Оглавление

[1. Дайте определения непрерывной интеграции, непрерывной доставки и непрерывного развёртывания. Какую роль в этих процессах играет автоматическое тестирование? 2](#_Toc164631933)

[2. Дайте определение изоляции предметной области. Зачем нужна изоляция предметной области? Как достигается? 5](#_Toc164631934)

[3. Дайте определение сущности, объекта значения, службы и агрегата в рамках предметно-ориентированного проектирования. Приведите пример. 5](#_Toc164631935)

[4. Перечислите слои многоуровневой архитектуры, дайте краткое описание каждого слоя, а также поясните какие задачи решает разделение на слои. 7](#_Toc164631936)

[5. Что такое предметно-ориентированное проектирование? Приведите основные понятия. Назовите основные преимущества и недостатки. 7](#_Toc164631937)

[6. Что такое взаимодействие контекстов в DDD? Перечислите основные варианты взаимодействия. Что такое карта контекстов? 7](#_Toc164631938)

[7. Дайте определение гибкой архитектуры. Назовите основные правила и приемы. 7](#_Toc164631939)

[8. Дайте определение проектирования по модели, опишите особенности этого подхода. Приведите последовательность формирования модели, кратко опишите каждый этап. 7](#_Toc164631940)

[9. Дайте определение Event Storming. Опишите основные характеристики и этапы. Приведите пример. 8](#_Toc164631941)

[10. Дайте определение микросервиса. Опишите основные преимущества и недостатки микросервисной архитектуры. 10](#_Toc164631942)

[11. Что такое микросервисная архитектура? Приведите пример нескольких проблемных сторон микросервисной архитектуры и предлагаемые варианты решений. 10](#_Toc164631943)

[12. Дайте определение микросервиса. Назовите отличия микросервисной архитектуры от сервисноориентированной. 10](#_Toc164631944)

[13. Назовите основные преимущества и недостатки синхронного межпроцессного взаимодействия. Назовите приёмы, применяемые для компенсации этих недостатков. 10](#_Toc164631945)

[14. Дайте определение основных видов сообщений, канала сообщений при асинхронном взаимодействии. Опишите роль брокера сообщений. 10](#_Toc164631946)

[15. Приведите формулировку закона Конвея, обратного манёвра Конвея. Объясните как эти идеи используются в микросервисной архитектуре. 10](#_Toc164631947)

[16. Почему возникают дубликаты сообщений? Какие приёмы используются для работы с дубликатами сообщений? Опишите их. 10](#_Toc164631948)

[17. Опишите основные приёмы и особенности использования повествований (saga). Приведите пример. 10](#_Toc164631949)

[18. Опишите принципы построения шестигранной архитектуры. Приведите пример реализации микросервиса в этой архитектуре. 10](#_Toc164631950)

[19. Опишите основные принципы построения бизнес-логики на основе порождения событий. Укажите основные преимущества и недостатки этого подхода. 13](#_Toc164631951)

[20. Опишите шаблоны реализации запросов. Дайте определения, приведите примеры. 13](#_Toc164631952)

[21. Почему в микросервисной архитектуре нежелательно использовать распределённые транзакции? Какие используются приёмы для достижения согласованности данных? Опишите их. 13](#_Toc164631953)

## 1. Дайте определения непрерывной интеграции, непрерывной доставки и непрерывного развёртывания. Какую роль в этих процессах играет автоматическое тестирование?

Непрерывная интеграция (Continuous Integration) — когда продукт регулярно (несколько раз в день) собирается из исходного кода и для него запускается существенная часть автоматических тестов, например все модульные тесты. Если автоматические тесты работают долго, то их можно запускать реже, например раз в сутки. Стандартный подход для организации непрерывной интеграции — это запустить TeamCity или Jenkins, которые будут загружать исходный код из системы контроля версий, собирать его и запускать тесты. Другие известные решения: Travis CI, GitLab, Space, GitHub, BitBucket.

