

# Визуальное моделирование, UML

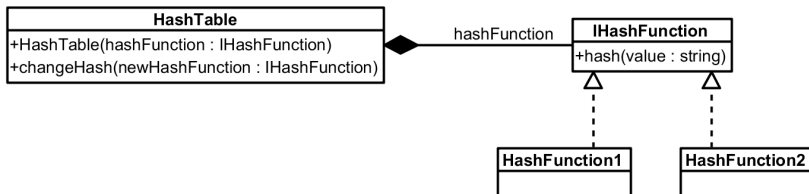
Юрий Литвинов  
yurii.litvinov@gmail.com

24.04.2020г

# Визуальное моделирование

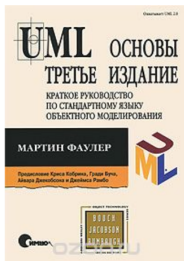
- ▶ Модели
  - ▶ Содержат меньше информации, чем код
- ▶ Метафора моделирования
  - ▶ Не более чем соглашение между разработчиками
- ▶ Цель моделирования
  - ▶ Каждая модель существует для кого-то и для чего-то
  - ▶ Модели как средства общения, как документация и как графические исходники

# Пример модели



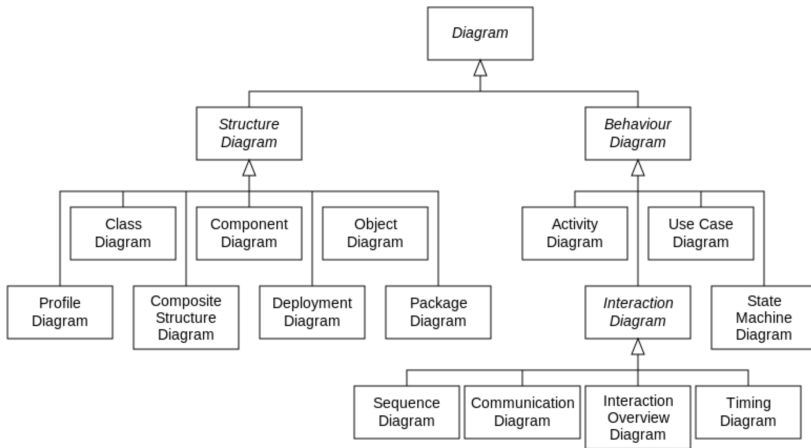
# Unified Modeling Language

- ▶ Семейство графических нотаций
  - ▶ 14 видов диаграмм
- ▶ Общая метамодель
- ▶ Стандарт под управлением Object Management Group
  - ▶ UML 1.1 — 1997 год
  - ▶ UML 2.5.1 — декабрь 2017 года
- ▶ Прежде всего, для проектирования ПО

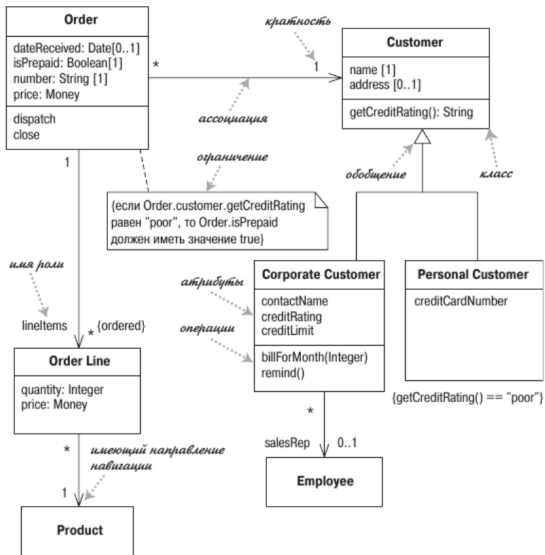


М. Фаулер, UML. Основы. Краткое руководство по стандартному языку объектного моделирования. СПб., Символ-Плюс, 2011. 192 С.

