设计模式案例分析

以JHotDraw 为例

目录

- 设计模式和框架的理论
- Jhotdraw简介
- Jhotdraw的设计模式和框架应用
  - MVC模式
  - Composite设计模式
  - Strategy设计模式
  - FactoryMethod 设计模式

设计模式和框架的理论

为什么需要设计模式和框架

- 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，但是他们可能都没有意识到。

设计模式和框架

- 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。

- 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。

应用框架的好处

- 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。

应用框架的不足

- 你必须要学习和理解如何和应用框架进行交互，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。

- 甚至你还要学习应用框架的缺点。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。

JHOTDRAW简介

JHOTDRAW简介

- JHotDraw是一个基于LGPL协议（GNU 宽通用公共许可证的缩写形式）的二维的GUI框架开源项目，是一个设计良好的(Well-Designed)框架，主要用于支持用Java开发的图形编辑器，已用于各种研究。

- JHotDraw实际上是HotDraw的Java版，HotDraw也是一个图形框架，最初是由Kent Beck和Ward Cunningham用Smalltalk开发**为教学的目的**而设计的。

JHOTDRAW的程序包组织

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。
- CH.ifa.draw.standard里找到这些接口的标准实现。
- CH.ifa.draw.figures 和 CH.ifa.draw.contrib 里是一些附加的功能。
- 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的主框架

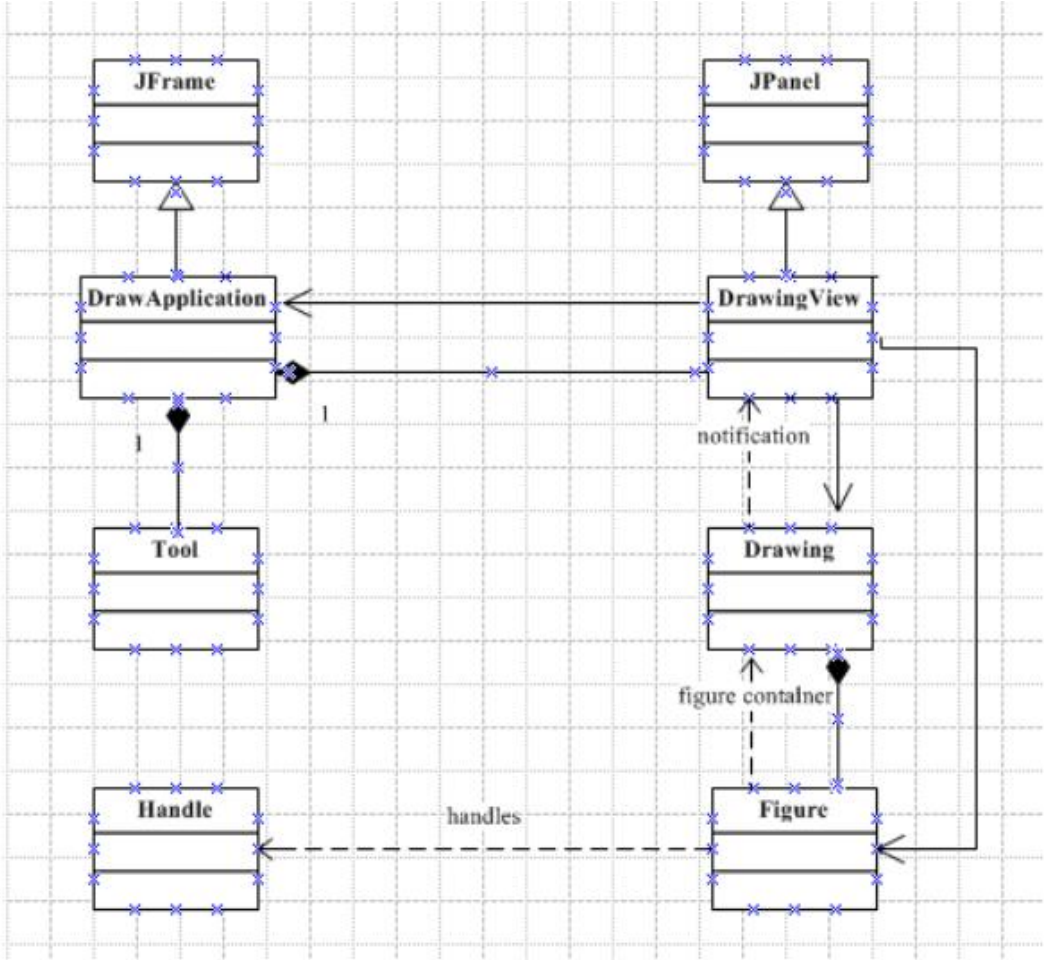


图 1 JHotDraw主框架

JHOTDRAW的主框架

- DrawApplication 定义了绘图的界面，并作为各个组件交互的中介者
- DrawingView 是一个显示绘图的区域，它也可用来接受客户的输入
- Drawing 容纳各种图元(Figure)，是图元的容器。Drawing 的改变被传递到DrawingView，并负责更新图形。
- 每个图元都有句柄(Handle)，句柄用来定义访问点以及如何与图元交互。
- 在DrawingView 中，可以选择几个图元并对其操作。
- 工具栏里有各种工具(Too1)用来生成图元或对图元进行操作。

JHOTDRAW典型开发过程

- 为开发的应用程序创建自己的图元（figure）。
  - 针对特定的应用程序创建对应的图元
  - 已有预定义的图元，例如AbstactFigure、compositeFigure、AttributeFigure等，可以通过继承并重载一些方法来重新定义它们的行为和表现形式
- 按照应用的需求开发图形工具。
  - JHotDraw本身已提供了一些工具，例如创建工具CreationTool、联接工具ConnectionTool、选择工具SelectionTool以及文本工具TextTool
  - 同时也可以定义程序的交互过程
- 生成GUI并整合进应用程序中。
  - JHotDraw已经有了一个基本的应用程序框架；如，DrawApplication、MDI_DrawApplication、DrawApplet。
- 使用javac编译你的应用程序。

JHOTDRAW典型开发过程

任何利用JHotDraw的应用都是用一个窗口用来绘图，这个窗口就是GUI编辑器窗口，它是JFrame的子类，包含一个或多个内部框架(Internal Frames)，每个都与一个绘画视图(Drawing View)相关。DrawingView 是Jpanel 的子类，是一个显示绘图的区域。

