Domain-Driven Design

Юрий Литвинов
yurii.litvinov@gmail.com

22.03.2018г

Domain-Driven Design, напоминание

- Архитектура приложения строится вокруг **Модели предметной области**
- Модель определяет **Единый язык**, на котором общаются и разработчики, и эксперты, описывая естественными фразами то, что происходит и в программе, и в реальности
- Модель — это не только диаграммы, это ещё (и прежде всего) код, и устное общение
- Модель строится в процессе непрерывной **Переработки знаний**
  - Рефакторинг — неотъемлемая часть процесса

Книжка, по которой рассказ

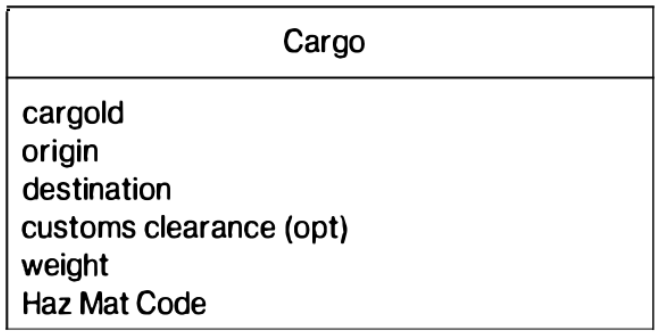
И откуда картинки

Эрик Эванс, “Предметно-ориентированное проектирование. Структуризация сложных программных систем”. М., “Вильямс”, 2010, 448 стр.

Единый язык

- У программистов и специалистов предметной области свой профессиональный жаргон
- Свои жаргоны появляются даже среди групп разработчиков в одном проекте
- Необходимость перевода размывает смысл понятий
- “Еретики” используют понятия в разных смыслах
- Единый язык — понятия из модели (классы, методы), паттерны, элементы “высокоуровневой” структуры системы (которая не отражается в коде)
- Изменения в языке — рефакторинг кода
- Языков в проекте может быть много

