Диаграммы классов UML

Юрий Литвинов
yurii.litvinov@gmail.com

04.03.2020г

Ликбез по Continuous Integration

Непрерывная интеграция — практика слияния всех изменений по нескольку раз в день, сборки их в известном окружении и запуска юнит-тестов.

- Автоматический билд
  - Всё, что нужно для сборки, есть в репозитории, может быть получено на чистую (ну, практически) машину и собрано одной консольной командой
- Большое количество юнит-тестов, запускаемых автоматически
- Выделенная машина, слушающая репозиторий и выполняющая билд
  - Чаще всего каждый билд запускается на заранее настроенной виртуалке

Continuous Integration

- Извещение всех разработчиков о статусе
  - Если билд не прошёл, разработка приостанавливается до его починки
- Автоматическое выкладывание
- Пока билд не прошёл, задача не считается сделанной
  - Короткие билды (<10 мин.)
  - deployment pipeline
    - Отдельная машина для сборки, для коротких тестов, для длинных тестов, для выкладывания

Travis

- https://travis-ci.org/ — пример бесплатной для open source-проектов облачной CI-системы
- Собирает на чистой виртуальной машине под Ubuntu 12.04, 14.04, 16.04 или OS X 10.13 (есть экспериментальная поддержка Windows)
- Интегрируется с GitHub-ом, Slack-ом, умеет деплоить
- Окружение настраивается конфигурационным файлом или “вручную” из скрипта сборки (некоторые конфигурации разрешают sudo)
- Сборка выполняется либо автоматически, либо указанным скриптом сборки
- Build Matrix
  - Разные конфигурации сборки, выполняемые на разных виртуальных машинах

Travis, настройка сборки

- Установить commit hook на гитхабе
  - Travis умеет это делать сам, надо залогиниться под своим GitHub-аккаунтом на Travis и выбрать нужный репозиторий в профиле
- Добавить .travis.yml в корень репозитория
  - Это конфигурационный файл, говорящий, что и как собирать
- Закоммитить и запушить, это инициирует процесс сборки
  - Коммит, где в комментарии есть подстрока “[ci skip]”, игнорируется Travis-ом, остальные он собирает
- Результаты будут видны у каждого коммита в истории и в пуллреквесте

Travis, конфигурационный файл

language: java

— всё :) Если у вас проект прямо в корне репозитория.
Жизненный цикл сборки:

- Install apt addons
- before_install
- install
- before_script
- script
- after_success или after_failure
- [OPTIONAL] before_deploy
- [OPTIONAL] deploy
- [OPTIONAL] after_deploy
- after_script

Travis, примеры

- Веб-приложение из нескольких сервисов на Java:
  - https://github.com/qreal/wmp/blob/master/.travis.yml
- Относительно большое десктопное приложение на C++:
  - https://github.com/qreal/qreal/blob/master/.travis.yml
- Билд в Docker-окружении (все зависимости носим с собой):
  - https://github.com/trikset/trikRuntime/blob/master/.travis.yml

Travis, пример конфига для домашки

```
language: java
os:
  - linux
env:
  - PROJECT_DIR=hw1
  - PROJECT_DIR=hw2
  - PROJECT_DIR=hw3
script: cd $PROJECT_DIR && ./gradlew check
```

AppVeyor

- CI с поддержкой Windows и Linux, прежде всего для сборки .NET-приложений, но умеет много чего ещё
  - Windows Server 2016 или Windows Server 2012 R2
  - Ubuntu 16.04.4 LTS или Ubuntu 18.04 LTS
- Тоже бесплатен для open source
- Конфигурируется через appveyor.yml
- Собирает по умолчанию MSBuild
  - Можно переубедить
- Синтаксис .yml-файла: https://www.appveyor.com/docs/appveyor-yml/
- Пример: https://github.com/qreal/qreal/blob/master/appveyor.yml
  - Собирать двумя CI-серверами один проект не зазорно

