第17章

抽象工厂（Abstract Factory）模式

在创建对象时，有时会指定具体的类去实例化一个对象。可以使用工厂方法模式来定义一个外部方法以决定实例化哪个类。但有时候，控制实例化哪个类的因素可能与很多类息息相关。

抽象工厂模式又名工具箱，其意图是允许创建一族相关或相互依赖的对象。

经典范例：图形用户界面工具箱

GUI 工具箱是抽象工厂模式的一个典型范例。一个 GUI 工具箱就是一个对象，也是一个抽象工厂，它为用户提供各种 GUI 控件，并为控件提供一些基本的用户界面。它可以决定一些控件，例如按钮、文本区域等的实现方式，还可以设置具体的界面风格，比如背景颜色、外观以及控件家族中一些 GUI 设计的其他方面。当然，你可以为整个系统建立一个统一的界面，但是，也可以使用不同的界面风格反映应用程序版本的改变，体现公司标准图像的变化，甚至一些日常的简单改进。抽象工厂实现界面风格的多样化，使得应用程序更加容易理解和使用。图 17.1 所示的 Oozinoz 公司的 UI 类设计思路是一个很好的例子。

UI 类的子类可以重写用户控件工厂中的任何元素。GUI 对象是 UI 类的一个实例，创建该实例的应用程序可以通过在 UI 类的子类中创建不同的界面风格。例如，Oozinoz 公司使用 `visualization` 类去帮助工程师布局一个新的设备生产线。该可视化界面如图 17.2 所示。