Без единого языка



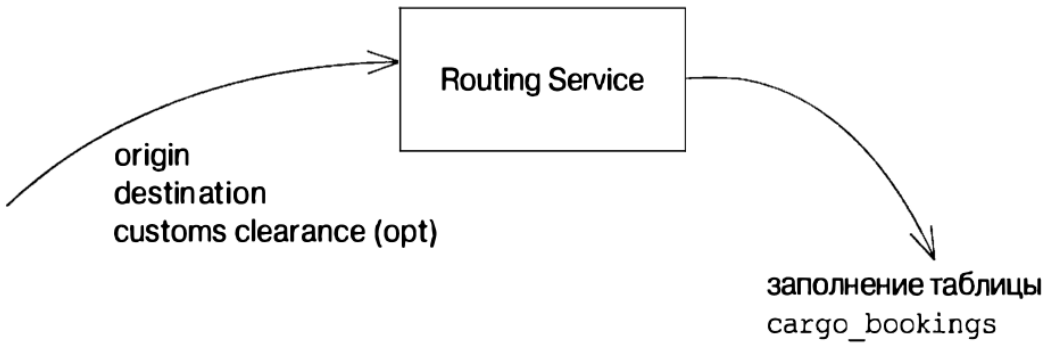
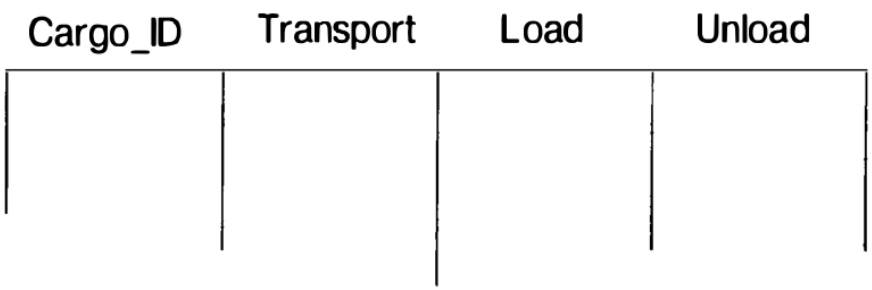
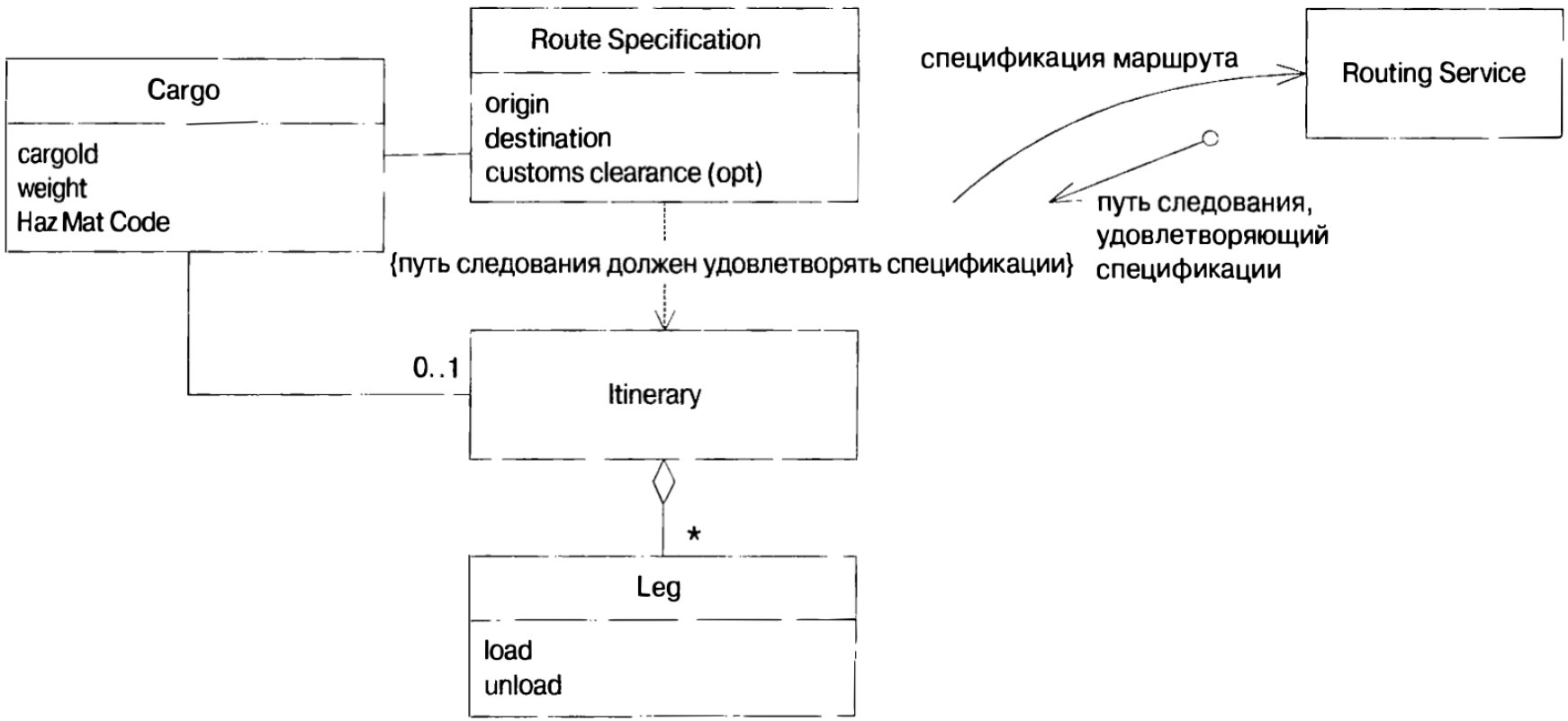


Таблица БД: `cargo_bookings`



С единым языком



“Моделирование вслух”

*Если передать в **Маршрутизатор** пункт отправки, пункт назначения, время прибытия, то он найдет нужные остановки в пути следования груза, а потом, ну... запишет их в базу данных.*