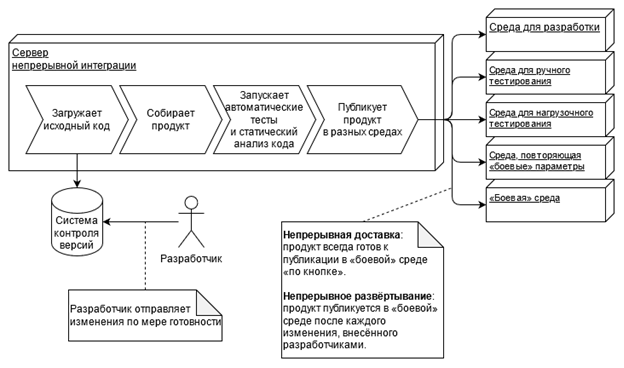
* **Непрерывная интеграция (CI)** — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.
* В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ.
* Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.

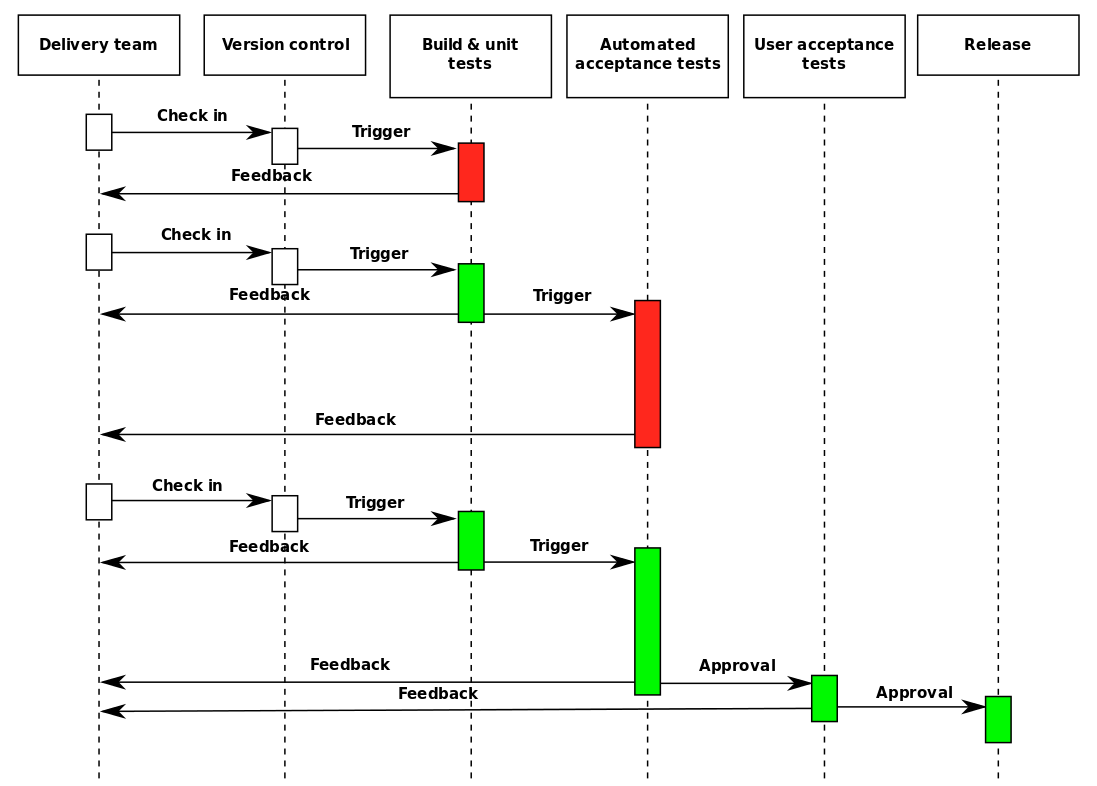
Непрерывная доставка (Continuous Delivery) — продукт всегда находится в «собранном» состоянии и готов к передаче в промышленную эксплуатацию, даже с учетом последних изменений, внесенных разработчиками в код. Скорее всего, непрерывная доставка будет реализована на основе тех же технологий, что и непрерывная интеграция. В этой статье на «Хабре» приводятся еще несколько подходящих открытых инструментов. В другой работе подробно рассматривается разница между реализацией конвейера сборки в облаке и «на земле».