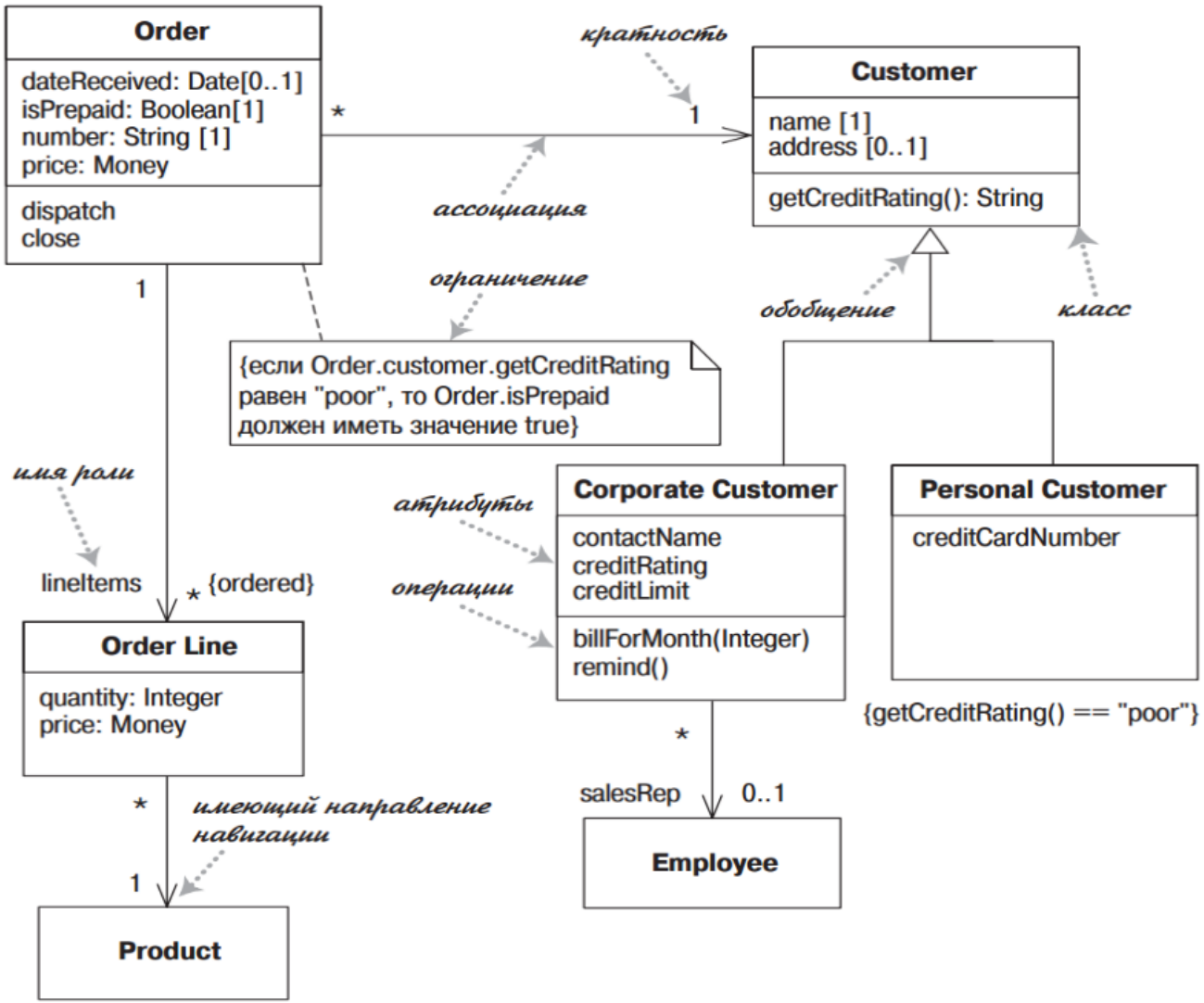
Computer-Aided Software Engineering

- В 80-е годы термином CASE называли всё, что помогает разрабатывать ПО с помощью компьютера
  - Даже текстовые редакторы
- Теперь — прежде всего средства для визуального моделирования (UML-диаграммы, ER-диаграммы и т.д.)
- Отличаются от графических редакторов тем, что “понимают”, что в них рисуют
- Нынче чаще используются термины “MDE tool”, “UML tool” и т.д.
- Репозиторий + набор визуальных редакторов + генераторы + средства обратного проектирования + иногда много чего ещё