JHOTDRAW
设计模式和框架应用

MVC框架介绍

- MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范。

- 用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

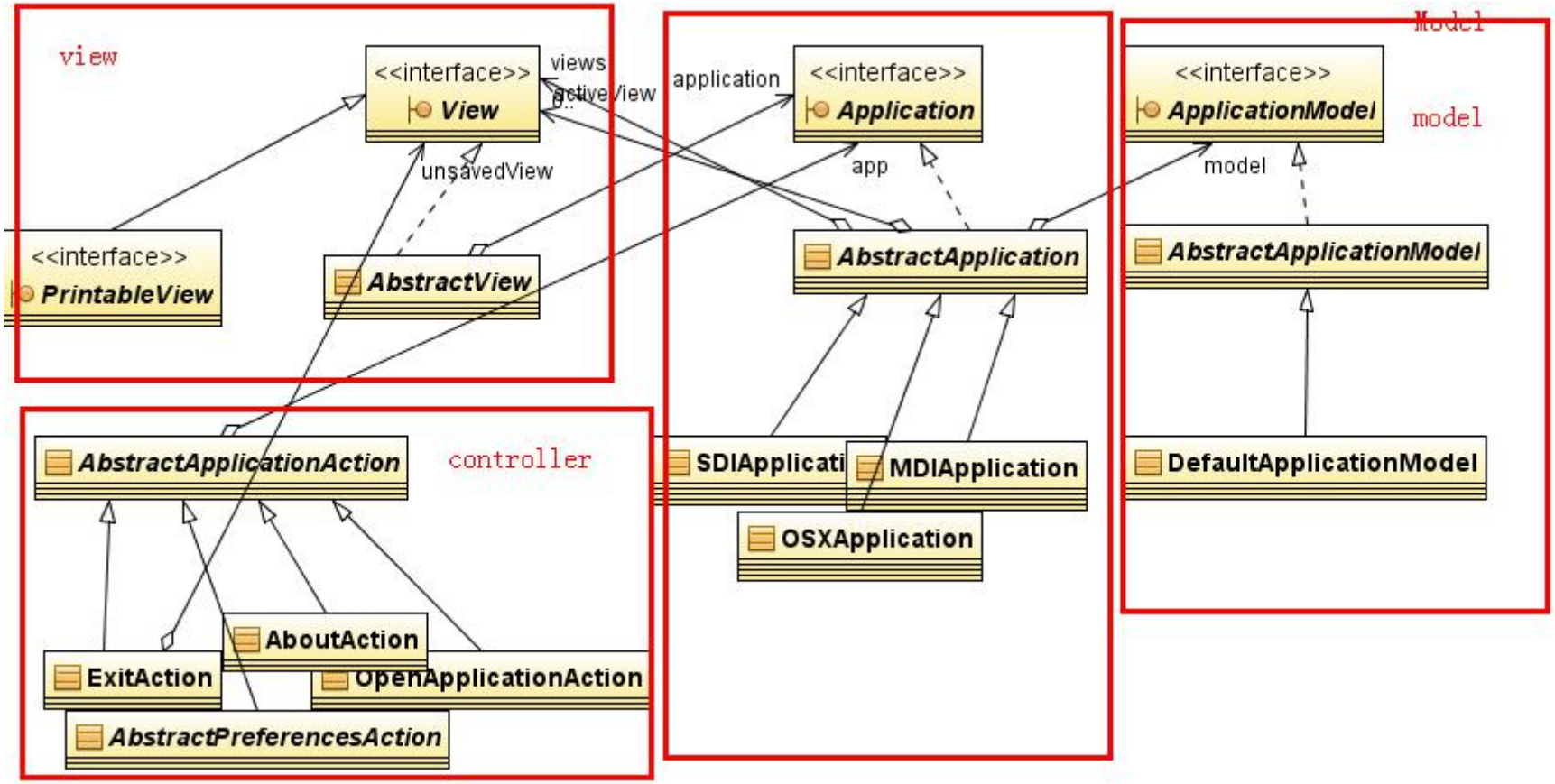
MVC框架介绍

- Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。

- View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。

- Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

JHOTDRWA的MVC框架



JHOTDRWA的MVC框架

- JHotDraw框架正是构建在MVC之上
- JHotDraw应用框架的api都位于包org.jhotdraw.app，app包定义了Application，ApplicationModel，View等接口
- Application封装了控制应用程序生命周期的方法
  1. init() 初始化程序
  2. launch() 启动程序
  3. start() 开始运行
  4. stop() 停止运行
  5. destroy() 退出程序

基于JHotDraw MVC框架构建应用程序

- 定义一个View继承AbstractView

```
import java.io.IOException;
import java.net.URI;

import org.jhotdraw.app.AbstractView;
import org.jhotdraw.gui.URIChooser;

public class HelloView extends AbstractView{
    public void clear() {
    }

    public void read(URI uri, URIChooser chooser) throws IOException {
    }

    public void write(URI uri, URIChooser chooser) throws IOException {
    }
}
```