* Подход к разработке программного обеспечения, при котором программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время.
* Передача его происходит вручную
* Подход позволяет уменьшить стоимость, время и риски внесения изменений путём более частных мелких обновлений в продакшн-приложение.
* Автоматическое обновление тест-сервера доступно в CI.
* **Непрерывная доставка** (англ. Continuous delivery или CD, или CDE) — это подход к разработке программного обеспечения, при котором программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время.
* Передача его происходит вручную.
* Подход позволяет уменьшить стоимость, время и риски внесения изменений путём более частных мелких обновлений в продакшн-приложение.

Непрерывное развертывание (Continuous Deployment) — когда обновления продукта регулярно (например, после каждого изменения, внесенного разработчиками) вводятся в промышленную эксплуатацию, и при этом все процессы предметной области продолжают работать без сбоя. В некотором роде непрерывное развертывание — это хорошо автоматизированная и часто выполняющаяся непрерывная доставка.

* Подход к разработке программного обеспечения, при котором функциональные возможности программного обеспечения часто предоставляются посредством автоматизированного развертывания
* (в отличие от Continuous Delivery)
* **Непрерывное развертывание** (англ. Continuous deployment или CD) — это подход к разработке программного обеспечения, при котором все изменения, вносимые в исходный код, автоматически развертываются в продакшен, без явной отмашки от разработчика. Как правило, задача разработчика сводится к проверке запроса на включение (pull request) от коллеги и к информированию команды о результатах всех важных событий.
* Непрерывное развертывание требует, чтобы в команде существовала отлаженная культура мониторинга, все умели держать руку на пульсе и быстро восстанавливать систему.





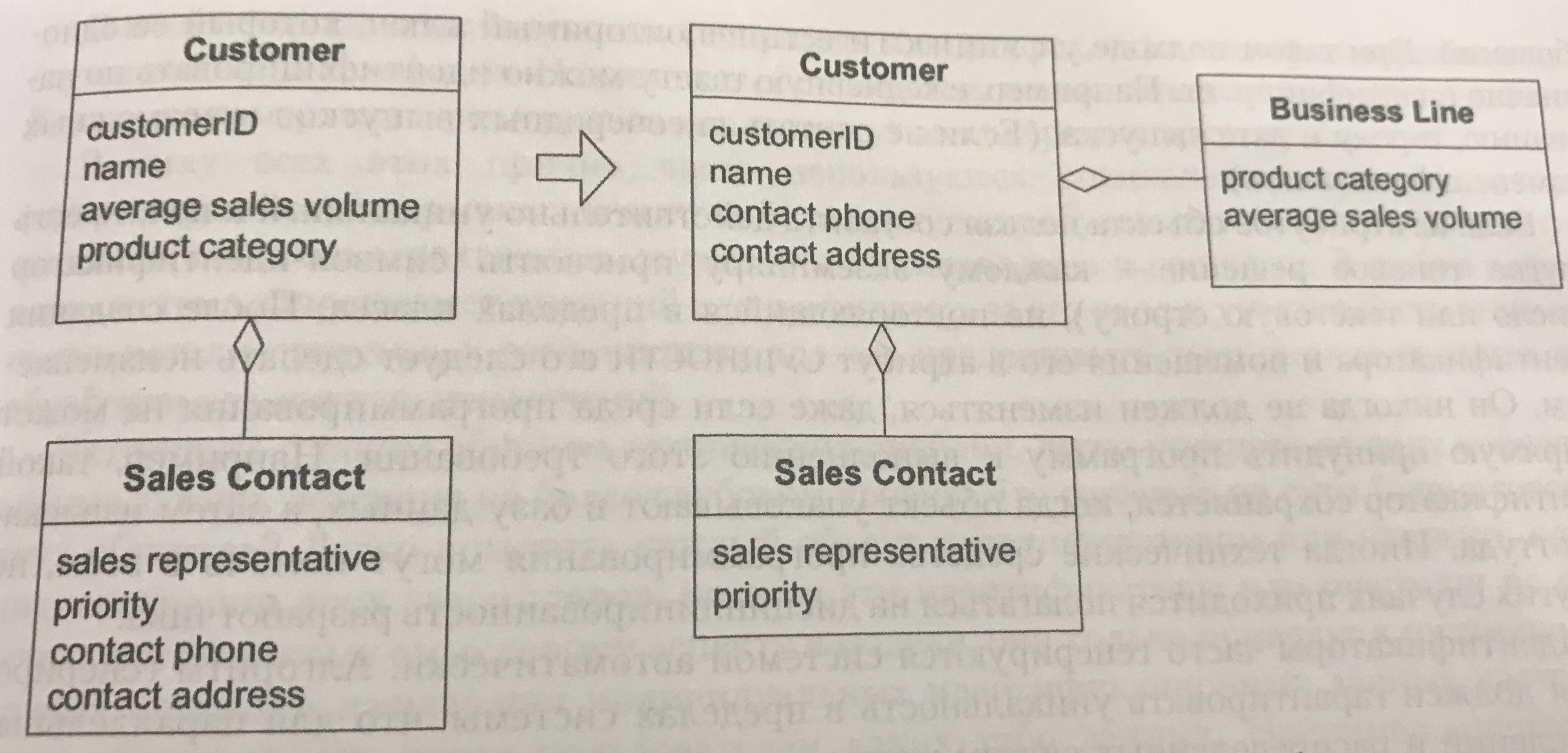
## 2. Дайте определение изоляции предметной области. Зачем нужна изоляция предметной области? Как достигается?