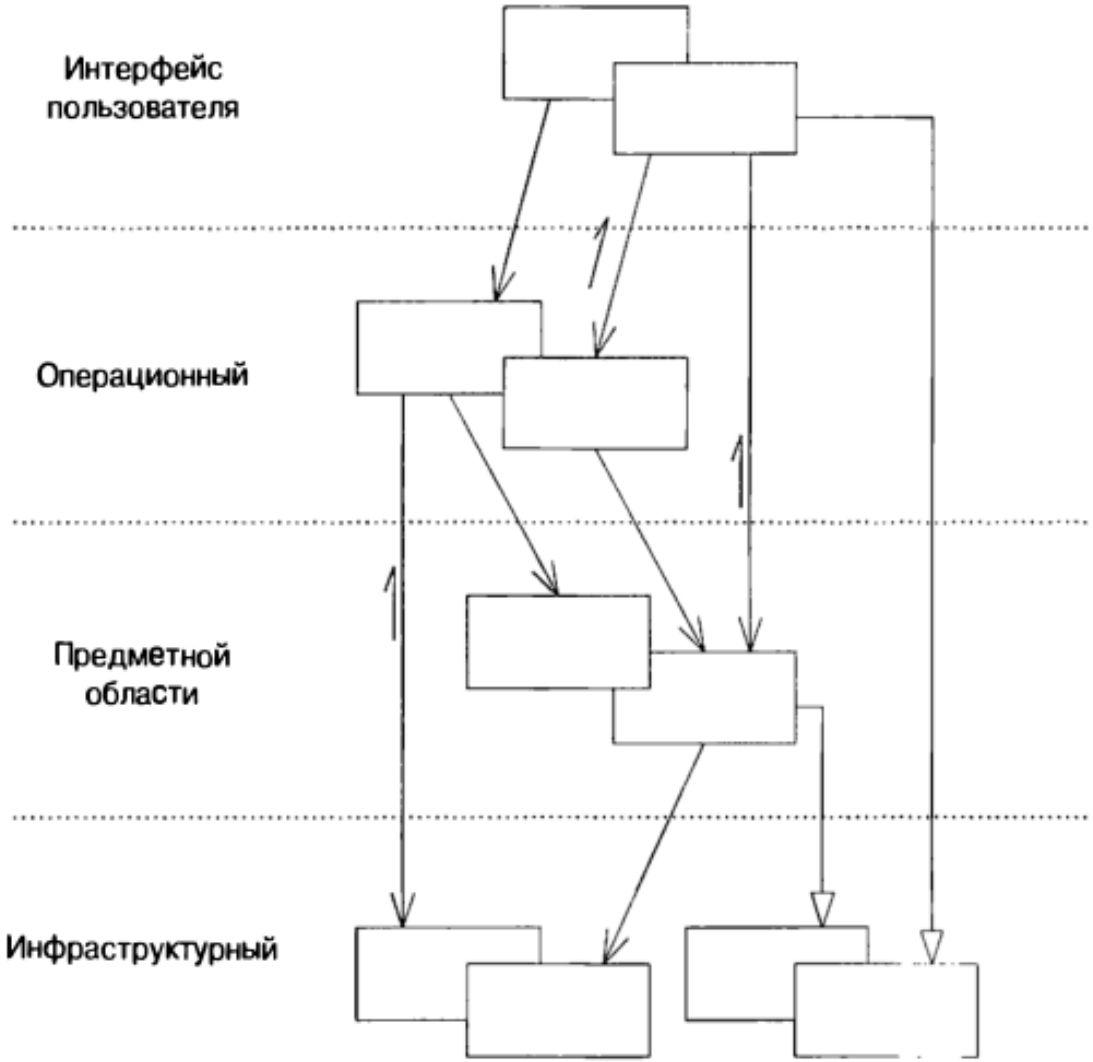
*Пункт отправки, пункт назначения и все такое... все это идет в **Маршрутизатор**, а оттуда получаем **Маршрут**, в котором записано все, что нужно.*

