当 FactoryModel 类弹出当前的配置信息时，之前存储为备忘录的配置会显露出来，undo()方法会通知所有的 ChangeListeners。Visualization 类在其构造函数中注册如下信息：

```
public Visualzation(UI ui) {
    super(new BorderLayout());
    this.ui = ui;
    mediator = new VisMediator(factoryModel);
    factoryModel.addChangeListener(this);
    add(machinePanel(), BorderLayout.CENTER);
    add(buttonPanel(), BorderLayout,SOUTH);
}
```

对于工厂模型中每台机器的位置，可视化应用程序都维护一个 Component 对象，该对象是在 createPictureBox()方法中建立起来的。stateChanged()方法必须从机器面板上清除所有的图片箱子控件，同时，在工厂模型的当前位置集合中重新添加新的图片箱子。如果工厂栈中只剩下唯一一条备忘录记录，stateChanged()方法必须禁用 Undo 按钮。

挑战 19.2

完成 Visualization 类中 stateChanged()方法的代码编写。

答案参见第 343 页

借助备忘录模式，你可以保存和恢复对象的状态。备忘录模式最常见的用法是在应用程序中提供撤销操作。在某些应用程序中，例如工厂可视化程序，你需要存储信息的仓库可能是一个对象。在某些情况下，你可能需要持久地保存备忘录。

备忘录的持久性

备忘录是一个用来存储对象状态的小型数据仓库。通常，你可以用另一个对象、字符串或者文件来创建备忘录。在设计备忘录的时候，应事先预测对象状态可能在备忘录中存储的时间，因为这个时间将影响我们的设计策略。这段时间可能是一瞬间、几小时、几天或者几年。

挑战 19.3

写出你决定将备忘录存储在文件中而不是对象中的两个理由。

答案参见第 344 页

跨会话的持久性备忘录

用户运行一个程序，执行程序中的事务，直到最后退出，整个过程称为一次会话。假设你的用户希望存储一次会话中的一个仿真（对象），然后在另一个会话中恢复该对象。这种能力通常叫做持久性存储。持久性存储能力和备忘录模式的意图是一致的，同时它还是已经实现的撤销操作的自然扩展。

假定由 Visualization 类派生出一个 Visualization2 类，该类具有一个菜单栏，包含了

File 菜单，其中提供了 Save As…和 Restore From …命令。

```
package com.oozinoz.visualization;

import javax.swing.*;
import com.oozinoz.ui.SwingFacade;
import com.oozinoz.ui.UI;

public class Visualization2 extends Visualization {
    public Visualization2(UI ui) {
        super(ui);
    }

    public JMenuBar menus() {
        JMenuBar menuBar = new JmenuBar();

        JMenu menu = new JMenu("File");
        MenuBar.add(menu);

        JMenuItem menuItem = new JMenuItem{"Save As..."};
        menuItem.addActionListener(mediator.saveAction());
        menu.add(menuItem);

        menuItem = new JMenuItem("Restore From...");
        menuItem.addActionListener(
            mediator.restoreAction());
        menu.add(menuItem);

        return menuBar;
    }

    public static voin main(String[] args) {
        Visualization2 panel = new Visualization2(UI.NORMAL);
        JFrame frame = SwingFacade.launch(
            panel,"Operational Model");
        frame.setJMenuBar(panel,menus());
        frame.setVisible(true);
    }
}
```

为了实现上述代码，VisMediator 类还需要添加 saveAction()和 restore Action()两个方法。当用户选择菜单项时，MenuItem 对象会促使这些动作被调用。运行 Visualization2 类的 GUI，如图 19.7 所示。