## 3. Дайте определение сущности, объекта значения, службы и агрегата в рамках предметно-ориентированного проектирования. Приведите пример.

* **Сущность** — объект (понятие) предметной области, характеризующиеся *непрерывностью* и *индивидуальностью* (уникальностью) существования.
* *Непрерывность* — модель должна гарантировать сохранение состояния объекта на протяжении времени его существования.
* *Индивидуальность* — в модели должно даваться точное определение, что такое одинаковые объекты.

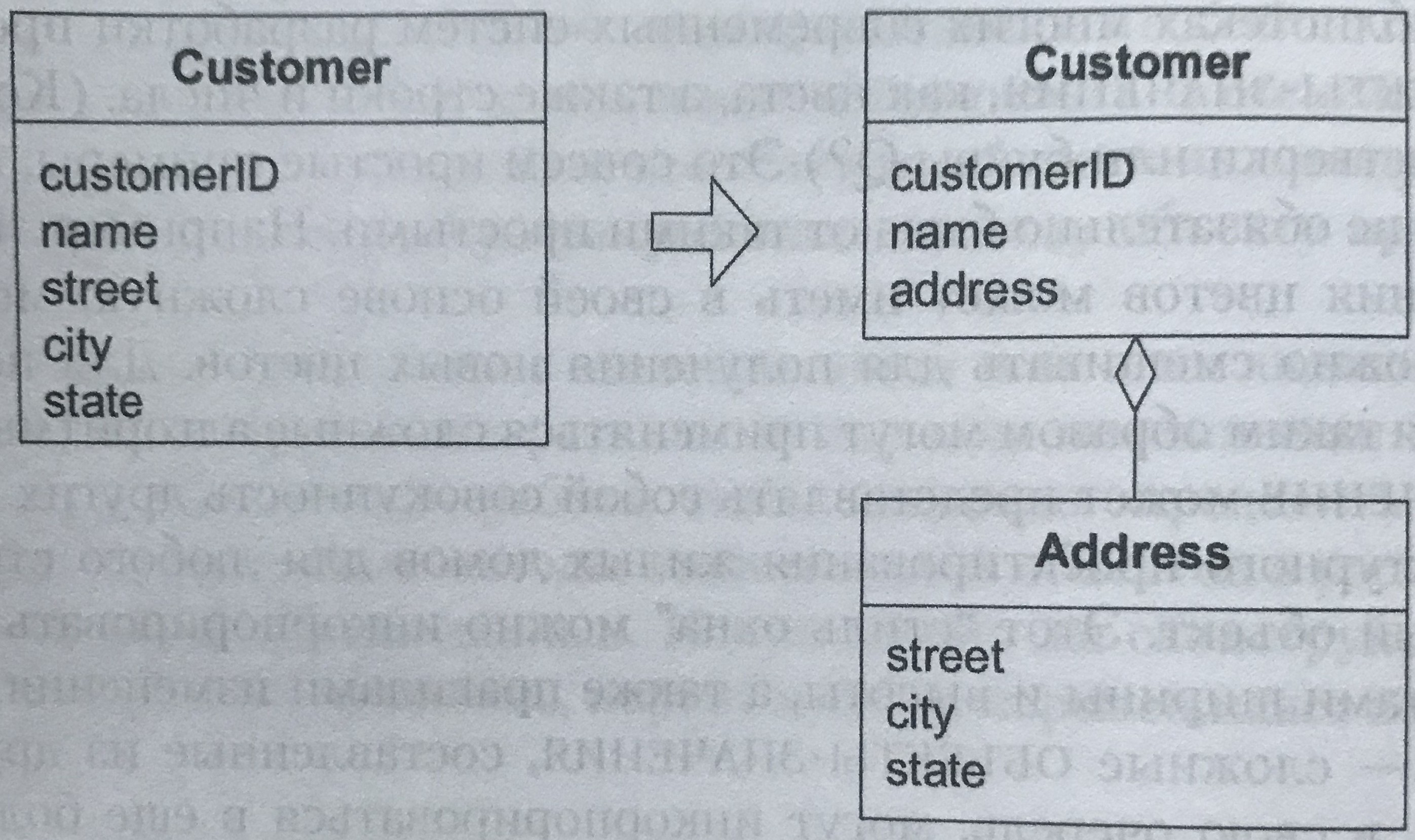
Свойства сущностей

* Понятие предметной области:
  + атрибуты,
  + поведение.
* Уникальность:
  + ключ,
  + генерируемый номер,
  + GUID (Globally Unique Identifier):
    - 6F9619FF-8B86-D011-B42D-00CF4FC964FF.
* Непрерывность



Объекты-значения

* Объект, который представляет описательный аспект предметной области и не имеет индивидуального существования, собственной идентичности
* Полностью определяются своими атрибутами
* Очень схожи с неизменяемыми объектами



Службы

* Операция или группа операций, предлагаемая в модели в виде обособленного интерфейса, который в отличие от сущности или объекта-значения не имеет никакого состояния
* Роль служб в изоляции предметной области
* Ограничения доступа
* **Агрегат** — совокупность взаимосвязанных объектов, которые мы воспринимаем как единое целое с точки зрения изменения данных.
* Понятие агрегата очень важно в проектировании по модели, так как выделяет из набора понятий ***корневую*** (главную) сущность, которая позволяет:
