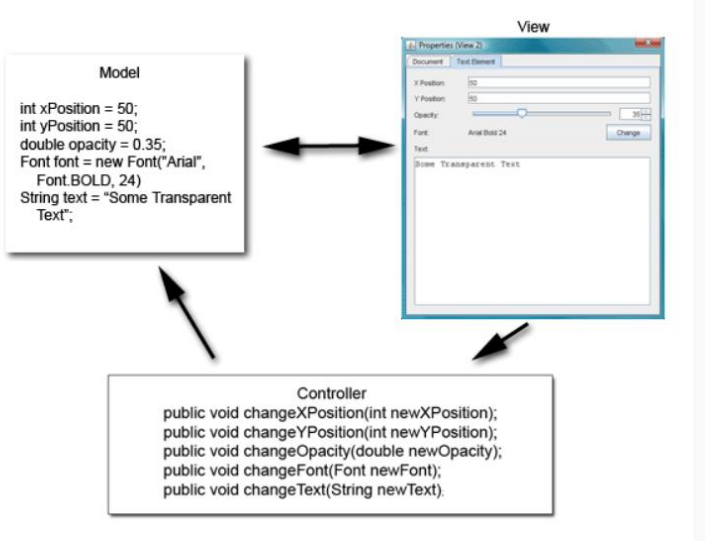
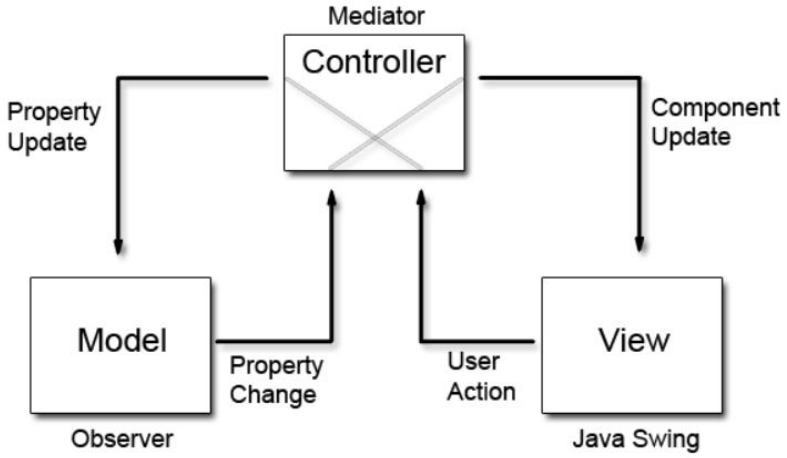
Homework：

1、 阅读：Java SE Application Design With MVC

答： 这篇 Oracle 的文章讨论了 MVC（Model-View-Controller）设计模式在 J2EE 应用开发中的应用。这个模式将应用程序的功能分为模型、视图和控制器三个部分，以提高代码的可维护性和可扩展性。其中，模型处理数据和业务逻辑，视图管理用户界面，控制器处理用户输入并更新模型和视图，具体如图所示：



文章中也提到了 MVC 设计的一种较新的实现是将控制器置于模型和视图之间。其主要区别在于模型对象的状态变化通知通过控制器传递给视图。控制器在模型和视图对象之间双向调节数据流。视图对象使用控制器将用户操作转化为模型的属性更新，并通过控制器将模型状态的变化传递给视图。采用这种设计有助于更彻底地解耦模型和视图，使控制器能够管理模型属性和视图方法。



接着文章展示了如何实践这种设计，首先从模型开始。假设要使用具有五个属性的简单显示模型来绘制一些文本。定义了 AbstractModel 类，使用 PropertyChangeSupport 类注册、注销和通知监听器关于模型变化的通知。定义 AbstractController 抽象类，维护已注册的视图和模型，并处理属性变化事件。然后

定义 DefaultController 类，包含属性常量和由视图的 GUI 事件监听器调用的方法。再接着定义 PropertiesViewPanel 类，初始化 Swing 组件并设置文档监听器。应用的设计问题主要在于 Swing 组件的自我更新逻辑，可能导致无限循环或冗余更新，解决方案是在组件更新前检查新旧值是否相同。通过以上设计，MVC 模式可以更有效地分离关注点，提升代码的可维护性和扩展性。

阅读这篇文章后，我对 MVC 设计模式有了更深的理解。特别是在 Java 中的实现示例，使我对如何实际应用这一设计模式有了更清晰的认识。这篇文章不仅理论讲解清晰，还提供了实用的代码示例，对实际编程有很大的指导意义。

2、 LoD 原则强调“只和朋友通信，不和陌生人说话”。请举例说明 “朋友圈”认定依据是啥？

答：LoD（Law of Demeter，迪米特法则）强调“只和朋友通信，不和陌生人说话”，旨在降低系统的耦合度，提高代码的可维护性和可扩展性。LoD 原则的核心思想是一个对象应尽量少了解其他对象，从而减少代码之间的依赖关系。

我认为以下几条认定依据是符合 LoD 原则的：

a) 当前对象自身的成员变量、方法和属性，比如说以下代码：

```
class Student {
    private Address address;

    public void printCity() {
        System.out.println(address.getCity());
    }
}
```

Student 类可以直接访问它的成员变量 address。

b) 当前方法的参数对象，例如：

```
class Student {
    public void sendTranscript(Transcript transcript) {
        transcript.send();
    }
}
```

在这个例子中，Student 类可以直接与方法参数 transcript 通信。

c) 当前方法调用另一个方法时，该方法的返回值，比如说：

```
class Student {
    private Address address;

    public String getCity() {
        return address.getCity();
    }
}
```

在这个例子中，Student 类可以与 address.getCity()返回的 String 对象通信。

d) 在当前对象的方法中创建的对象，例如：

```
class Student {
    public void createAddress() {
```

```
            Address address = new Address();
            address.setCity("Xia Men");
        }
    }
```

Student 类可以直接与它创建的 Address 对象通信。

e) 如果当前对象是一个容器类，那可以与容器内的元素通信，例如：

```
class Class {
    private List<Student> students;

    public void notifyAllMembers() {
        for (Student student : students) {
            student.notify();
        }
    }
}
```

Class 类可以与 students 列表中的每个 Student 对象通信。

假设有以下一个程序：

```
class Engine {
    public void start() {
    }
}
class Car {
    private Engine engine;
    public void startCar() {
        engine.start();
    }
}
class Driver {
    private Car car;
    public void drive() {
        car.startCar();
    }
}
```

根据 LoD 原则，Driver 类与 Car 类通信，Car 类与 Engine 类通信，这是合理的。但如果 Driver 类直接调用 Engine 类的方法，就违反了 LoD 原则，例如：

```
class Driver {
    private Car car;
    public void drive() {
        car.getEngine().start();
    }
}
```

这就违反了 LoD 原则，Driver 类不应该直接与 Engine 类通信，而是通过 Car 类进行操作。这是因为 Driver 类与 Car 类，Car 类与 Engine 类是分别属于各自的朋友。