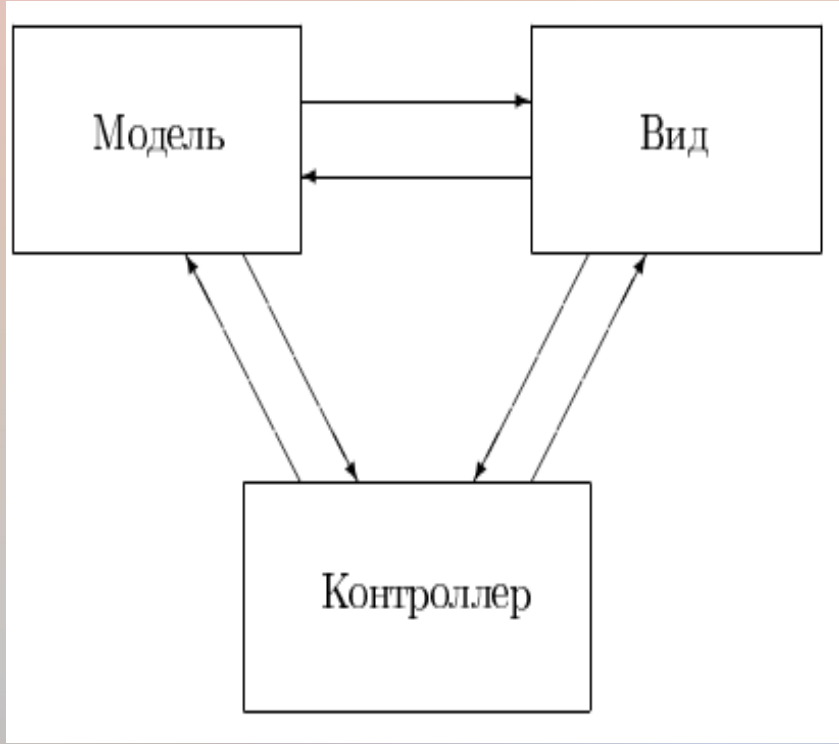
Концепция Модель-Вид-Контроллер (MVC, Model-View-Controller)

Модель – данные, с которыми оперирует программное приложение и их обработка

Вид – представление данных пользователю

Контроллер – обработка запросов пользователя и использование модели и вида для реализации необходимой реакции



Модель

- Реализует совокупность правил, принципов, зависимостей поведения объектов предметной области (бизнес-логику приложения)
- Не зависит ни от вида, ни от контроллера
- Для некоторых проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;
- Для других проектов модель — это просто слой данных (база данных, XML-файл)

Классы модели

- Хранят данные в виде полей
- Имеют методы для получения значений данных в нужном формате
- Имеют методы для изменения данных
- Имеют поля и методы для сообщения об изменении данных (активная модель)

Вид (представление)

- Отображает данные полученные от модели
- Не может напрямую влиять на модель
- Часто в качестве представления выступает окно с графическими элементами
- К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели
- В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику

Класс вида

- Реализует окно или элемент сцены с графическими элементами
- Получает информацию о данных, хранимых в модели, через ссылку на объект модели (ссылка является полем в классе вида)
- Имеет методы для изменения графических элементов при изменении данных в модели

Контроллер

- Управляет запросами пользователя
- Вызывает и координирует действия

необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем, обычно контроллер вызывает соответствующую модель и выбирает подходящий вид

- Не затрагивая реализацию видов и модели, можно изменить реакцию на действия пользователя, для этого достаточно использовать другой контроллер

Класс контроллера

- Создает и работает с объектами классов модели и вида
- Содержит методы, описывающие обработку действий пользователя
- Инициирует изменение модели, в соответствии с действиями пользователя
- Может сочетать функции контроллера и вида