***Маршрутизатор** находит **Маршрут**, удовлетворяющий **Спецификации маршрута**.*

Модель и реализация

- Модель, не соответствующая коду, бесполезна
- Код, созданный без модели, скорее всего, работает неправильно
  - “Разрушительный рефакторинг”
  - Нельзя разделять моделировщиков и программистов
- Модель в DDD выполняет роль и модели анализа, и модели проектирования одновременно
  - Это требует баланса между техническими деталями и адекватностью выражения предметной области
  - Часто требуется несколько итераций рефакторинга
- Язык программирования должен поддерживать парадигму модели
- Модель, привязанная к реализации, хороша и для пользователя

Изоляция предметной области



Изоляция предметной области, соображения

- Модель предметной области должна быть отделена от остальной программы
- Классы модели умеют делать только “суть”
- Сборка всего воедино и общее управление процессом — на операционный уровень
  - Бизнес-регламенты — на уровне модели предметной области
- Все технические вещи — на инфраструктурный уровень
  - Работа с БД
  - Middleware, сетевые коммуникации
  - Утилиты
  - Абстрактные базовые классы
- Observer или вариации MVC для связей “снизу вверх”

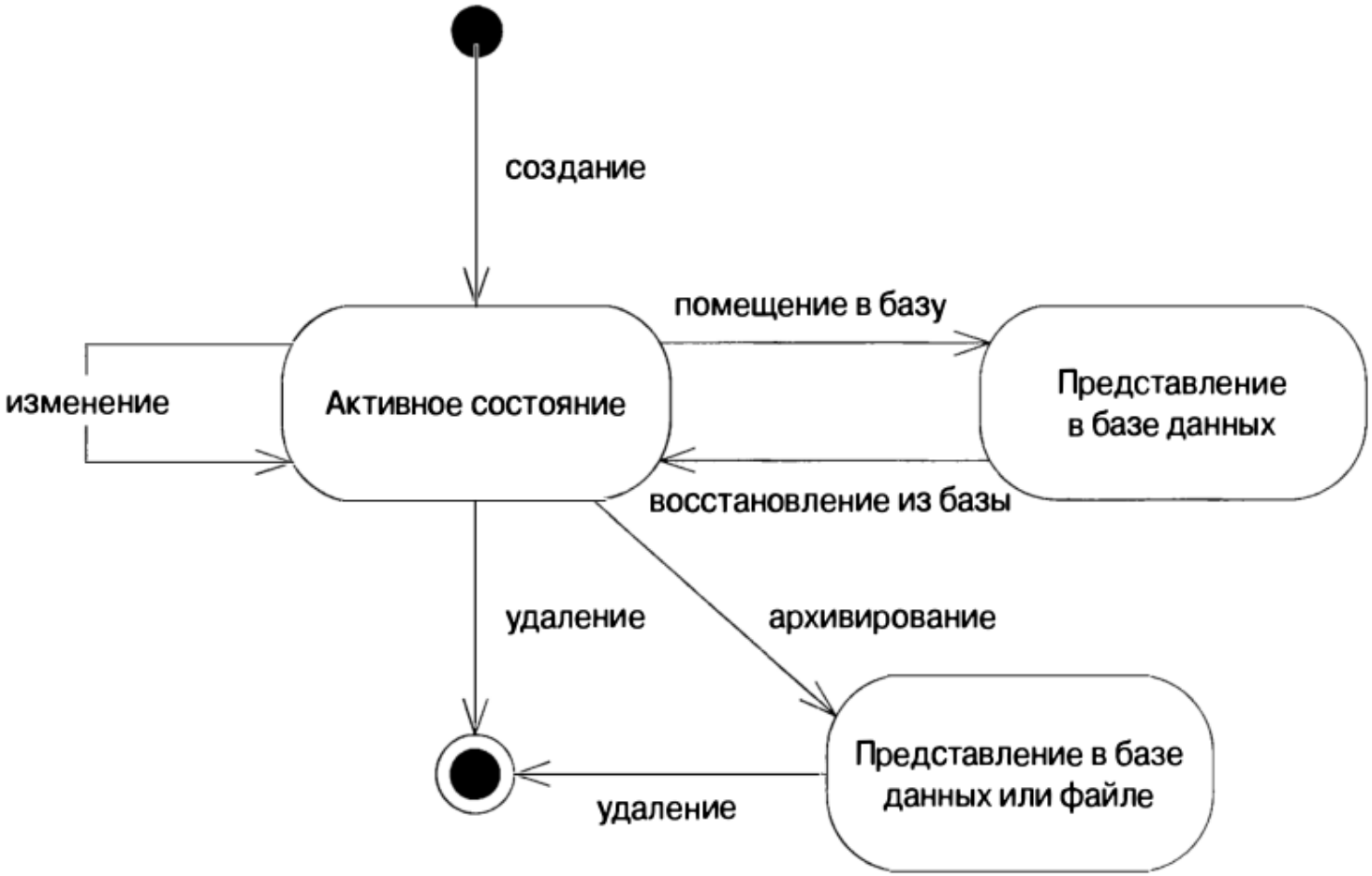
Антипаттерн “Умный GUI”