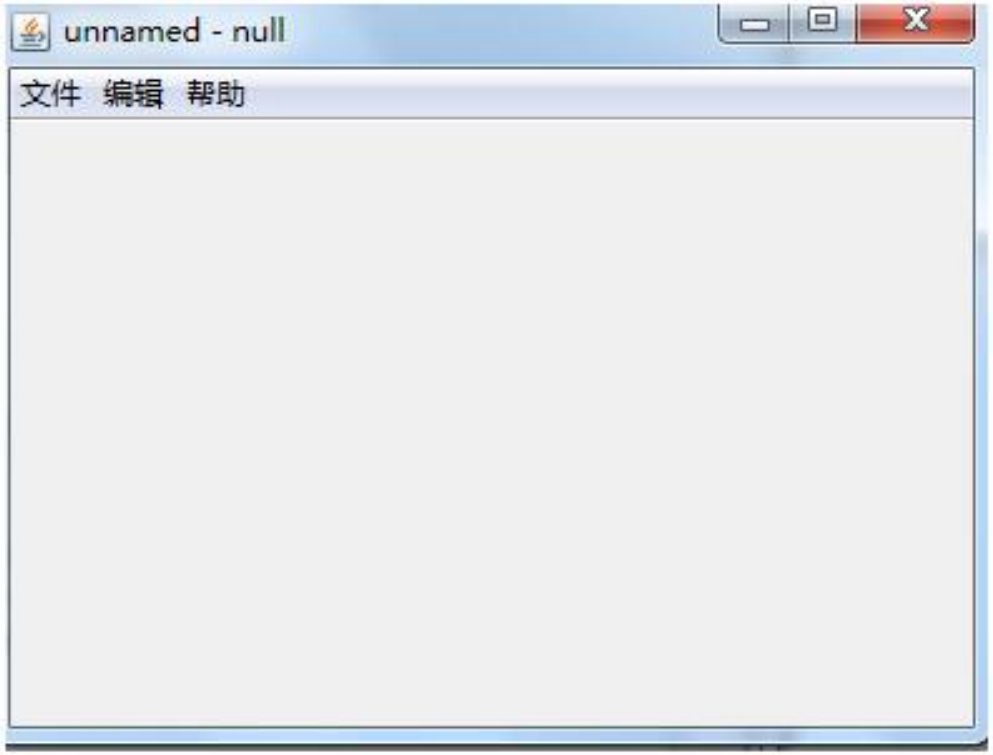
基于JHotDraw MVC框架构建应用程序

- 创建Application，并启动它

```
import org.jhotdraw.app.Application;
import org.jhotdraw.app.DefaultApplicationModel;
import org.jhotdraw.app.SDIApplication;

public class HelloApplication {

public static void main(String[] args) {
        Application app=new SDIApplication();
        DefaultApplicationModel model=new DefaultApplicationModel();
        model.setViewClass(HelloView.class);
        app.setModel(model);
        app.launch(args);
    }
}
```