  + управлять ***транзакционностью***,
  + контролем ***инварианта*** агрегата,
  + выполнять ***создание*** целостной совокупности объектов и
  + ***хранением*** её состояния.
* Доступ к агрегату должен осуществляться только через корневую сущность.

Правила работы с агрегатом

* Корневой объект-сущность имеет глобальную идентичность и несёт полную ответственность за проверку инвариантов
* Не коренные объекты-сущности имеют локальную идентичность – они уникальны только в границах агрегата
* Нигде за пределами агрегата не может храниться ссылка на что-либо внутри него (только временные ссылки и значения)
* Значит, только корневые объекты агрегатов можно непосредственно получать по запросам из БД. Все остальные объекты разрешается извлекать по цепочке связей
* Объекты внутри агрегата могут хранить ссылки на корневые объекты других агрегатов
* Операция удаления должна ликвидировать всё, что находится в границах агрегата
* Как только вносится изменение в любой объект агрегата, следует сразу удовлетворять все инварианты этого агрегата

## 4. Перечислите слои многоуровневой архитектуры, дайте краткое описание каждого слоя, а также поясните какие задачи решает разделение на слои.

## 5. Что такое предметно-ориентированное проектирование? Приведите основные понятия. Назовите основные преимущества и недостатки.

## 6. Что такое взаимодействие контекстов в DDD? Перечислите основные варианты взаимодействия. Что такое карта контекстов?