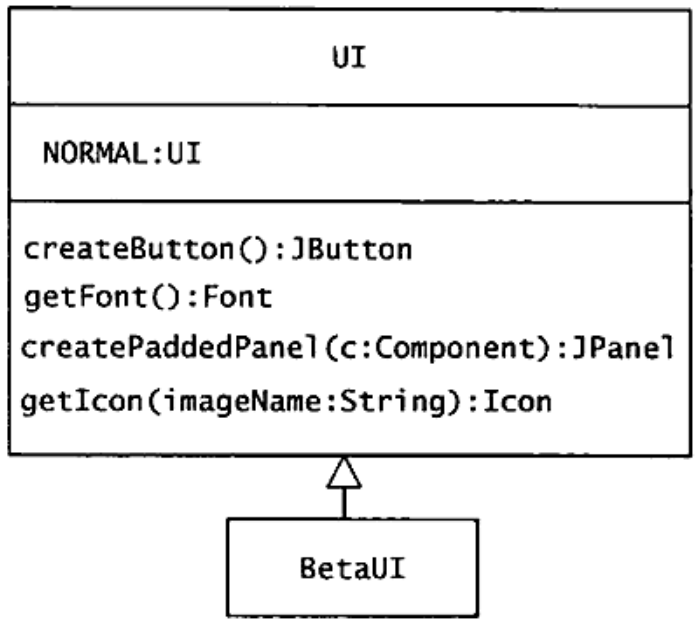

图 17.1　UI 类和 BetaUI 类的实例化对象是创建 GUI 控件族的抽象工厂

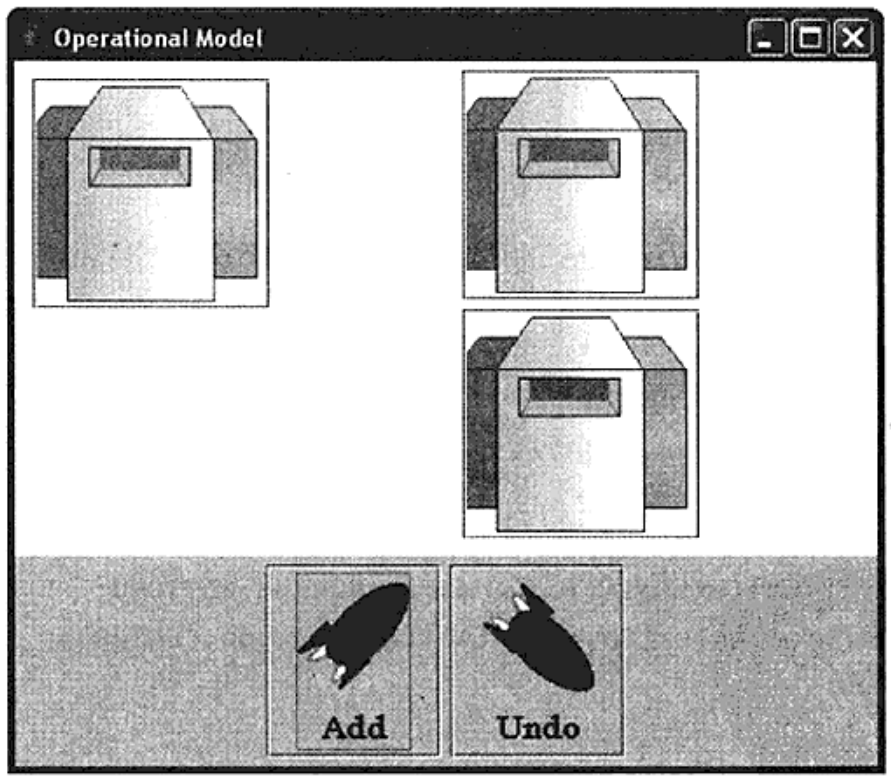

图 17.2　用户可以单击 Add 按钮，在面板的左上区域添加机器，并把机器拖曳至合适位置。单击 Undo 按钮可撤销添加或拖曳操作

图 17.2 的可视化界面告诉我们用户可以在工厂区域任意添加或拖曳机器（参见 app.abstractFactory 目录中的 ShowVisualization 程序）。Visualization 类在构造函数内接收一个 UI 对象来获取按钮。图 17.3 显示了 Visualization 类。

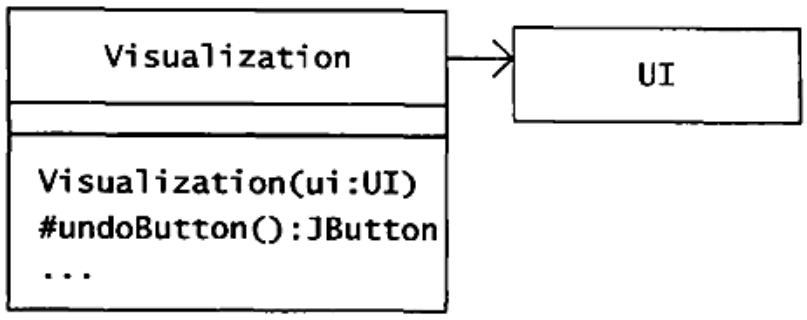

图 17.3 Visualization 类使用一个 UI 工厂对象创建 GUI

Visualization 类使用 UI 对象创建它的 GUI。下面是 undoButton()方法的代码示例。

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

这段代码用于创建一个名称为 Undo 的取消按钮。UI 类会决定按钮上的图片与大小。代码如下所示：

```
public JButton createButton() {
    JButton button = new JButton();
    button.setSize(128, 128);
    button.setFont(getFont());
    buttom.setVerticalTextPosition(AbstractButton.BOTTOM);
    button.setHorizontalTextPosition(AbstractButton.CENTER);
    return button;
}

public JButton createButtonOk() {
    JButton button = createButton();
    button.setIcon(getIcon("images/rocket-large.gif"));
    button.setText("Ok!");
    return button;
}
```

```
public JButton createButtonCancel() {
    JButton butto = createButton();
    button.setIcon(getIcon("images/rocket-large-down.gif"));
    button.setText("Cancel!");
    return button;
}
```

为了给可视化提供一个与众不同的界面风格，我们创建了一个子类，它重写 UI 工厂类中的一些元素，并且可以传递该 GUI 工厂的实例到 Visualization 类的构造函数中。

假定我们发行了一个增添了新特性的 Visualization 类的新版本，同时，相关代码正在进行 beta 测试。此时，我们希望改变一些用户接口，例如把字体改成斜体并且用 cherry-large.gif 和 cherry-large-down.gif 两张图片来替换火箭图片。BetaUI 类代码的主要实现如下：

```
public class BetaUI extends UI {
    public BetaUI () {
        Font oldFont = getFont();
        font = new Font(
            oldFont.getName(),
            oldFont.getStyle() | Font.ITALIC,
            oldFont.getSize());
    }

    public JButton createSuttonOk() {
        // 挑战!
    }

    public JButton createButtonCancel() {
        // 挑战!
    )
```