基于JHotDraw MVC框架构建应用程序

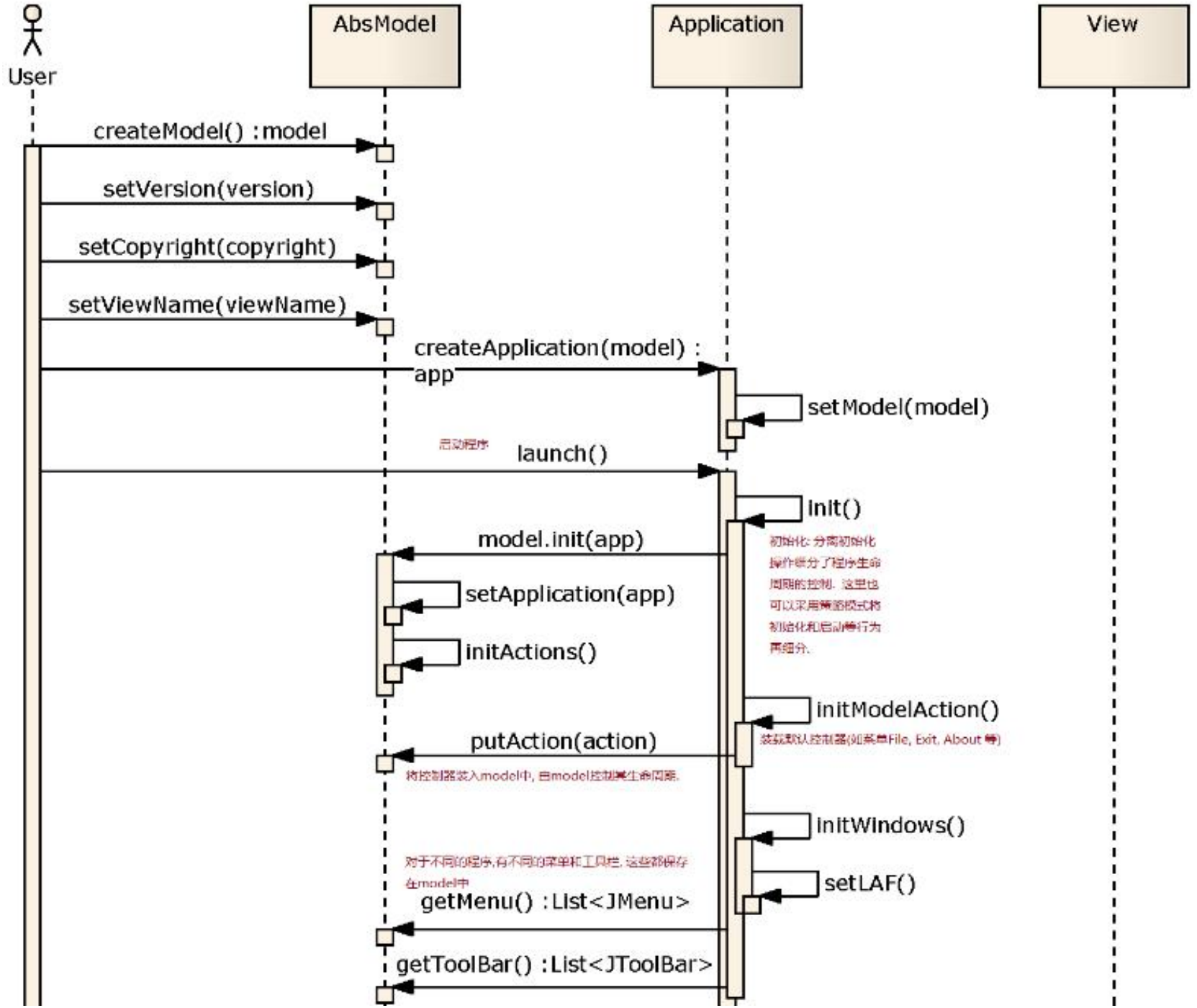
- 运行效果



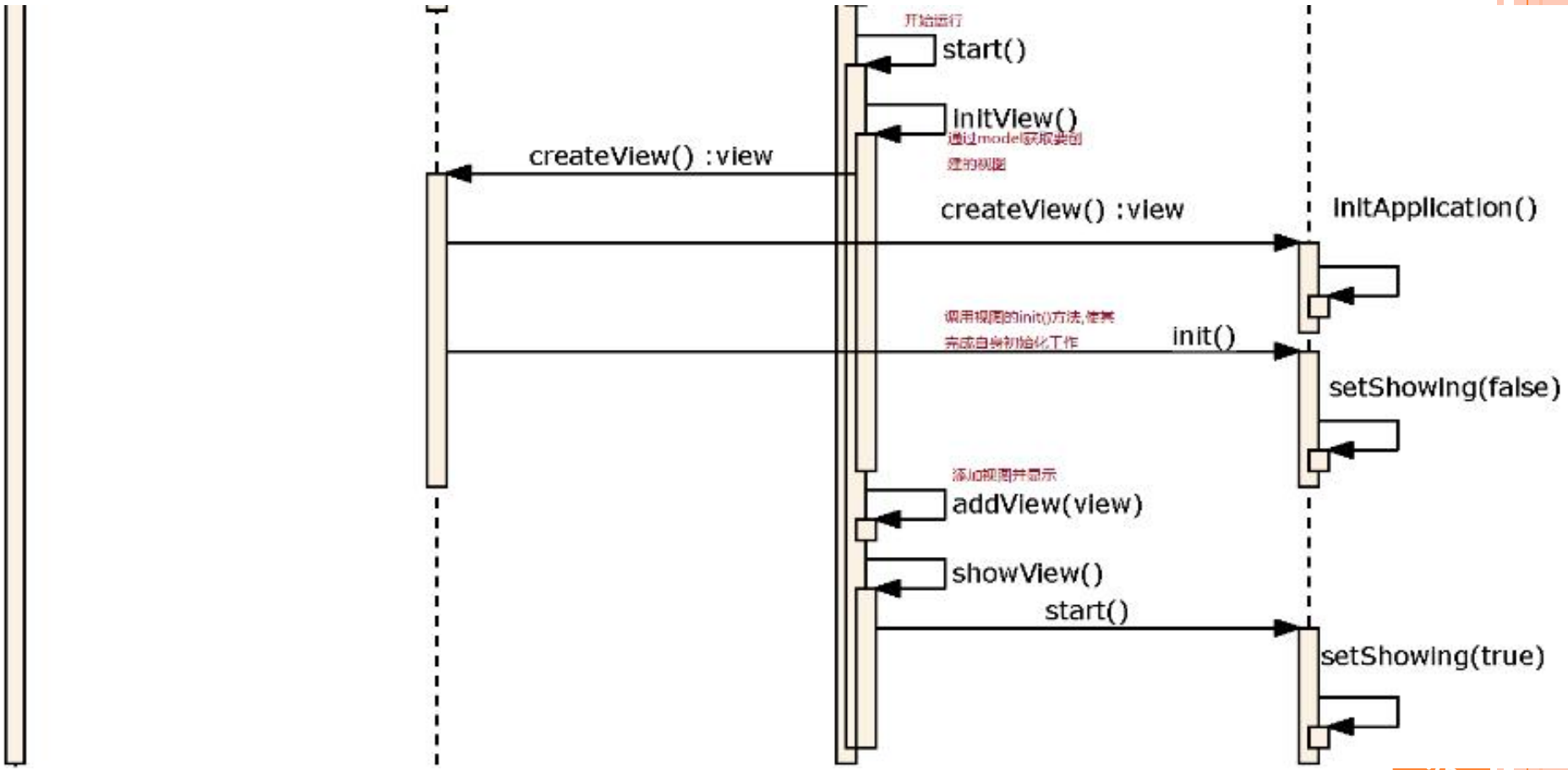
示例程序的运行效果

```
import org.jhotdraw.app.Application;
import org.jhotdraw.app.DefaultApplicationModel;
import org.jhotdraw.app.SDIApplication;

public class HelloApplication {

public static void main(String[] args) {
        Application app=new SDIApplication();
        DefaultApplicationModel model=new DefaultApplicationModel();
        model.setViewClass(HelloView.class);
        app.setModel(model);
        app.launch(args);
    }
}
```

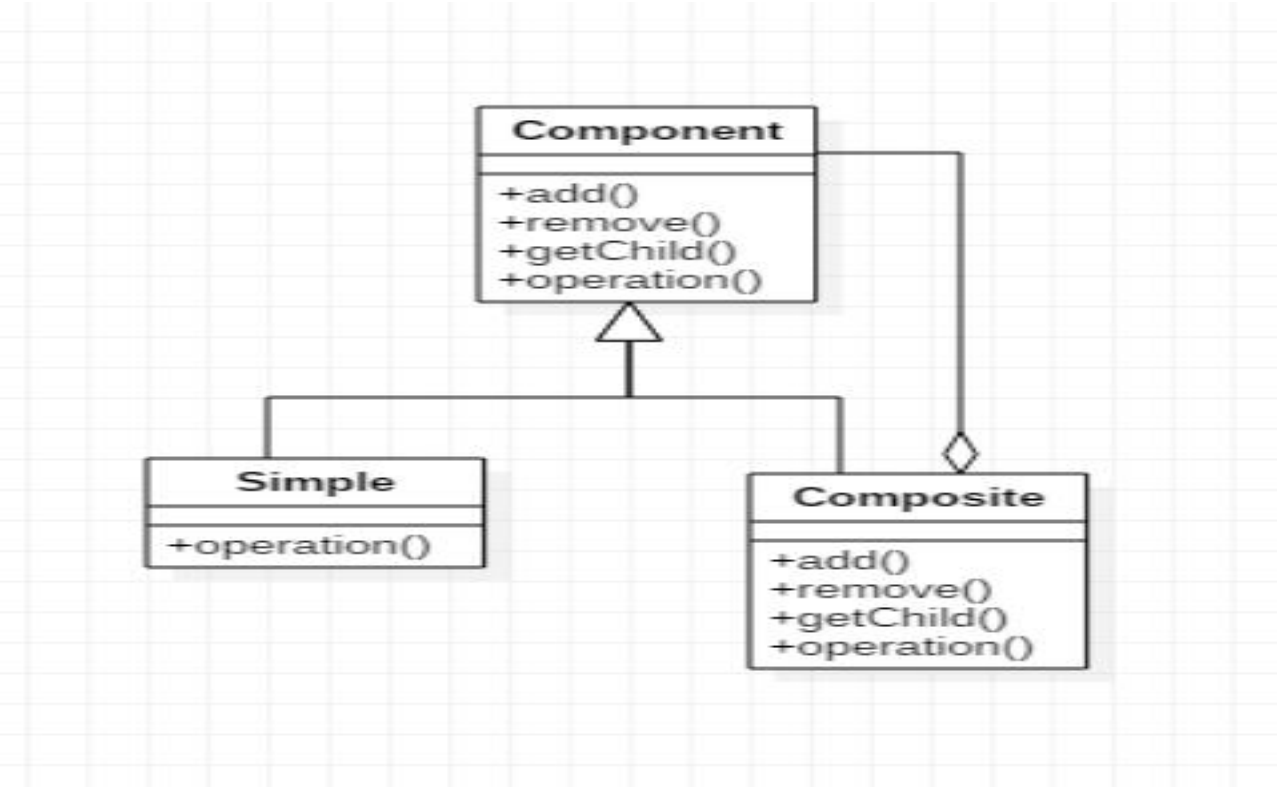
- 首先是模型的建立(model)(即上面第8行代码)。这里的model不是业务逻辑的model，而是应用程序的model，它是在应用程序层次上对应用程序建模而来的
- model封装了程序名称，程序版本号以及一系列的action(控制器)，model还存放了应用程序的视图(view)(第10行代码)
- 当程序运行时，application便调用model的getView()方法来获取视图并将其显示，而用户通过点击控制器(按钮、菜单)激活控制器，控制器将用户的操作传送给model处理，model将处理完之后，显示相应的视图来告知用户的操作结果