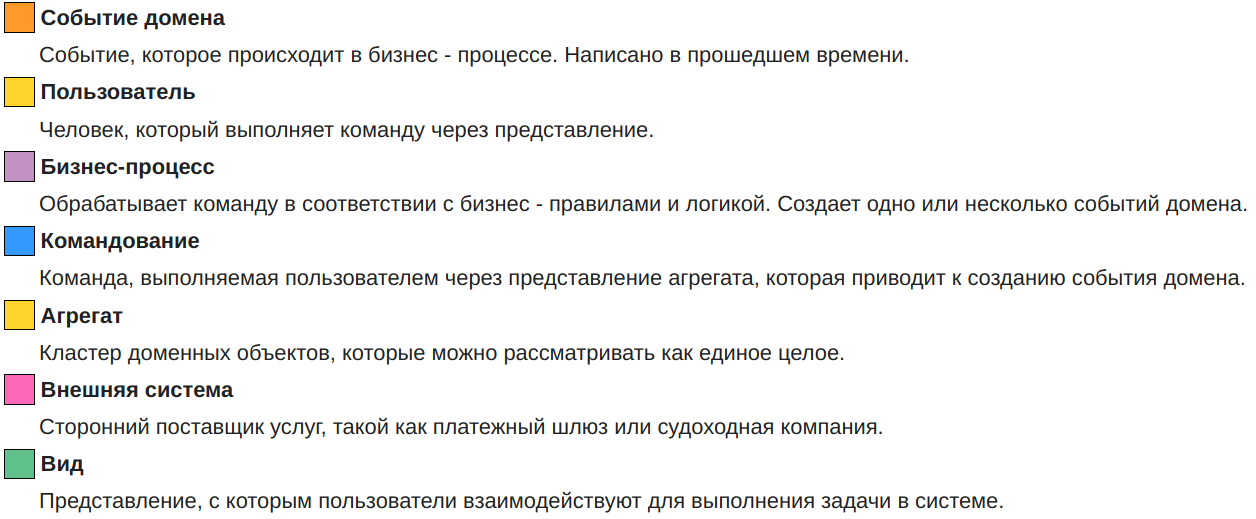
## 7. Дайте определение гибкой архитектуры. Назовите основные правила и приемы.

## 8. Дайте определение проектирования по модели, опишите особенности этого подхода. Приведите последовательность формирования модели, кратко опишите каждый этап.

* **Проектирование по модели** (model-driven design, **MDD**) — проектирование структуры объектов программы, при котором соблюдается максимально точное соответствие между некоторым подмножеством элементов программы и элементами модели
* или, иными словами, процесс совместной разработки модели и её программной реализации с сохранением такого соответствия между ними
* т.е. формирование / модификация модели = формирование / модификация кода
* Вместо проектной документации упор делается на:
  + понимании того факта, что модель не может точно соответствовать предметной области (модель отражает предметную область в кривом зеркале нашего понимания);
  + совместном обсуждении предметной области на её языке и формировании модели в виде специальных диаграмм (временных);
  + уточняющем рефакторинге, подготавливающем изменения;
  + обеспечении максимального соответствия релиза принятой модели;
  + соблюдении принципа самодокументируемости кода (код и есть документация);
  + использовании инструментов непрерывной интеграции;
  + использовании инструментов непрерывного развёртывания.