挑战 17.1

完成 BetaUI 类的代码。

答案参见第 338 页

运行如下代码可以获得新的界面效果：

```
package app.abstractFactory;
```

```
// ...
public class ShowBetaVisualization {
    public static void main(String[] args) {
        JPanel panel = new Visualization(new BetaUI());
        SwingFacade.launch(panel, "Operational Model");
    }
}
```

Visualization 类的程序运行界面如图 17.4 所示。UI 和 BetaUI 类的实例为 GUI 应用程序提供了不同风格的 GUI 控件。尽管这是抽象工厂模式的一个有效应用，但有时候这样的设计还是显得不够健壮。特别是 BetaUI 类将依赖这种重写创建方法并访问受保护的实例变量的能力，尤其是 UI 类中的 font 变量。

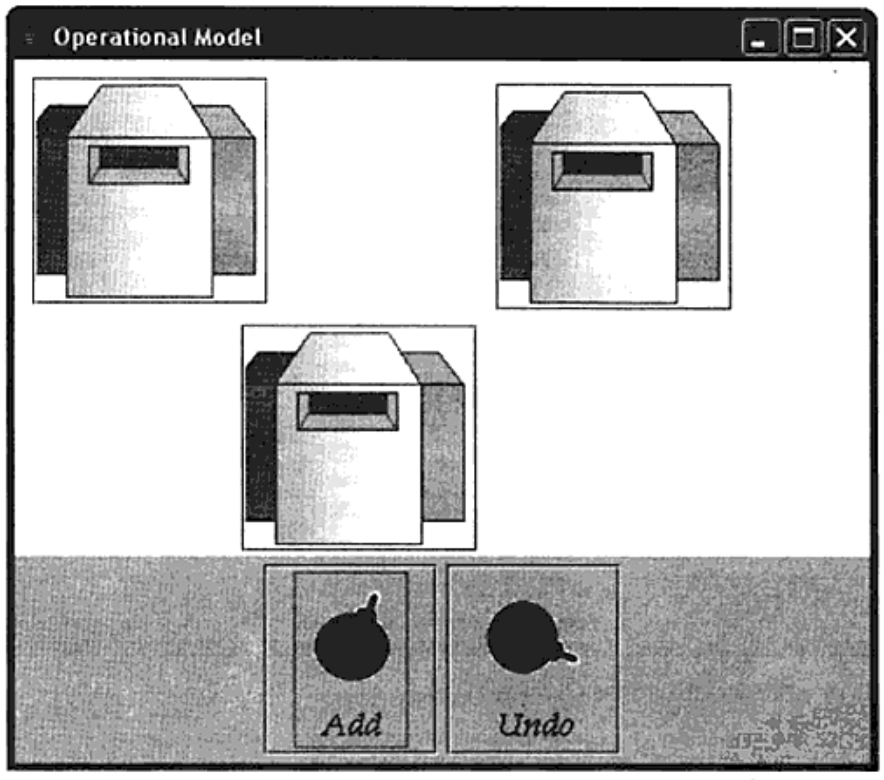

图 17.4　在不修改 Visualization 类的任何代码的前提下，BetaUI 类呈现了新的界面风格

挑战 17.2

对设计进行修改，使得 GUI 控件工厂仍然允许支持多种扩展，但需要减少子类对 UI 类方法修饰符的依赖。

答案参见第 338 页

抽象工厂模式使得客户端代码在需要新对象的时候，无须关心该对象是由哪个类实例化的。就这点而言，抽象工厂就像是工厂方法的集合。在某些情况下，工厂方法模式的设计可能会迈向抽象工厂模式的设计。

抽象工厂和工厂方法

在第 16 章中，介绍了实现 CreditCheck 接口的两个类。当客户端代码调用 CreditCheckFactory 类的 createCreditCheck()方法时，该工厂会实例化其中的一个类。工厂具体实例化哪一个类，取决于信贷机构是否已经建立和营业。这个设计使得其他开发者无须关心信贷机构的状态。图 17.5 是 CreditCheckFactory 类和 CreditCheck 接口的当前实现。

