★最后一道大题分析设计题（该题不出意外就是这个了）

问题：

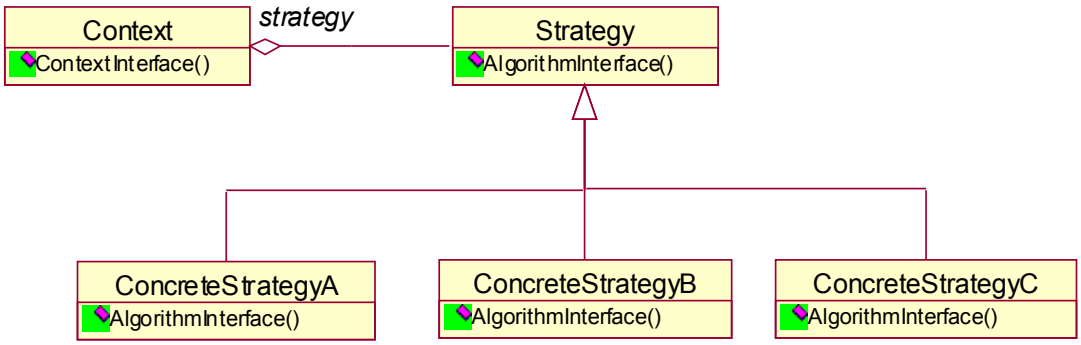
选用一种设计模式实现以下功能

用户需要对一个正文流进行分行，当前要求有以下三种算法来进行分行。

- 一次决定一个换行位置的简单分行算法；
- 一次处理一段文字的换行算法；
- 使得每一行都含有一个固定数目的项分行算法。

不同的时候需要不同的算法，不想支持并不使用的换行算法；且希望能较容易地增加新的换行算法或改变现有算法。

答：可采用策略模式来解决：定义一系列的算法,把它们一个个封装起来， 并且使它们可相互替换。此模式使得算法可独立于使用它的客户而变化。



UML结构图（一定要会画）

参与者

• **Strategy(策略类Compositor)**

– 定义所有支持的算法的公共接口。Context使用这个接口来调用某Concrete Strategy定义的算法。

• **Concrete Strategy（具体策略类SimpleCompositor,TexCompositor, ArrayCompositor）**

– 以Strategy接口实现某具体算法。

• **Context(上下文类Composition)**

– 用一个Concrete Strategy对象来配置。

– 维护一个对Strategy对象的引用。

– 可定义一个接口来让Strategy访问它的数据。

协作

•Strategy和Context相互作用以实现选定的算法。

• Context将它的客户的请求转发给它的Strategy。

★（PAC和MVC模式共30分，以下两部分内容大家最好都记住）

★简述PAC模式

PAC模式：表示-抽象-控制模式。

以合作Agent的层次形式定义了交互式软件系统的一种结构；

每个Agent负责应用程序的某个特定方面；

每个Agent由表示，抽象，和控制三个组件组成；

将Agent的人机交互部分与其内核和它与其他Agent的通信分隔开来。

场景I：用户要求视图协调程序agent的表示组件打开一个新的直方图。

1.视图协调程序agent的控制组件实例化用户所期望的直方图agent。

2.视图协调程序agent发送一个open事件到新的直方图agent的控制组件。

3.直方图agent的控制组件检索来自顶层agent的数据。视图协调程序agent协调底层和顶层的agent。返回到直方图的数据被存放到他的抽象组件。直方图agent的控制组件调用表示组件显示直方图。

4.表示组件在屏幕上创造窗口，通过控制组件发出请求检索从抽象组件得到的数据并显示。

动态特性（2）：

场景II:说明了新的选举数据输入后系统的行为。

1.用户向电子数据表录入新数据。电子数据表agent的控制组件将数据输送到顶层agent。

2.顶层agent的控制组件更新所有依赖于这些新数据的agent。

3.视图协调程序agent的控制组件把更新通知提交给他所负责协调的所有视图PAC agent。

4.与以前场景一样，所有视图PAC agent都更新他们的数据并且恢复他们展示的图像。

★简述MVC模式

（书上的例子是一个Information system for political selection：它包括一个core data；若干种表现形式，如piechart、cubebarchart、linechart、speadsheet，要求描述它的MVC模式的动态特征）

该模式将一个交互式应用程序分成3个组件。
模型：包含核心功能和数据。
视图：向用户显示信息。
控制器：处理用户输入。
视图和控制器组成了用户接口。
变更-传播机制保证了模型和用户接口之间的一致性。

动态特性（1）

场景I 用户输入导致模型变化，并触发变更-传播机制。

1.控制器接受到事件，解释事件并且启动模型的服务过程。

2.模型执行相应的过程，并导致内部状态的变化。

3.模型调用其更新过程,向所有登记请求了变更-传播机制的视图和控制器发出通知。

4.每个视图从模型中读取新数据并且重新显示。

5.每个控制器修改自己的行为，比如禁用某个功能。

6.最初的控制器恢复控制并从事件处理过程返回。

动态特性(2)