## 9. Дайте определение Event Storming. Опишите основные характеристики и этапы. Приведите пример.

* **Event Storming** — это метод, основанный на сессионных встречах (семинарах) представителей заказчика и команды разработки, позволяющий быстро объяснить, что происходит в предметной области для проектирования её модели.
* Отличается лёгкостью в освоении и намеренно не использует компьютерного документирования (впрочем, как и всё в DDD).
* Результат выражается в наклеивании стикеров на длинную стену.
* Бизнес-процесс оформляется в виде серии событий предметной области, которые обозначаются как оранжевые стикеры.
* Event Storming может быть использован в качестве средства для моделирования бизнес-процессов и разработки требований.
* Основная идея состоит в том, чтобы объединить разработчиков программного обеспечения и экспертов в предметной области и учиться друг у друга.
* Чтобы облегчить этот процесс обучения, Event Storming должен быть увлекательным.
* Модель будет размещена на широкой стене с раскатанным на ней рулоном бумаги. Стикеры будут размещены на этой бумаге. Вам потребуется по крайней мере 5 различных цветов для стикеров.
* Выбрать интересуемые события предметной области и записать их на оранжевых стикерах.
* Когда все события предметной области выявлены, вторым шагом является определение команды, которая вызвала каждое из событий. Команды записываются на синих заметках и помещаются непосредственно перед соответствующим событием.
* На третьем этапе определяются агрегаты, в пределах которых выполняются команды и где происходят события. Агрегаты написаны желтыми стикерами.
* Информация о предметной области, собранная во время сессии Event Storming, делится на несколько категорий, каждая из которых имеет свой собственный цвет стикера:



* Клиент инициирует выполнение команды. Команда всегда формулируется в будущем времени: это запрос на выполнение действия. Оно еще не выполнено и может быть отменено.
* Допустим, речь идет о том, чтобы добавить товар в корзину. Команда приходит в агрегат (термин из DDD), либо уходит во внешнюю систему. Если команда может быть выполнена без нарушения инварианта агрегата, создается событие. Событие — это факт и действие, произошедшие в прошлом. В отличие от команды, событие не может быть отменено («Товар добавлен в корзину»).
* На основе событий, например, обновляется модель чтения, цель которой — помочь пользователю в принятии решений («Купи еще на 1000 рублей и получи скидку 10%») и выполнении последующих команд. Или же событие может активировать некое бизнес-правило. Которое, в свою очередь, вызовет команду, и цикл повторится.
* Определяются абсолютно все события, происходящие в предметной области. Их могут быть сотни, и даже тысячи.
* Примеры событий: товар добавлен в корзину; товар оплачен; доставка оформлена.
* К этим событиям приписываем команды. Все очень просто. Товар добавлен в корзину. Команда, которая относится к этому событию: «Добавить товар в корзину».
* Постепенно идет усложнение и появляются определенные бизнес-правила. Здесь очень хорошо помогают именно представители бизнеса, потому что программисты, по сути, переводят эти бизнес-правила в код. И важно здесь не то, как технически это выглядит, а то, как продукт работает в реальном бизнесе.
* Появляются агрегаты. В нотации Event Storming агрегаты обозначаются желтыми стикерами. Вначале мы можем оставить эти стикеры пустыми, без названия. Мы пока не знаем, что это за агрегат. Просто будем собирать вокруг него те команды и события, которые, как нам кажется, должны обрабатываться вместе. А уже потом назовем его.
* Агрегаты сбиваются вместе и образуют то, что в терминах DDD называется ограниченными контекстами. Они представляют из себя модель + общий словарь.
* Над этой схемой работали около пяти часов. Вначале выявили события, затем команды, добавили бизнес-правила. Со временем результатом командной работы стал сбор некоторых команд и событий в контексты. Например, контекст работы с ботами, контекст диалога с клиентом, контекст закрытия заявки.

ЛЕКЦИЯ 1-04 EVENT STORMING

## 10. Дайте определение микросервиса. Опишите основные преимущества и недостатки микросервисной архитектуры.

## 11. Что такое микросервисная архитектура? Приведите пример нескольких проблемных сторон микросервисной архитектуры и предлагаемые варианты решений.

## 12. Дайте определение микросервиса. Назовите отличия микросервисной архитектуры от сервисноориентированной.

## 13. Назовите основные преимущества и недостатки синхронного межпроцессного взаимодействия. Назовите приёмы, применяемые для компенсации этих недостатков.

## 14. Дайте определение основных видов сообщений, канала сообщений при асинхронном взаимодействии. Опишите роль брокера сообщений.