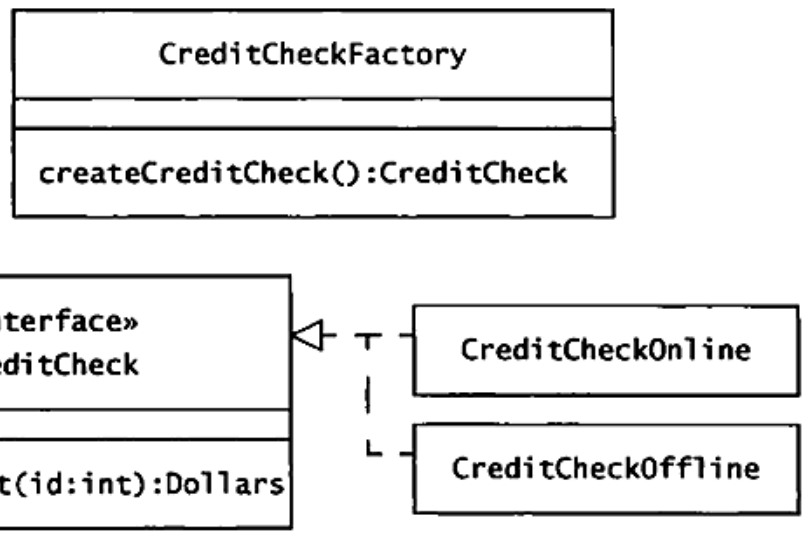

图 17.5　在一个工厂方法模式的设计中，客户端代码无须知道信贷账单是由哪个类实例化的

通常情况下，CreditCheckFactory 类为信贷机构提供客户的信贷信息。除此之外，credit 包还包括可以帮客户查询交易额和账单记录的类。图 17.6 是 com.oozinoz.credit 包的设计。

现在，假定需求分析者告诉你 Oozinoz 公司打算在加拿大开展客户的服务业务。为了在加拿大做业务，不得不使用一个不同的信贷机构和数据源去处理发货交易和账单记录的相关信息。