- А давайте всю бизнес-логику писать прямо в обработчиках на форме
- Код GUI напрямую работает с БД
- Делает невозможным проектирование по модели
- Не всегда плохо
  - Применимы средства быстрой разработки приложений
  - Прирост производительности на начальных этапах
  - Легко приделывать новые фичи и переписывать старые
- Не всегда хорошо
  - Очень сложно переиспользование
  - Сложно реализовать сложное поведение (зато легко простое)
  - Сложно интегрироваться

Основные структурные элементы модели

- **Ассоциации** — чем проще, тем лучше
- **Сущность (Entity)** — объект, обладающий собственной идентичностью
  - Нужна операция идентификации
  - Нужен способ поддержания идентичности
- **Объект-значение (Value object)** — объект, полностью определяемый своими атрибутами
  - “Лучше”, чем сущность
  - Как правило, немутабельны
  - Могут быть разделяемыми
- **Служба (Service)** — объект, представляющий операцию
  - Как правило, не имеет собственного состояния
  - Операции нет естественного места в других классах модели
- **Модуль (Module)** — смысловые части модели

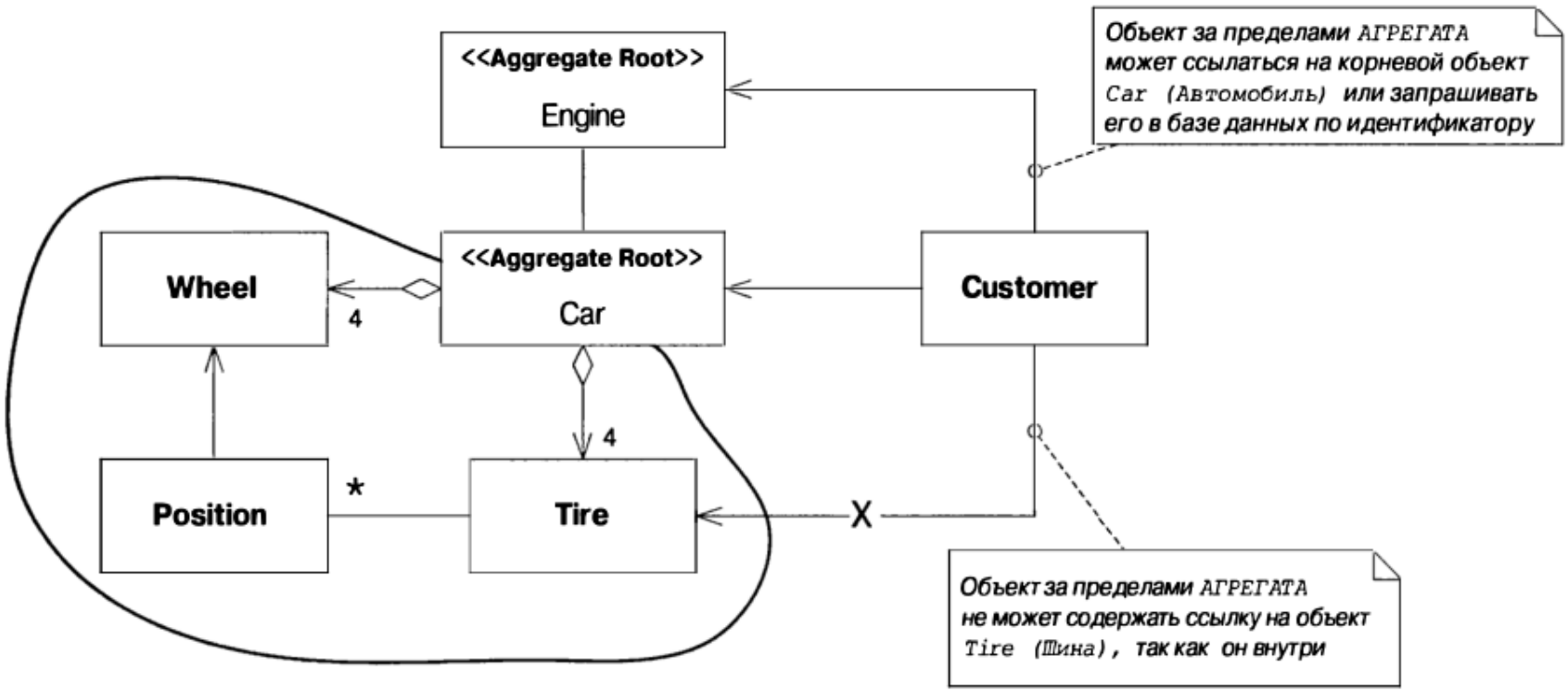
Жизненный цикл объекта



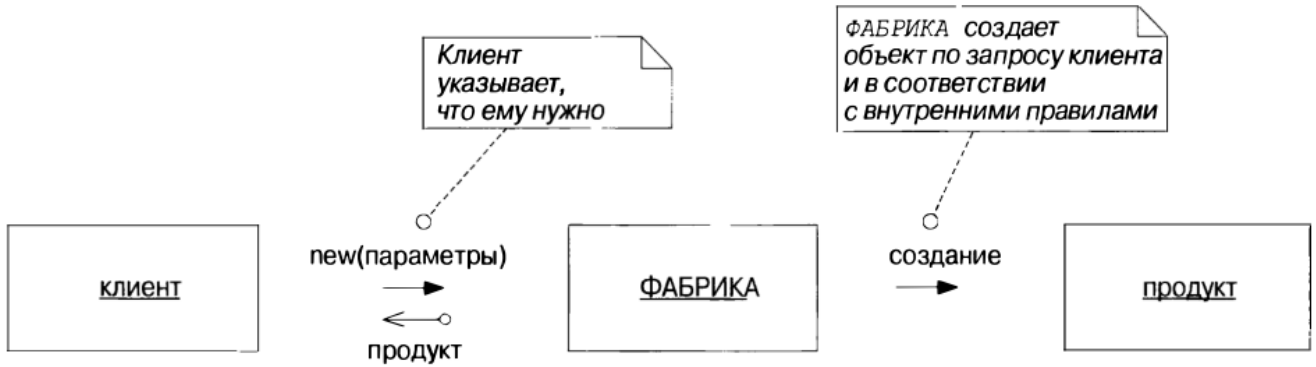
Агрегаты

- **Агрегат** — изолированный кусок модели, имеющий **корень** и **границу**
- Корень — глобально идентичный объект-сущность
- Остальные объекты в агрегате идентичны локально
- Извне агрегата можно хранить ссылку только на корень
  - Отдавать временную ссылку можно
- Корень отвечает за поддержание инвариантов всего агрегата