# Диаграммы UML



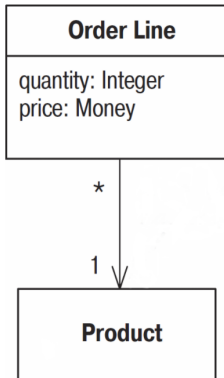
# Диаграммы классов UML



© М. Фаулер. "UML. Основы"

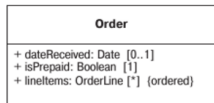
# Как это связано с кодом

```
public class OrderLine {  
    private int quantity;  
    private Product product;  
    public int getQuantity() {  
        return quantity;  
    }  
    public void setQuantity(int quantity) {  
        this.quantity = quantity;  
    }  
    public Money getPrice() {  
        return product.getPrice().multiply(quantity);  
    }  
}
```

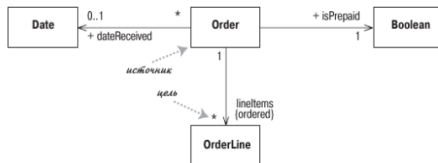




# Свойства



## Атрибуты



## Ассоциации

### Синтаксис:

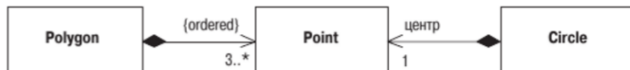
- ▶ видимость имя: тип кратность = значение по умолчанию {строка свойств}
- ▶ Видимость: + (public), - (private), # (protected), ~(package)
- ▶ Кратность: 1 (ровно 1 объект), 0..1 (ни одного или один), \* (сколько угодно), 1..\*, 2..\*

# Агрегация и композиция

Агрегация – объект “знает” о другом (не управляет его временем жизни, имеет на него ссылку или указатель)

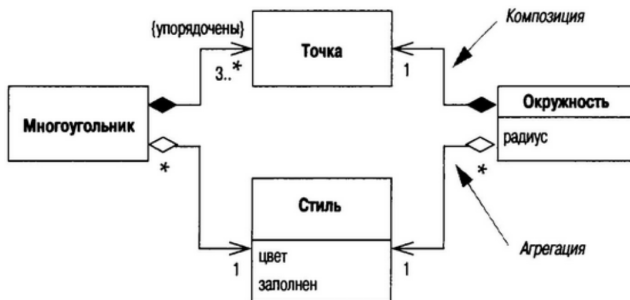


Композиция — объект владеет другим объектом (управляет его временем жизни, хранит его по значению или по указателю, делая delete)



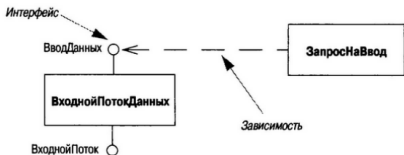
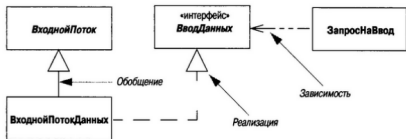
Уточнение обычной ассоциации, используется только если очень надо

# Агрегация и композиция, пример

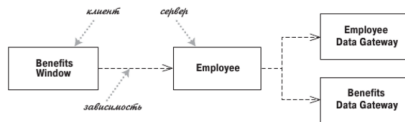


© М. Фаулер. "UML. Основы"