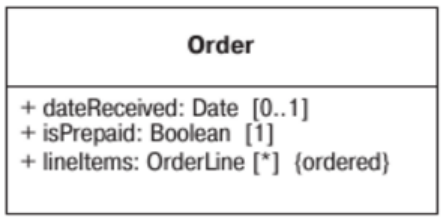
Примеры CASE-инструментов

- “Рисовалки”
  - Visio
  - Dia
  - SmartDraw
  - 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/

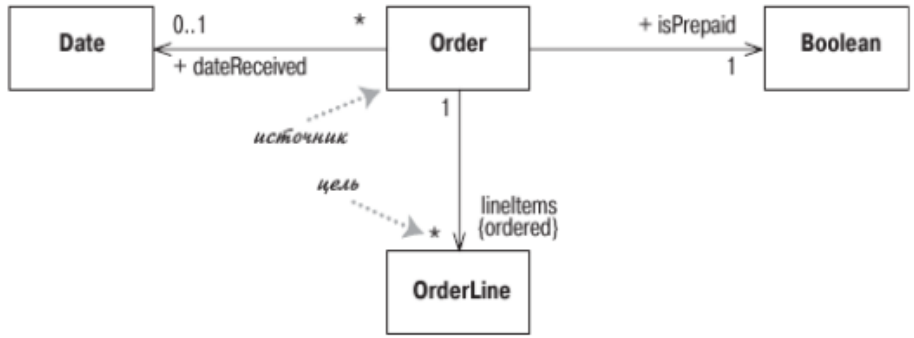
Диаграммы классов UML



Свойства



Атрибуты



Ассоциации

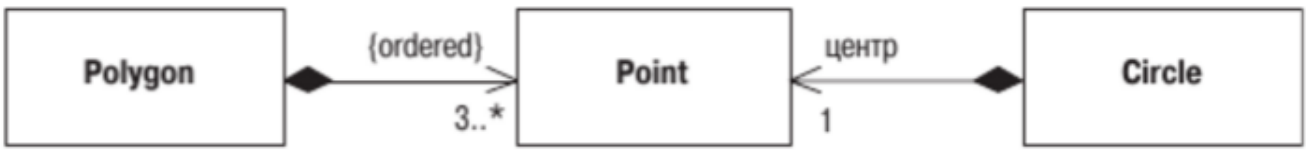
Синтаксис:

- 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)

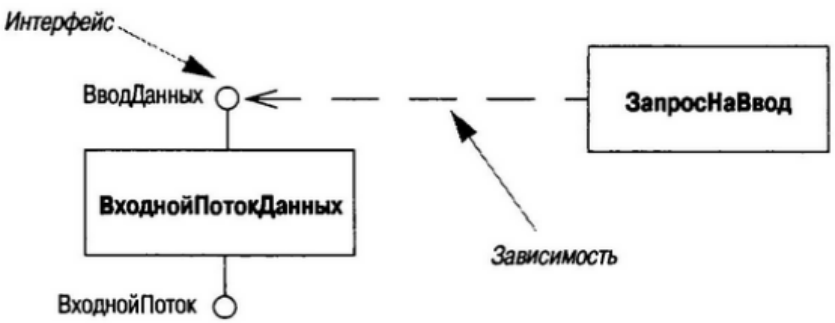
Композиция — объект владеет другим объектом (управляет его временем жизни, хранит его по значению или по указателю, делая delete)



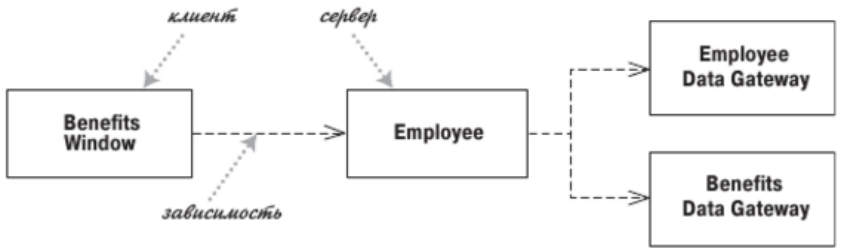
Уточнение обычной ассоциации, используется только если очень надо