User
AbsModel
Application
View
createModel() :model
setVersion(version)
setCopyright(copyright)
setViewName(viewName)
createApplication(model) :
app
setModel(model)
启动程序
launch()
init()
初始化: 分离初始化
操作将分了程序生命
周期的控制. 这里也
可以采用策略模式将
初始化和启动两行为
再细分.
model.init(app)
setApplication(app)
initActions()
initModelAction()
装载默认控制器(如菜单File, Exit, About 等)
putAction(action)
将控制器装入model中, 由model控制其生命周期.
initWindows()
对于不同的程序,有不同的菜单和工具栏. 这些都保存
在model中
setLAF()
getMenu() :List<JMenu>
getToolBar() :List<JToolBar>

应用JHotDraw框架的程序启动顺序图

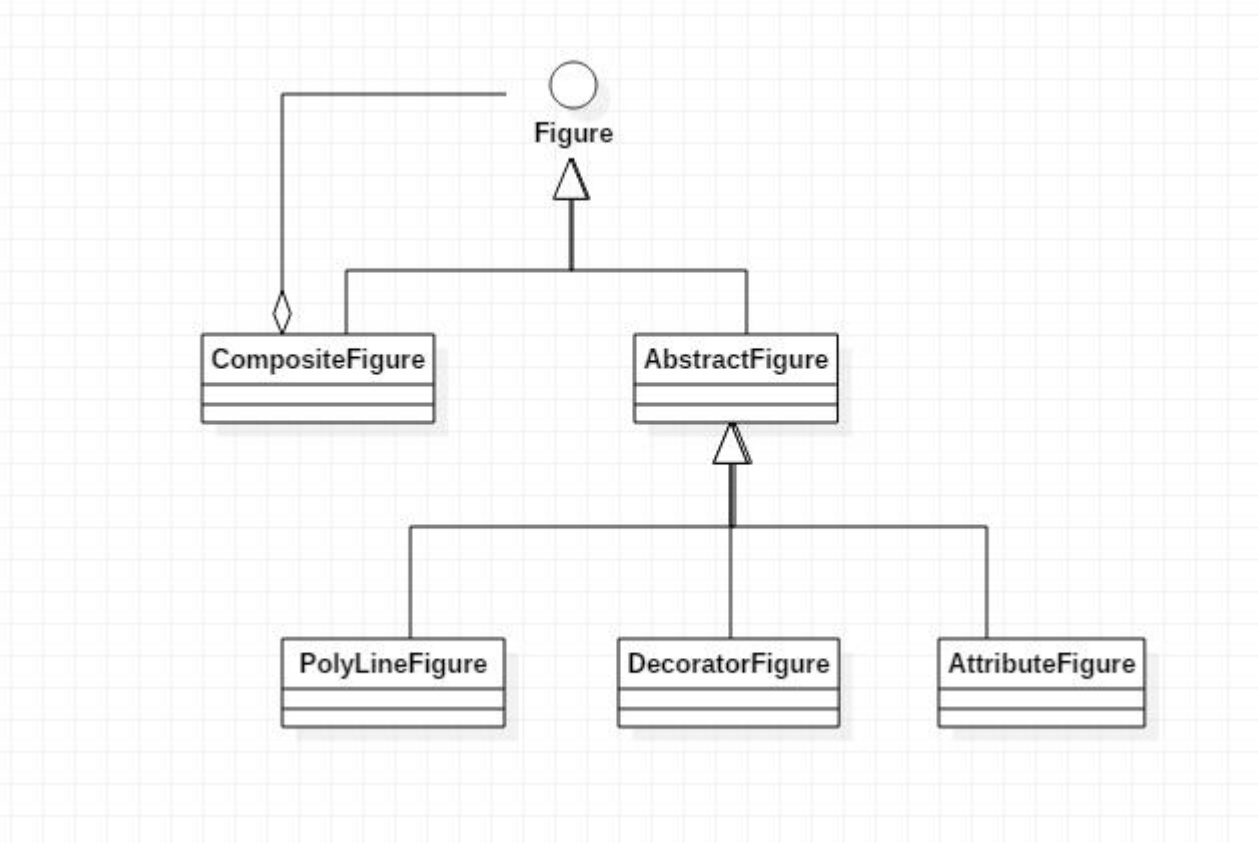


COMPOSITE 设计模式



组合模式(Composite Pattern)：组合多个对象形成树形结构以表示具有“整体—部分”关系的层次结构。组合模式对单个对象（即叶子对象）和组合对象（即容器对象）的使用具有一致性