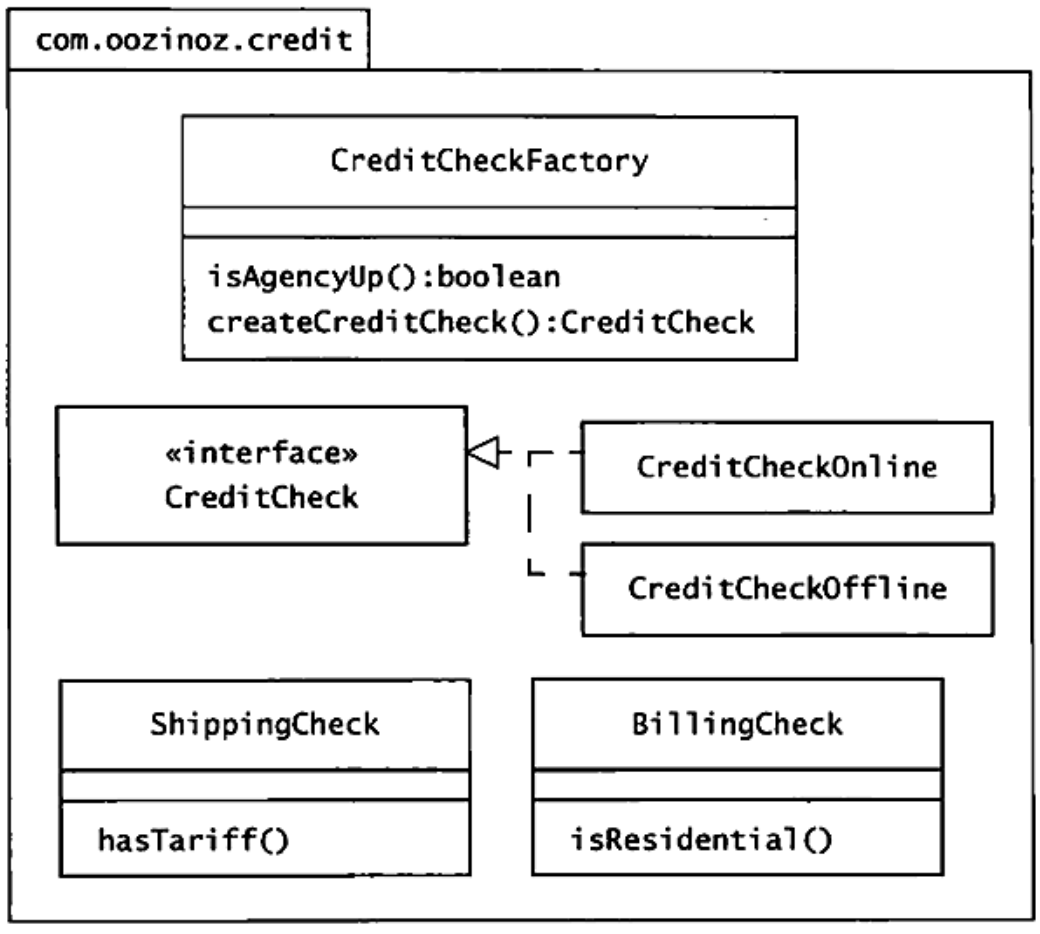

图 17.6　com.oozinoz.credit 包中的类负责核对顾客的信贷信息、发货地址和账单地址

当顾客发出查询请求时，呼叫中心应用程序需要一系列的对象去执行不同的信息核对。使用哪些对象取决于查询呼叫的所在地是加拿大还是美国。你可以运用抽象工厂模式去创建这些对象组。

业务扩展到加拿大势必要使得关于信贷查询的类数量倍增。假定你已经决定将这些类分别放置在三个不同的包中。credit 包将包含三个“check”接口和一个抽象工厂类。这个类有三个创建方法，为查询信贷信息、交易额和账单信息提供合适的对象。同时，如果你不介意呼叫的所在地使用 CreditCheckOffline 类进行离线查询，也可以将该类放到这个包中。图 17.7 是加入 com.oozinoz.credit 包中的一些新内容。

为了在 credit 包中使用具体类实现接口，可以引入两个新的包：com.oozinoz.credit.ca 和 com.oozinoz.credit.us。每个包都包含了一个工厂类和实现了 credit 包中每个接口的若干类。