Прочее

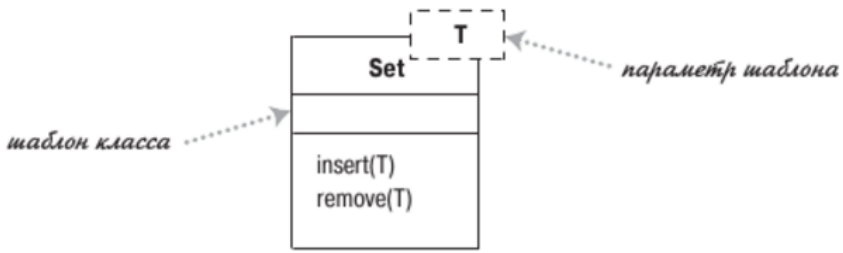
Интерфейсы



Зависимости

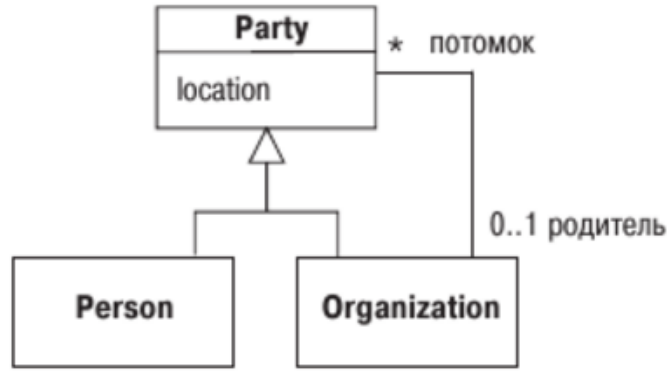


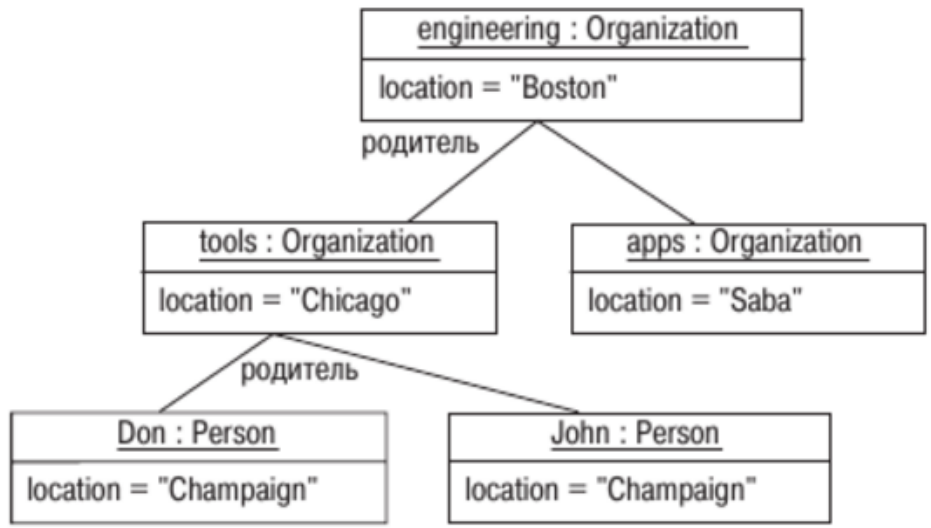
Шаблоны



Диаграммы объектов

- snapshot структуры классов во время выполнения
- Используются обычно чтобы пояснить диаграмму классов
- Полезны на этапе анализа предметной области, ещё до диаграмм классов





Домашнее задание: cd, ls

- Реализовать команды **ls** и **cd** на базе кода одногруппника
  - Обе команды могут принимать 0 или 1 аргумент
  - Не забывайте про юнит-тесты
- Написать ревью на архитектуру оного одногруппника, указав, что оказалось удобным, а что неудобным при реализации, что можно было бы улучшить
- Сделать fork на GitHub, выложить изменения туда и сделать пуллреквест в свой форк
  - Если “жертва” не против, можно и в исходный репозиторий
- Реализация, в которой надо сделать команды, определяется циклическим сдвигом на **2** вверх по списку на HwProj, с пропуском несданных решений
- Дедлайн: **10:00 18.03.2020г**.

Задание на остаток пары

- Нарисовать диаграмму классов UML для своего решения CLI, как оно есть
- Обращать внимание на синтаксис UML и читаемость диаграммы
- Как будет готово, позвать меня и показать
- Не пытаться рисовать методы, кроме самых важных
- Не рисовать все поля, можно даже не рисовать все классы — надо успеть до конца пары