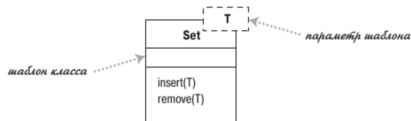
## Интерфейсы



## Зависимости

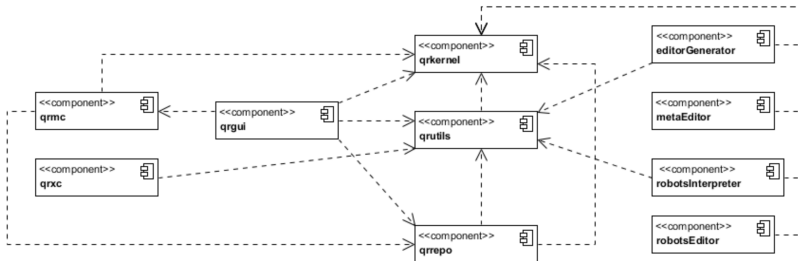


## Шаблоны

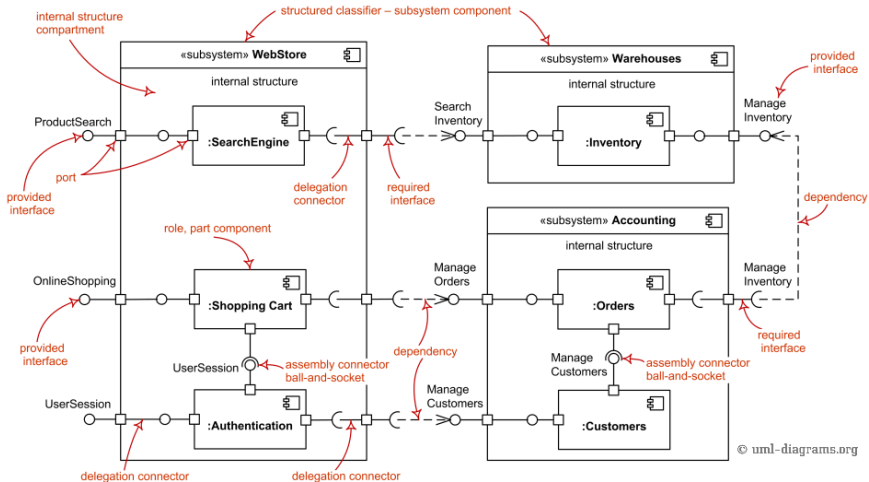


# Диаграммы компонентов

## Component diagrams



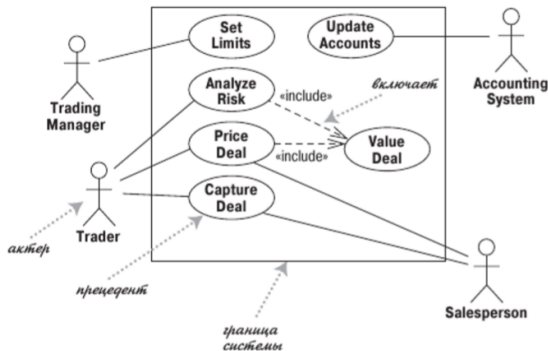
# Более подробно



© <http://www.uml-diagrams.org>

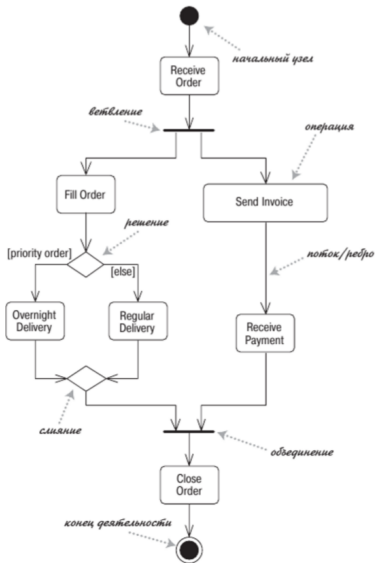
# Диаграммы случаев использования

## Use case diagrams



# Диаграммы активностей

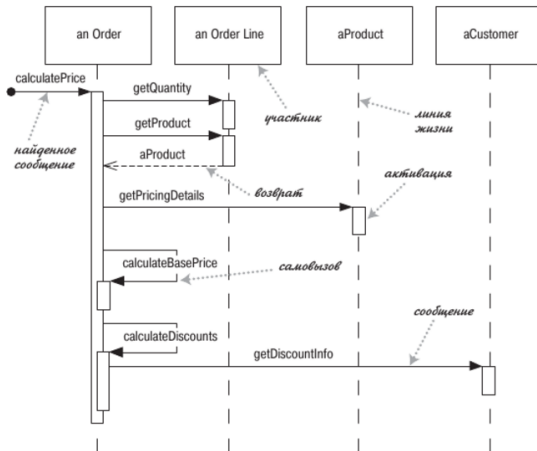
## Activity diagrams





# Диаграммы последовательностей

## Sequence diagrams



# Диаграммы конечных автоматов

## State diagrams

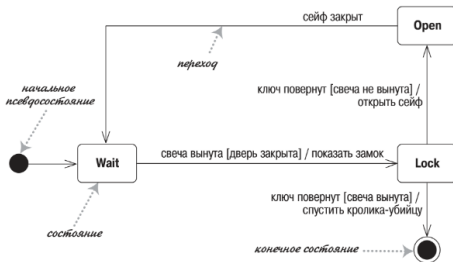
### ► Состояние

- entry activity
- exit activity
- do activity
- внутренний переход

### ► Событие

### ► Переход

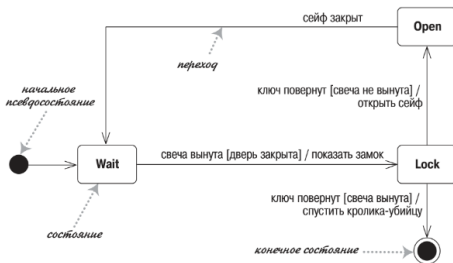
- имя события (список параметров) [сторожевое условие] выражение действия