JHOTDRAW中的COMPOSITE模式



JHOTDRAW中的COMPOSITE模式

- 复合图形由多种简单图形叠加而成
- 将复合图形看作容器节点，简单图形看作叶子节点
- 利用递归调用将容器节点与叶子节点的操作统一化，使程序能有更好的扩展性

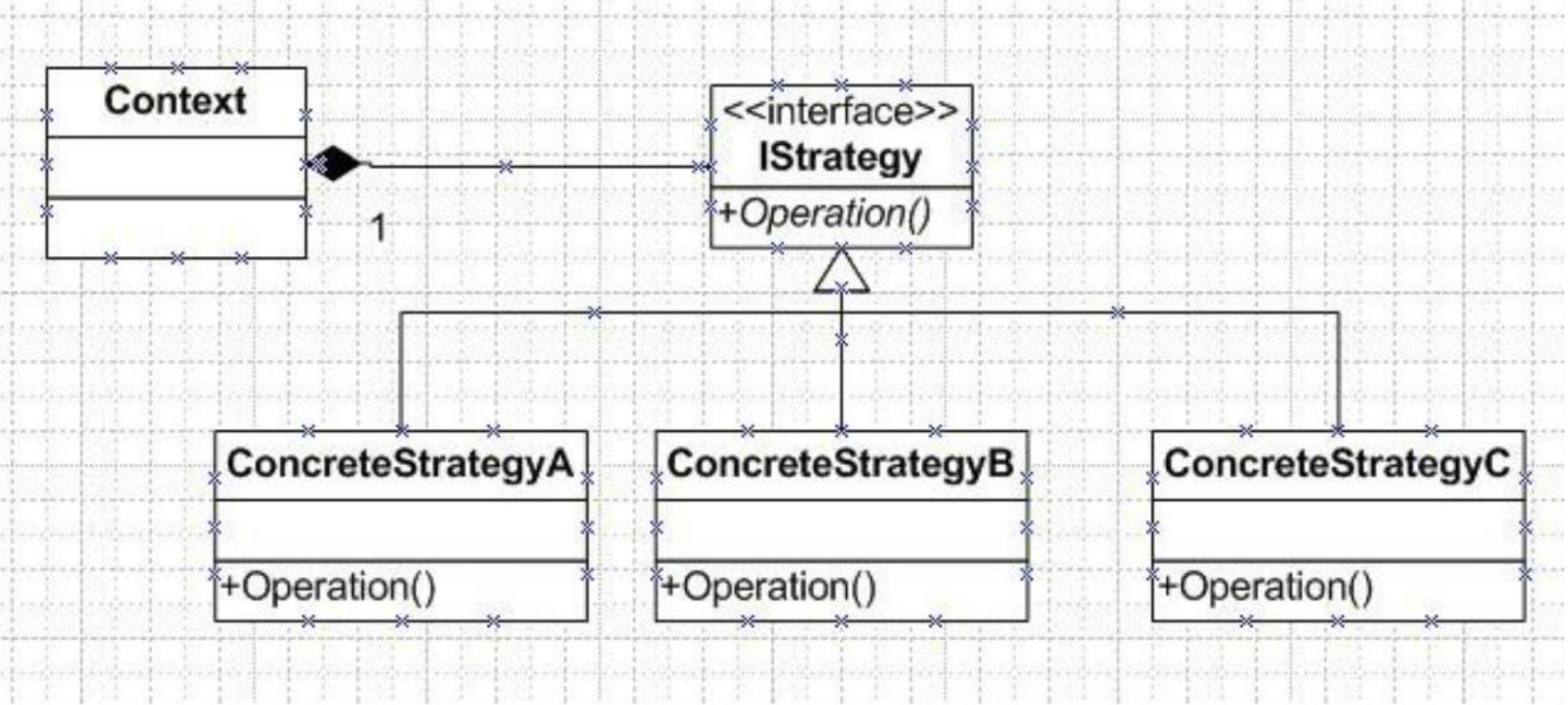
STRATEGY设计模式

- Strategy策略模式是一种对象行为模式。

- 主要是应对在软件构建过程中，某些对象使用的算法可能多种多样，经常发生变化。如果在对象内部实现这些算法，将会使对象变得异常复杂，甚至会造成性能上的负担。故把算法和使用算法的客户端分开（把行为和环境分割开），从而方便的选择其中一个算法。

- 正如在GoF《设计模式》中说道：定义一系列算法，把它们一个个封装起来，并且使它们可以相互替换。该模式使得算法可独立于它们的客户变化。

STRATEGY设计模式结构图



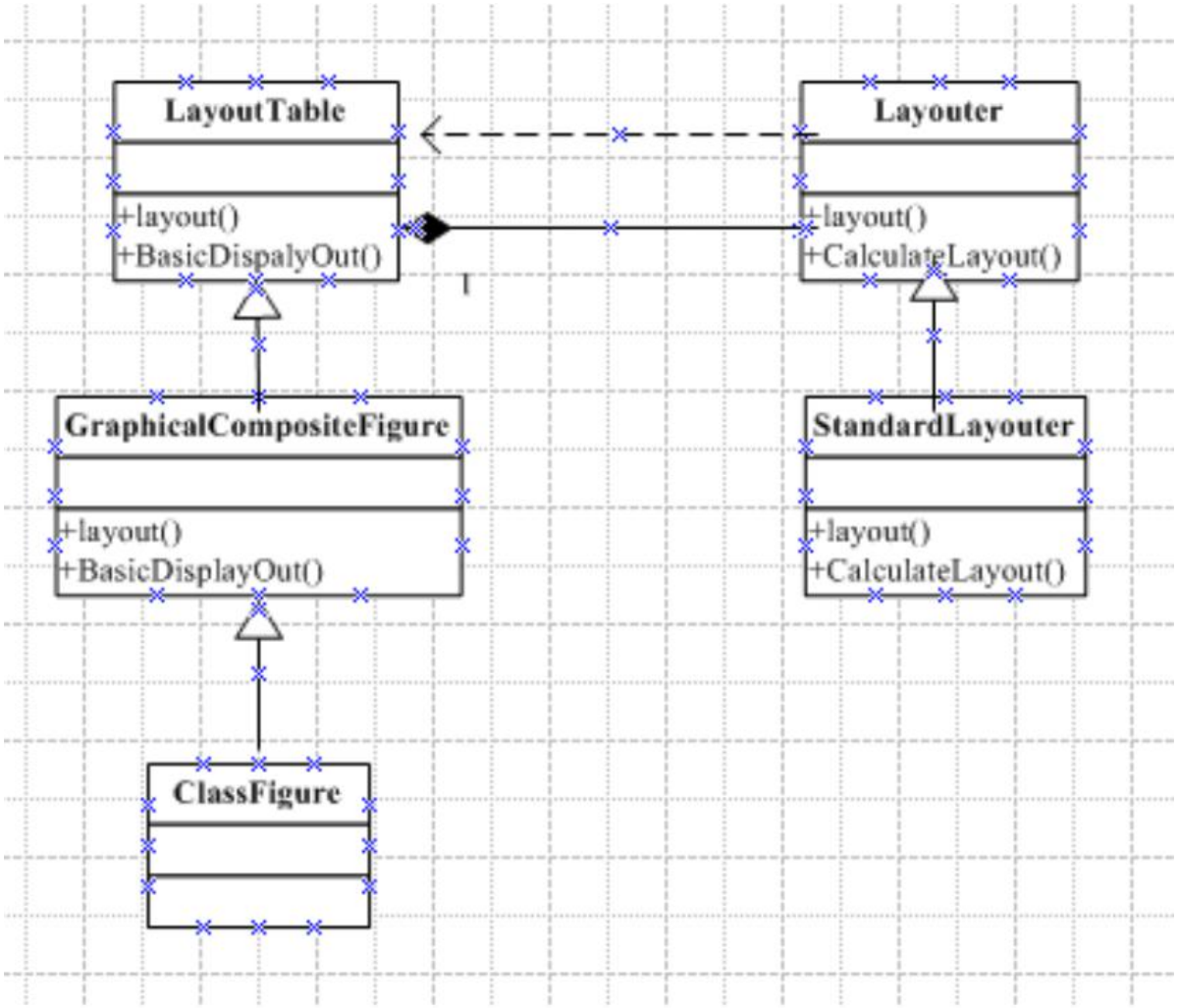
从图中可知Strategy模式实际上就是将算法一一封装起来，ConcreteStrategyA、ConcreteStrategyB、ConcreteStrategyC，但是它们都继承于一个接口，这样在Context调用时还可以以多态的方式来实现对于不用算法的调用。