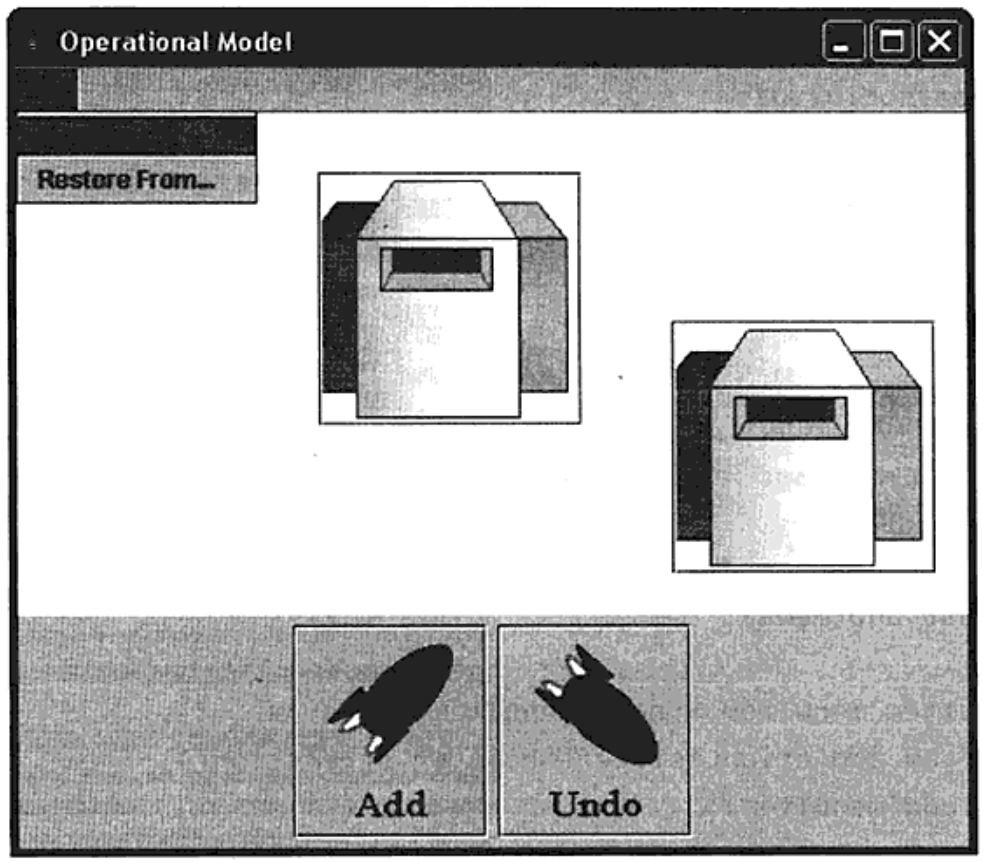

图 19.7 File 菜单的添加使用户可以保存一个备忘录，以便于应用程序的后期恢复

一个存储对象（比如工厂模型的配置）的简易方式是序列化。VisMediator 类的 saveAction()方法的代码如下所示：

```
public ActionListener saveAction() {
    return new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
               VisMediator.this.save((Component)e.getSource());
            } catch (Exception ex) {
               System.out.println (
                  "Failed save: " + ex.getMessage0());
        }
    }};
}
```

```
public void save(Component source) throws Exception {
    JFileChooser dialog = new JFileChooser();
    dialog.showSaveDialog(source);

    if (dialog.getSelected File() == null)
        return;

    FileOutputStream out = null;
    ObjectOutputStream s = null;
    try (
        out = new FileOutputStream(dialog.getSelectedFile());
        s = new ObjectOutputStream(out);
        s.writeObject(factoryModel.getLocations());
    } finally {
      if (s != null) s.close();
    }
}
```

挑战 19.4

请写出 VisMediator 类中 restoreAction()方法的代码。

答案参见第 344 页

在 *Design Patterns* 一书中，备忘录模式的意图被定义为：在不违反封装的前提下，捕获对象的内部状态，并在该对象之外保存这个状态，以便于对象的状态可以在将来被恢复。

挑战 19.5

在这种情况下，我们使用 Java 序列化机制将备忘录写到一个二进制文件中。假定已经将备忘录信息以 XML 格式（文本格式）保存起来了。请结合自己的理解，简单陈述把备忘录保存为文本格式是否会破坏封装。

答案参见第 345 页

当开发者希望借助于序列化或者 XML 文件来保存和恢复对象的数据时，你应该认真理解其中的含义。这就涉及设计模式中的重点：通过使用公共词汇，我们才能更好地讨论设计模式

概念及其应用。

小结

借助备忘录模式，可以捕获对象的状态，以便于将来把对象恢复为以前的状态。具体采用哪种方法来存储对象状态，取决于对象状态需保存时间的长短。恢复一个对象，有时候只在敲击几下鼠标和键盘之后，但也可能要经过几天或者几年。在应用程序会话期间保存和恢复对象，最常见的理由是支持撤销操作。在这种情况下，我们可以将对象的状态存储在另一个对象中。为了支持对象跨多个会话的持久性存储，可以使用对象序列化或其他方式来保存备忘录。

第 4 部分

操作型模式