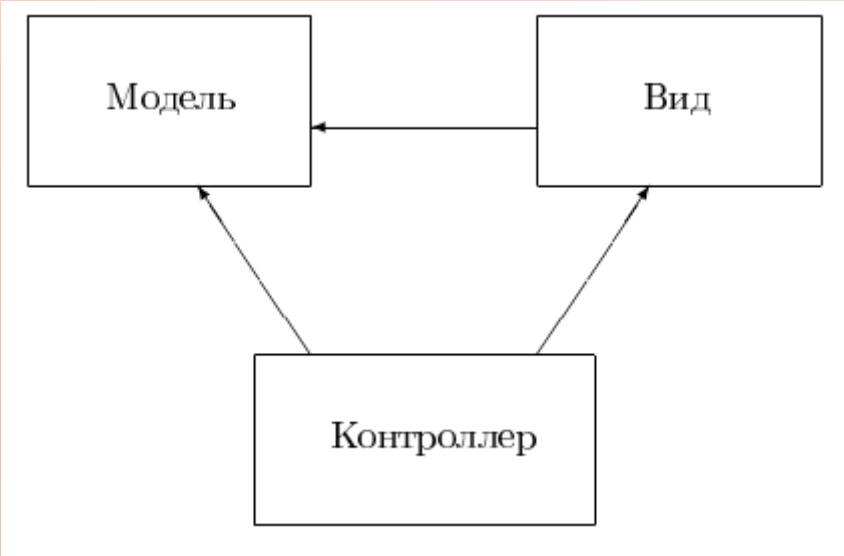
Основные модификации MVC

Пассивная модель

- не имеет никаких способов воздействовать на представление или контроллер
- изменения модели отслеживаются контроллером и он же отвечает за перерисовку представления

Активная модель

- оповещает представление о том, что в ней произошли изменения



Шаблон Observer или Listener (Наблюдатель или Слушатель)

Наблюдаемый (observable) **объект**, в котором могут произойти изменения (объект класса модели)

- должен предоставлять интерфейс для регистрации и удаления наблюдателей (listeners)

Наблюдатель (Observer или Listener) зависит от состояния модели

- должен регистрироваться в качестве наблюдателя и иметь метод, описывающий реакцию на изменение наблюдаемого объекта (вид, контроллер)