© М. Фаулер, UML. Основы

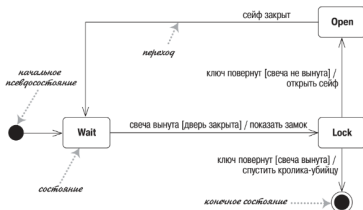
# Генерация кода

```
public void handleEvent(PanelEvent anEvent) {  
    switch (currentState) {  
        case PanelState.Open:  
            switch (anEvent) {  
                case PanelEvent.SafeClosed:  
                    currentState = PanelState.Wait;  
            }  
            break;  
        case PanelState.Wait:  
            switch (anEvent) {  
                case PanelEvent.CandleRemoved:  
                    if (isDoorOpen) {  
                        revealLock();  
                        currentState = PanelState.Lock;  
                    }  
            }  
            break;  
        case PanelState.Lock:  
            switch (anEvent) {  
                case PanelEvent.KeyTurned:  
                    if (isCandleIn) {  
                        openSafe();  
                        currentState = PanelState.Open;  
                    } else {  
                        releaseKillerRabbit();  
                        currentState = PanelState.Final;  
                    }  
            }  
            break;  
    }  
}
```



© М. Фаулер, UML. Основы

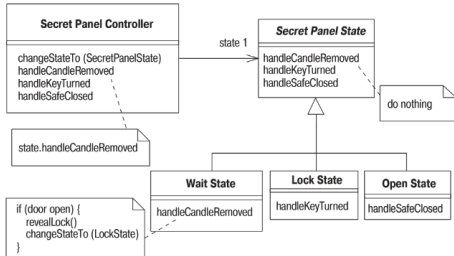
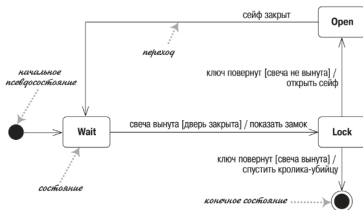
# Таблица состояний



Исходное состояние	Целевое состояние	Событие	Защита	Процедура
Wait	Lock	Candle removed (свеча удалена)	Door open (дверца открыта)	Reveal lock (показать замок)
Lock	Open	Key turned (ключ повернут)	Candle in (свеча на месте)	Open safe (открыть сейф)
Lock	Final	Key turned (ключ повернут)	Candle out (свеча удалена)	Release killer rabbit (освободить убийцу-кролика)
Open	Wait	Safe closed (сейф закрыт)		

© М. Фаулер, UML. Основы

# Паттерн “Состояние”

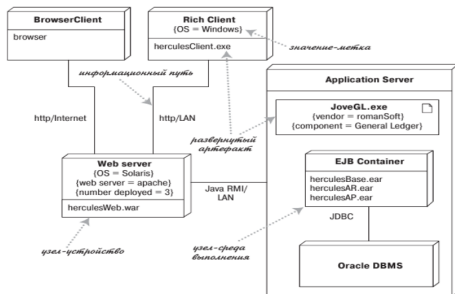


© М. Фаулер, UML. Основы

# Диаграммы развёртывания

## Deployment diagrams

- ▶ Показывает отображение компонентов и физических артефактов на реальные (или виртуальные) устройства
- ▶ Бывает полезна на начальных этапах проектирования, даже до диаграмм компонентов



© М. Фаулер, UML. Основы

# Примеры CASE-инструментов

- ▶ “Рисовалки”
  - ▶ Visio
  - ▶ Dia
  - ▶ SmartDraw
  - ▶ LucidChart
  - ▶ Creately
- ▶ Полноценные CASE-системы
  - ▶ Enterprise Architect
  - ▶ Rational Software Architect
  - ▶ MagicDraw
  - ▶ Visual Paradigm
  - ▶ GenMyModel
- ▶ Забавные штуки
  - ▶ <https://www.websequencediagrams.com/>
  - ▶ <http://yuml.me/>
  - ▶ <http://plantuml.com/>

# Предметно-ориентированные языки