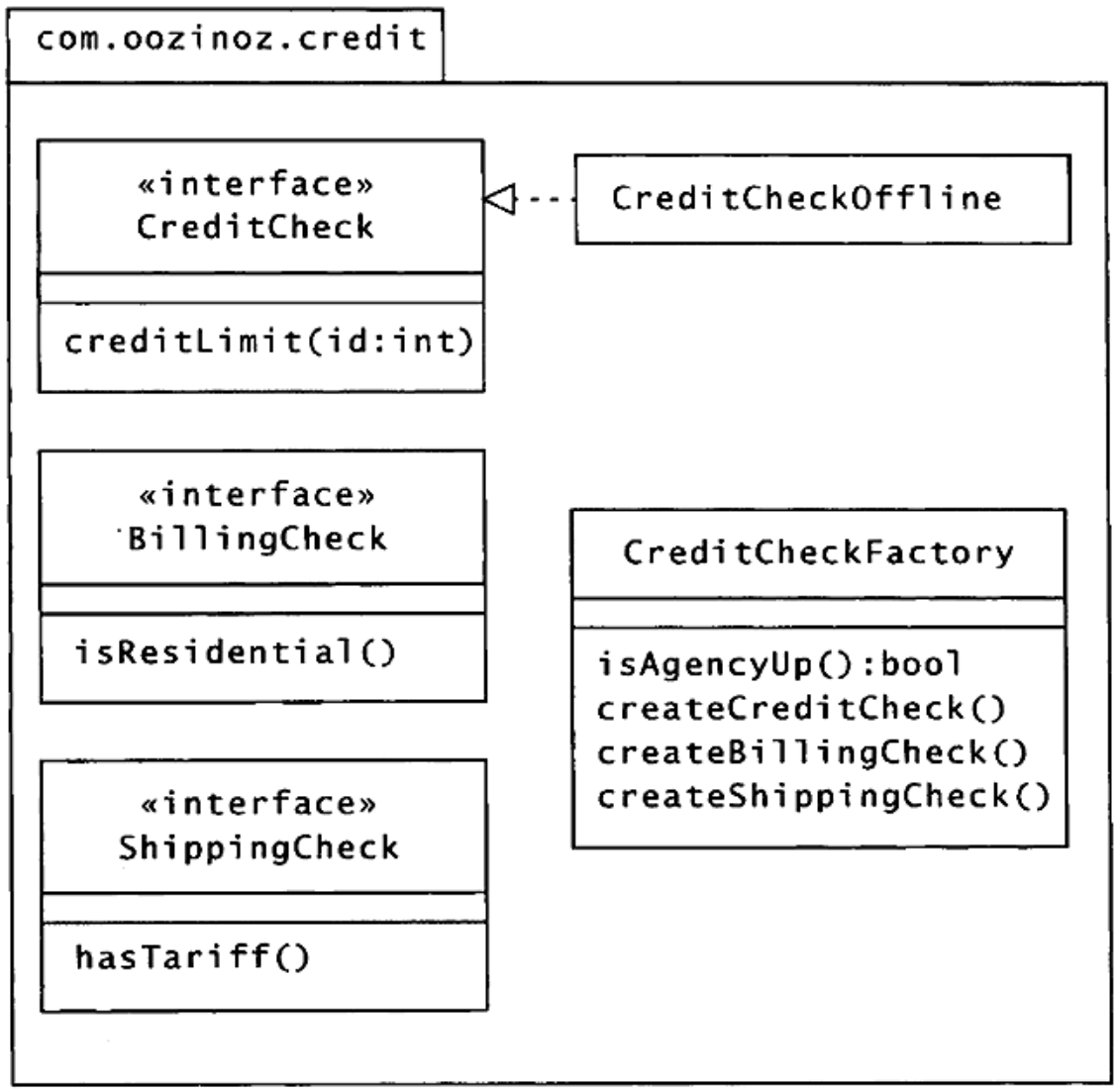

图 17.7　修订过的 com.oozinoz.credit 包所包含的主要接口和抽象工厂类

挑战 17.3

图 17.8 显示了 com.oozinoz.credit.ca 包中的类以及它们与 credit 包中类和接口的关系。请完成该图。

答案参见第 339 页

加拿大和美国信贷查询的具体工厂类非常简单。它们返回加拿大或美国版本的“check”接口，当本地信贷机构处于离线时，情况将变得特殊。此时，所有的具体工厂将返回 CreditCheckOffline 对象。正如前一章中所述，CreditCheckFactory 类中也有一个 isAgencyUp()方法来告知信贷机构是否已经可以访问。