Агрегат, пример



Фабрика

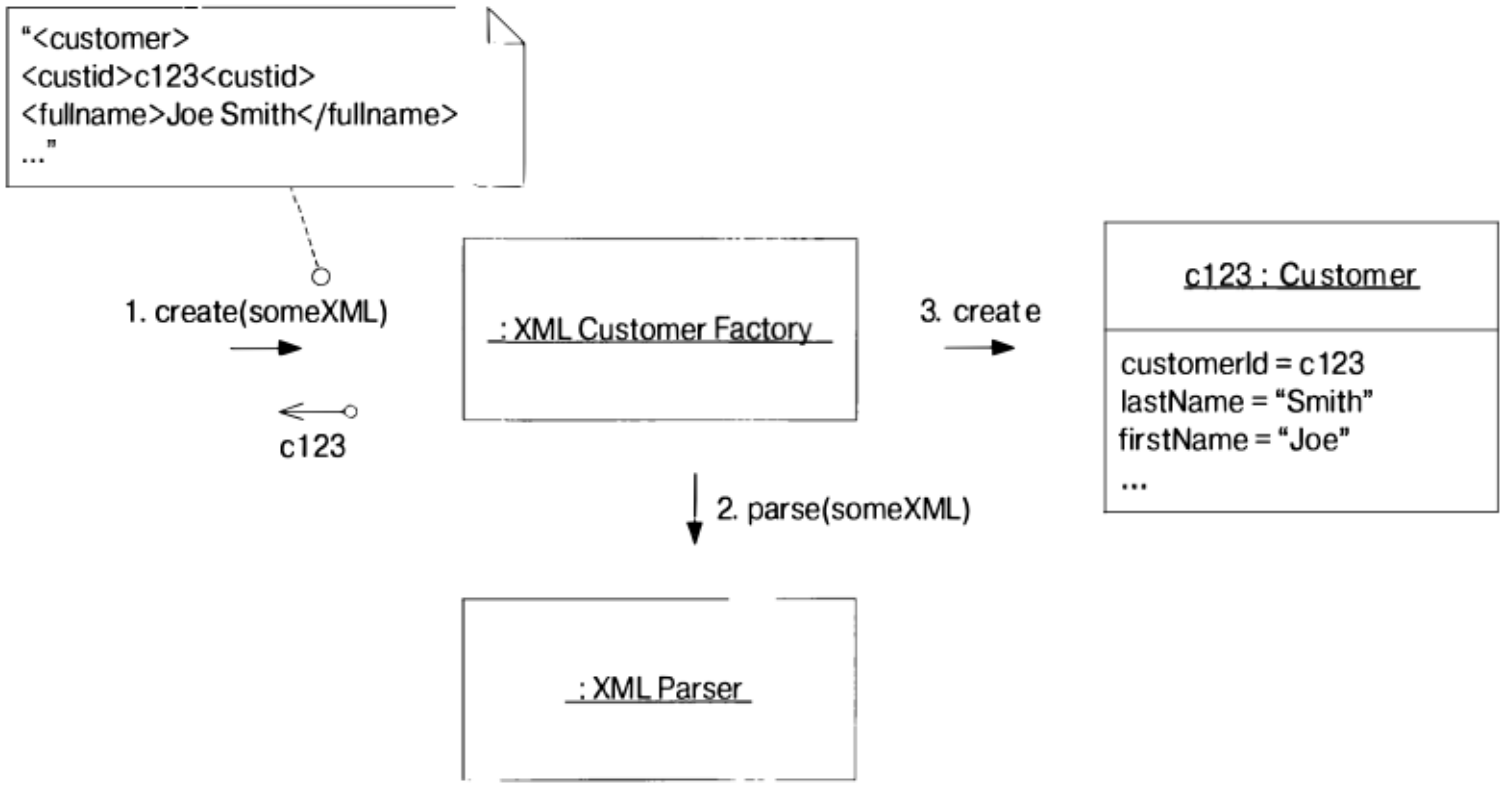


Фабрика служит для создания объектов или агрегатов

- Скрывает внутреннее устройство конструируемого объекта
  - Операция создания “атомарна” и обеспечивает инварианты
- Изолирует сложную операцию создания
- Как правило, не имеет бизнес-смысла, но является частью модели
- Реализуется аж несколькими разными паттернами

Пример

Фабрика, использующаяся для восстановления объекта



Хранилище (Repository)

Репозиторий хранит объекты и предоставляет к ним доступ

- Может инкапсулировать запросы к БД
- Может использовать фабрики
- Может обладать развитым интерфейсом запросов