场景II 初始化过程。

1.创建模型实例，并初始化其数据。

2.创建视图对象，并用对模型的引用作为初始化参数之一。

3.视图通过调用附属过程支持变更-传播机制。

4.视图创建控制器，此时将模型和视图的引用作为参数传递给控制器初始化过程。

5.控制器通过调用附属过程来支持变更-传播机制。

6.初始化完成，应用程序开始处理事件。

1.描述软件设计模式的定义和构成（最好全背）

设计模式是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

四个基本要素：模式名称、问题、解决方案、效果。

如何描述设计模式：模式名和分类、意图、别名、动机、参与者、协作、效果、实现。

2.设计模式怎样解决设计问题（知道这两种复用就行）

白箱复用：类继承，通过生成子类的复用，父类的内部细节对子类可见。

黑箱复用：对象组合，新的更复杂的功能可以通过组装或组合对象来获得，对象的内部细节是不可见。

3.简述UML的九种建模图形（了解一下就行，个人觉得用例图、类图、顺序图是重点，最好能画）

用例图：**捕获用户能够看到的系统功能，** 类图：**捕获系统的词汇表，**对象图：**捕获实例和连接，**顺序图：**捕获系统的动态行为(面向时间的)，**协作图：**捕获系统的动态行为 (面向消息的)，**状态图：**捕获系统动态行为(面向事件的)，**活动图：**捕获动态行为（面向活动的），**组件图：**捕获实现的物理结构，**分布图：**捕获系统硬件的拓扑结构。

4.软件体系结构的定义、意义和作用（能回答是什么有什么作用就行）

定义：软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。

意义：（1）体系结构是风险承担者进行交流的手段；（2）体系结构是早期设计决策的体现；（3）软件体系结构是可传递和可重用的模型。

作用：（1）基于体系结构设计思想，有助于设计者面临复杂领域问题时，做出正确的选择，最大限度地避免软件设计的结构性错误；（2）体系结构设计文档成为设计人员、用户及其他风险参与者一致的沟通文本，以保证软件产品开发的成功率；（3）体系结构有助于发现和提取可重用构件或模式。

5.三种软件体系结构评估方法（一道填空一道选择）

（1）体系结构权衡分析方法（ATAM方法）

（2）体系结构结构分析方法（SAAM方法）

（3）中间设计的积极评审（ARID方法）

ATAM分为：**第一阶段以体系结构为中心，重点是获取体系结构信息并进行分析；第二阶段以风险承担者为中心，重点是获取风险承担者的观点，验证第一阶段的结果。**

6.“4+1”模型 SA模型 生命周期模型（知道有这么个东西就行了）

“4”代表逻辑视图、进程视图、物理视图、开发视图，“1”代表场景。

SA核心模型由五种元素构成：由构件（component），连接子（connector），配置（configuration），端口（port）和角色（role）。

生命周期模型：1.非形式化描述2.规范化描述3.求精及其验证4.实施5.评估6.演化

7．软件体系结构风格（知道有这么个东西就行）

管道/过滤器风格、层次风格、正交风格、消息总线风格、PAC风格、解释器

8. 简述基于体系结构的软件开发过程(最好能记住)

基于体系结构的软件过程是在体系结构指导下的软件开发过程。首先设计体系结构，软件系统的开发过程可描述为软件的演化与组装过程。具体过程可划分为体系结构的需求、设计、文档化、复审、实现、演化等6个子过程。

9.简述建模

建模是开发优秀软件的所有活动中的核心部分，其目的是**把所要设计的结构和系统的行为沟通起来，并对系统的体系结构进行可视化控制**。

10.23种设计模式（选择题，形式是描述一种模式，让你选择是那种模式，代理模式大家可以多了解一点）

Abstract Factory：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。
Builder：将一个复杂对象的构件与它的表示分离，使得同样的构建过程可以创建不同的表述。
Factory Method：定义一个用于创建对象的接口，让子类决定将哪一个类实例化。Factory Method使一个类的实例化延迟到其子类。
Prototype：用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象。
Singleton：保证一个类仅有一个实例，并提供一个访问它的全局访问点。
Adapter：将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。
Bridge：将抽象部分与它的实现部分分离，使它们都可以独立地变化。
Composite：将对象组合成树型结构以表示"部分- 整体"的层次结构。Composite使得客户对单个对象和复合对象的使用具有一致性。
Decorator：动态地给一个对象添加一些额外的职责。就扩展功能而言，Decorator模式比生成子类方式更为灵活。
Facade：为子系统中的一组接口提供一个一致的界面，Facade模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。
Flyweight：运用共享技术有效地支持大量细粒度的对象。
Proxy：为其他对象提供一个代理以控制对这个对象的访问。
Chain of Responsibility：为解除请求的发送者和接受者之间耦合，而使多个对象都有机会处理这个请求。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。
Command：将一个请求封装为一个对象，从而使你可以用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可取消的操作。
Interpreter：给定一个语言，定义它的文法的一种表示，并定义一个解释器，该解释器使用该表示来解释语言中的句子。
Interator：提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。
Mediator：用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使器耦合松散，而且可以独立地改变它们之间的交互。
Memento：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可以将该对象恢复到保存的状态。
Observer：定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新。
State：允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它所属的类。
Strategy：定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法的变化可独立于使用它的客户。
Template Method：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。
Visitor：表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。