STRATEGY模式使用场合

场合很多

- 系统有许多类而他们的区别仅仅在于它们的行为
- 动态选择几种算法中的一种
- 一个对象有很多行为等等

JHOTDRAW中的STRATEGY模式



JHOTDRAW中的STRATEGY模式

- 关于JhotDraw 中的策略设计模式，在上图中ClassFigure 只对画图负责，它并不知道ClassFigure怎样布局。

- 事实上，图形表现是独立于布局算法的。因此，布局算法与ClassFigure 是分离的，并封装在一个外部组件里，这个组件对主组件有广泛的访问权和控制权。

- 如果我们要放置一个ClassFigure 时，它便委派任务给Layouter，它遍历所有的子元素并安排这些元素的布局。这种算法与内容的分离可以使你在运行时动态地改变行为并增加算法的可重用性。

JHOTDRAW中的STRATEGY模式

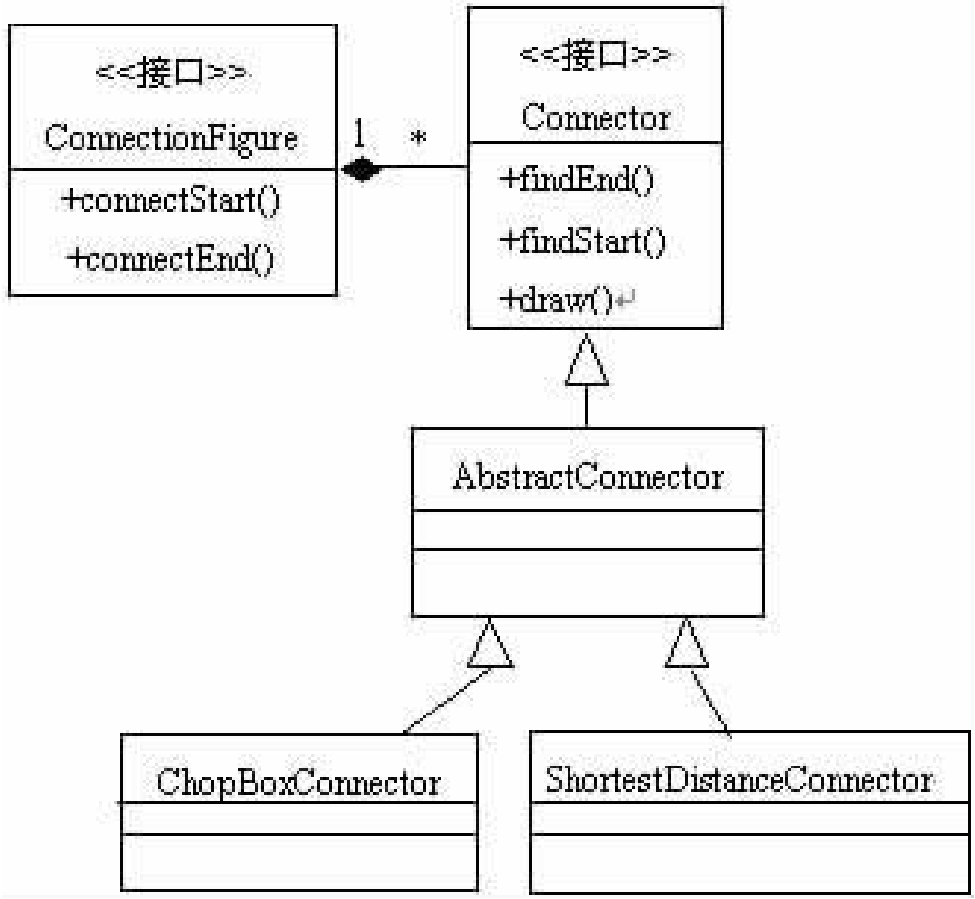
这种机制与安排 java.awt.Window的内容的

java.awt.LayoutMananger相似，这种算法与内容的

分离在设计模式中称为Strategy ， Strategy可以使你

在运行时动态地改变行为并增加算法的可重用性

JHOTDRAW中的STRATEGY模式



JHOTDRAW中的STRATEGY模式

- 接口Connector 的定义

```
public interface Connector extends Serializable，Storable {
    public abstract Point findStart(ConnectionFigure connection)；
    public abstract Point findEnd(ConnectionFigure connection)；
}
```