Программная реализация

Интерфейс

interface Listener { public void dataChanged();}

Модель

ArrayList<Listener> listeners // список слушателей

void addListener(Listener l) // добавление слушателя

void deleteListener(Listener l) // удаление слушателя

void fire() // оповещение всех слушателей

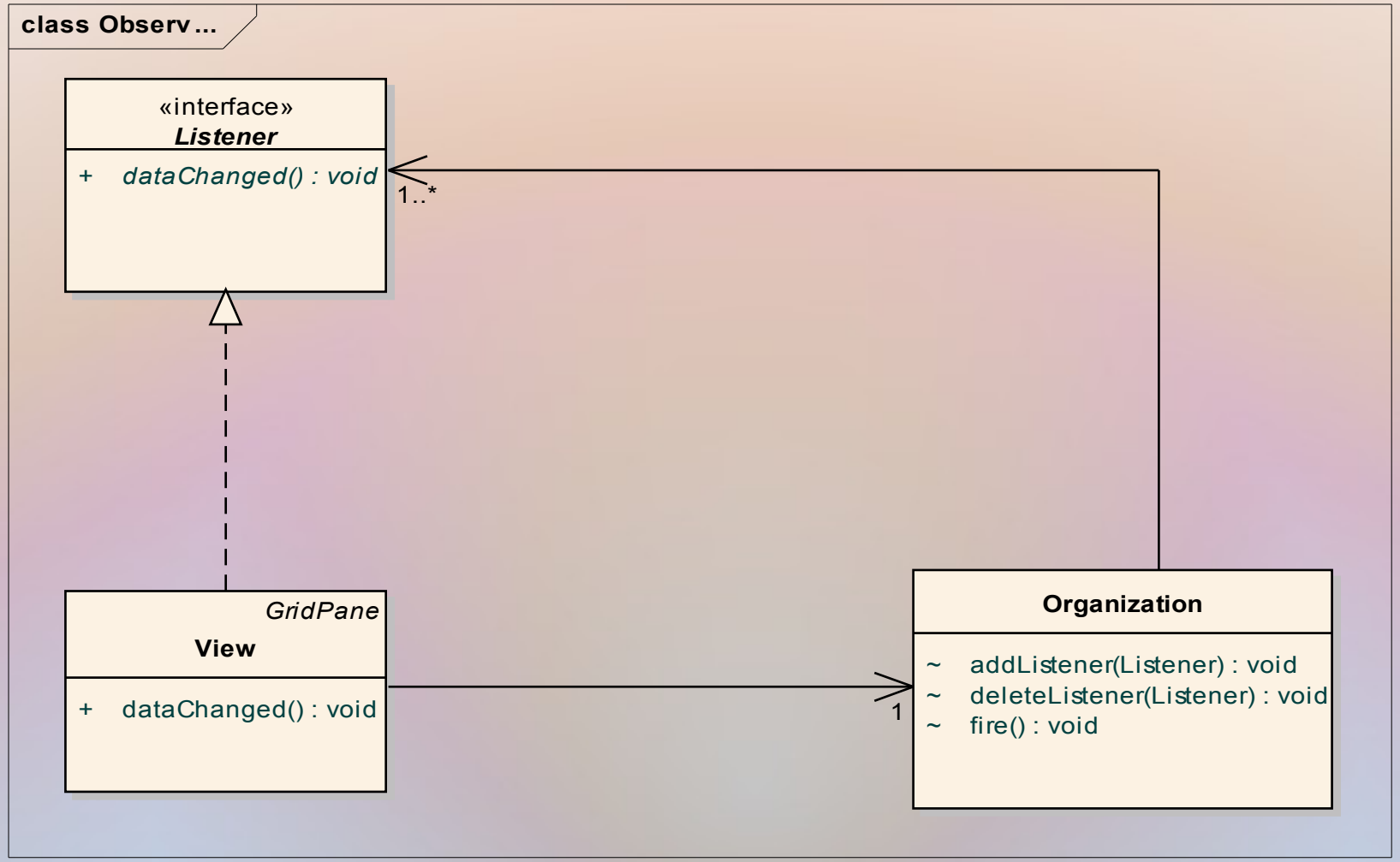
// вызов метода fire() везде, где происходят изменения полей

Вид

Реализация интерфейса Listener

Регистрация в качестве слушателя (с помощью метода модели addListener())

Диаграмма классов



Шаблон Listener в классах JavaFX

Специальные компоненты Java Beans – свойства
Class DoubleProperty
Class IntegerProperty
Class StringProperty
Свойство можно связать с графическим элементом
Class Bindings
static <T> void bindBidirectional
(Property<T> property1, Property<T> property2)
property1 – наблюдатель
property2 – поле модели
Метод свойства
void bind(ObservableValue<? extends T>
observable)

Пример свойства

Class IntegerProperty – абстрактный класс
Class SimpleIntegerProperty – соответствующий класс
Конструктор только по умолчанию
IntegerProperty()
Изменение значения
void set(int value)
Получение значения
int get()
Integer getValue()

Добавление в модель поля-свойства

Поле
private StringProperty drinkVolumeString
Три обязательных метода
public StringProperty drinkVolumeStringProperty()
public final void setDrinkVolumeString(String value)
public final String getDrinkVolumeString()

Поле
private DoubleProperty drinkVolume;
Три обязательных метода
public DoubleProperty drinkVolumeProperty()
public final void setDrinkVolume(double value)
public final double getDrinkVolume()

Привязка к графическим объектам

```
Label drinkVol = new Label();

Bindings.bindBidirectional
(drinkVol.textProperty(),
org.drinkVolumeStringProperty());

drinkVol.textProperty().bind
(org.drinkVolumeStringProperty());
```