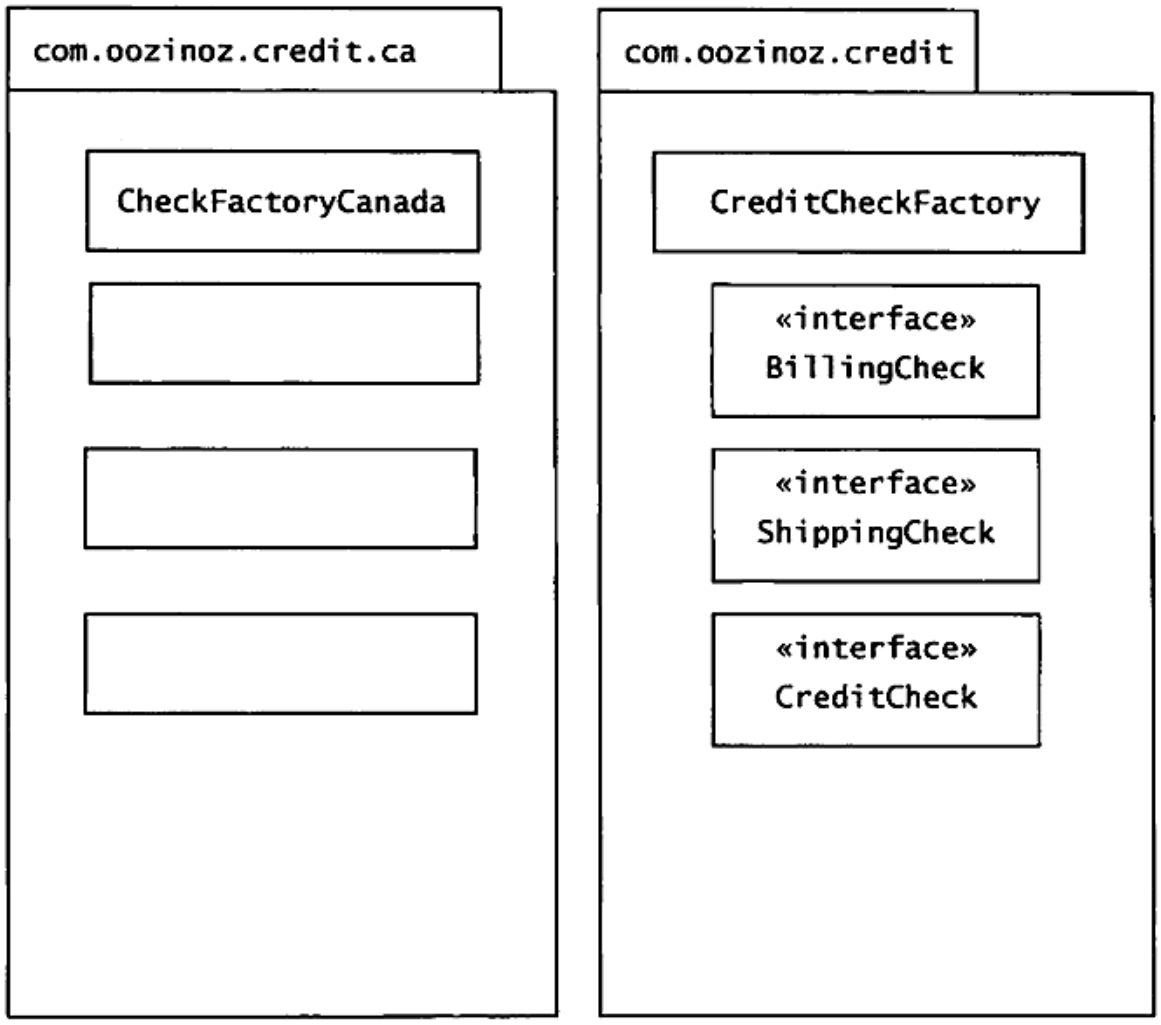

图 17.8 该图展示了 com.oozinoz.credit.ca 包中的类以及它们与 com.oozinoz.credit 包中的类的关系

挑战 17.4

完成 CheckFactoryCanada.java 的代码。

```
package com.oozinoz.credit.ca;

import com.oozinoz.credit.*;

public class CheckFactoryCanada extends CrdeitCheckFactory{
    // 挑战!
}
```

答案参见第 340 页

此时，可以应用抽象工厂模式来创建完成不同顾客信息核对的对象系列。CreditCheckFactory 抽象类的实例可以是 CheckFactoryCanada 类或 CheckFactoryUS 类。这些对象都是抽象工厂，可以创建账单、交易量，同时信贷查询对象完全匹配这个工厂对象所代表的国家。

包和抽象工厂

粗略地讲，一个包通常包含一系列的类，一个抽象工厂可以创建一系列的对象。在之前的例子中，使用两个不相关的包实现了加拿大和美国的抽象工厂，还用另一个包来提供工厂所产生的包的公共接口。

挑战 17.5

给出充分理由说明应该将每个工厂及其相关类放在相互独立的包中，或者提出一个更好的解决方案。

答案参见第 340 页

小结

可以使用抽象工厂模式为客户端代码做一些代码准备，比如，创建相互关联或相互依赖的系列对象中的部分对象。这个模式的一个经典应用是界面风格系列——GUI 控件（工具箱）。也有部分主题并不包含在这个对象族中，例如客户的所在国家决定不同的界面风格。如同工厂方法模式，通过运用抽象工厂模式，客户端代码无须知道哪个类被实例化。抽象工厂给客户端提供了工厂类，每个工厂类都可以创建同一主题下的相关对象。