JHOTDRAW中的STRATEGY模式

- 类AbstractConnector 定义

```
public abstract class AbstractConnector implements Connector{
    public Point  findStart(ConnectionFigure connection) {
        return findPoint(connection);
    }
    public Point  findEnd(ConnectionFigure connection) {
        return findPoint(connection);
    }
}
```

JHOTDRAW中的STRATEGY模式

- 类ChopBoxConnector定义

```
public class ChopBoxConnector extends AbstractConnector {
    public Point findStart(ConnectionFigure connection) {
        Figure startFigure = connection.start().owner();
        Rectangle r2 = connection.end().displayBox();
        Point r2c = null;
        if (connection.pointCount() == 2)
                    r2c = new Point(r2.x + r2.width/2，r2.y + r2.height/2);
        else
                    r2c = connection.pointAt(1);
        return chop(startFigure，r2c);
    }
```

JHOTDRAW中的STRATEGY模式

- 类ShortestDistanceConnector 定义

```
public class ShortestDistanceConnector extends AbstractConnector {
    public Point findStart(ConnectionFigure connection) {
        return findPoint(connection，true)；
    }
    public Point findEnd(ConnectionFigure connection) {
        return findPoint(connection，false)；
    }
    }
```

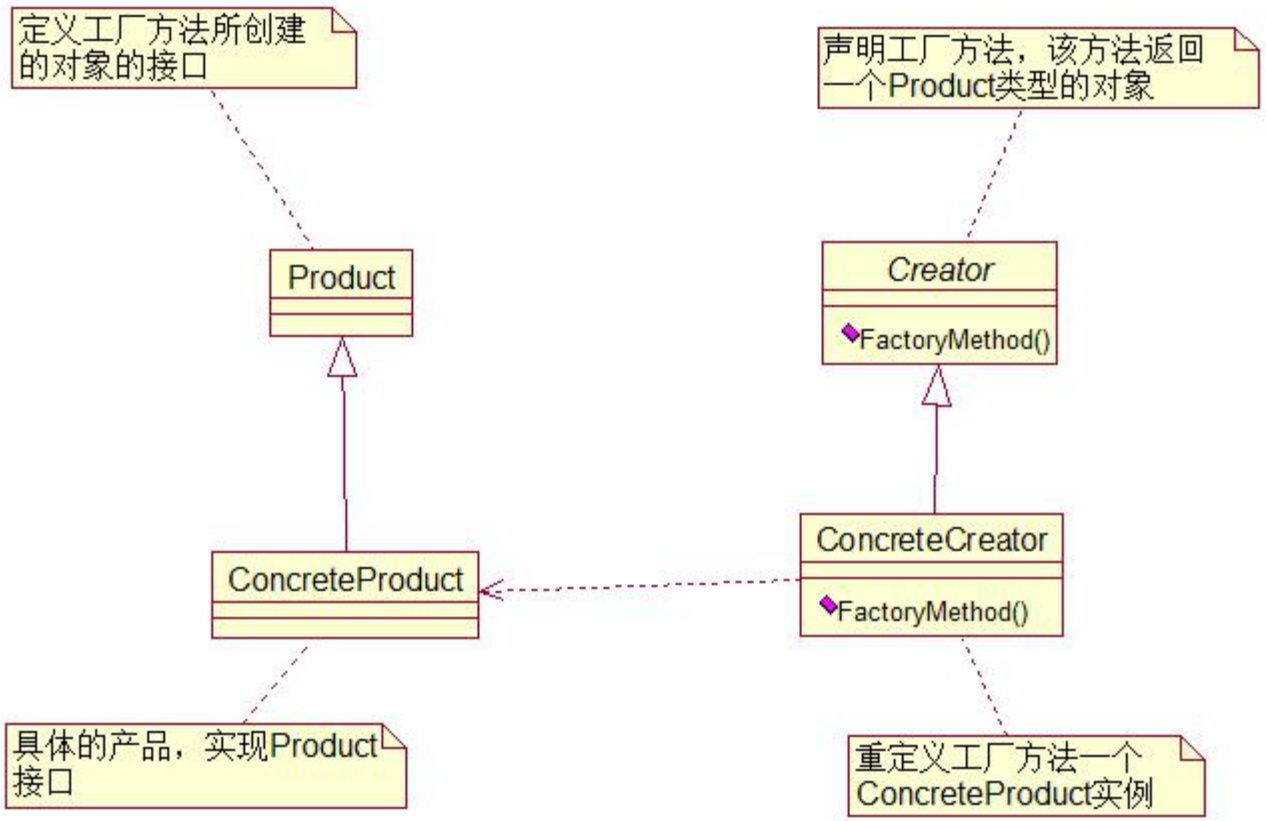
FACTORY METHOD设计模式

- 工厂模式专门负责将大量有共同接口的类实例化。工厂模式可以动态决定将哪一个类实例化，不必事先知道每次要实例化哪一个类。

- 工厂模式有以下几种形态：简单工厂（Simple Factory）模式、工厂方法（Factory Method）模式、抽象工厂（Abstract Factory）模式。

- 针对JHotDraw，这里说的是工厂方法（Factory Method）模式。

FACTORY METHOD设模式结构图



FACTORY METHOD模式的好处

- Factory Method可以根据不同的条件产生不同的实例，当然这些不同的实例通常是属于相同的类型，具有共同的父类。

- Factory Method把创建这些实例的具体过程封装起来了，简化了客户端的应用，也改善了程序的扩展性，使得将来可以做最小的改动就可以加入新的待创建的类。

JHotDraw中的Factory Method

- Factory Method在JHotDraw中被广泛的使用，特别是在创建用户界面组件(比如，菜单以及Tools)时，在CH.ifa.draw.application.DrawApplication中可以发现许多Factory Method，比如createTools()、createMenus()、CreateFileMenu()等等。

- 可以在相应的方法中做一些变化，比如，为建立自己的类以及在类之间建立关联关系，可以在主应用类中重载createTools0方法。

CREATETOOLS的例子

```
public class JModellerApplication extends MDI_DrawApplication {

        …

        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());
```

CREATETOOLS的例子

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

```
protected Tool createSelectionTool() {

    return new DelegationSelectionTool(view());

}
```

其他的设计模式

- 观察者模式（Observer）
- 模板设计模式（Template Method）
- 原型设计模式（Prototype）
- ……

Q&A