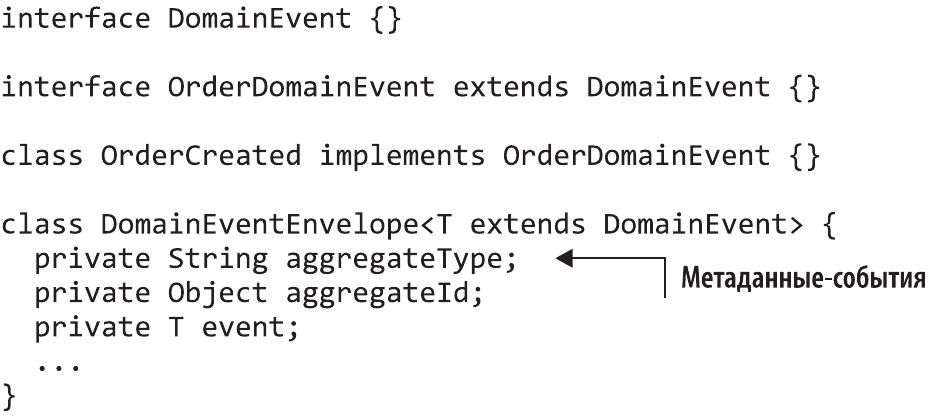
## 15. Приведите формулировку закона Конвея, обратного манёвра Конвея. Объясните как эти идеи используются в микросервисной архитектуре.

## 16. Почему возникают дубликаты сообщений? Какие приёмы используются для работы с дубликатами сообщений? Опишите их.

## 17. Опишите основные приёмы и особенности использования повествований (saga). Приведите пример.

## 18. Опишите принципы построения шестигранной архитектуры. Приведите пример реализации микросервиса в этой архитектуре.

* **Шестигранная архитектура** (она же «гексоганальная архитектура», она же «порты и адаптеры») была придумана Алистером Кокберном и описана в его блоге в 2005 году.
* **Шестигранная архитектура** позволяет приложению в равной степени управляться пользователями, программами, автоматизированными тестовыми или пакетными сценариями, а также разрабатываться и тестироваться изолированно от его конечных устройств времени выполнения и баз данных.
* Алистер Кокберн 2005
* Идея состоит в том, чтобы думать о нашем приложении как о центральном элементе системы, где все входные и выходные данные поступают / выходят из приложения через порт, который изолирует приложение от внешних инструментов, технологий и механизмов доставки.
* Приложение не должно знать о том, кто / что отправляет ему входные данные или получает его выходные данные.
* Это призвано обеспечить некоторую защиту от изменений технологий и бизнес-требований, которые могут сделать продукты устаревшими вскоре после их разработки.
* В отличие от типичной многоуровневой диаграммы, мы представляем бизнес-логику как нечто, разделяющее запросы ***разного*** вида со стороны внешнего мира и ***различные*** технологические абстракции.
* И располагаем схему вертикально.
* Чтобы подчеркнуть, что наше приложение имеет несколько точек входа / выхода, мы нарисуем диаграмму нашего приложения с несколькими гранями.
* Диаграммой мог быть любой многоугольник с несколькими гранями, но выбор пал на шестиугольник. Отсюда и название «Гексагональная архитектура».
* **Порт** — это независимая от потребителя точка входа и выхода в/из приложения.
* На многих языках это будет интерфейс. Например, это может быть интерфейс, используемый для выполнения поиска в поисковой системе. В нашем приложении мы будем использовать этот интерфейс в качестве точки входа и / или выхода, не зная о конкретной реализации.
* **Адаптер** — это класс, который преобразует (адаптирует) один интерфейс в другой.
* **Входящий адаптер** обрабатывает запросы от клиентов и вызывает бизнес-логику (левая сторона).
* **Исходящий адаптер**, который сам вызывается бизнес-логикой, обращается к другим сервисам и приложениям (правая сторона).
* **Порт** описывает внутреннюю модель объекта (API)
* **Адаптер** реализует удобное применение этой модели для конкретного случая использования (use case)
* Доменное событие — это класс с именем на основе страдательного причастия прошедшего времени. Он содержит свойства, которые достаточным образом описывают это событие. Каждое свойство представляет собой либо простое значение, либо объект. Например, класс события OrderCreated содержит свойство orderId.
* OrderCreated является примером доменного события. У него нет никаких полей, поскольку идентификатор заказа входит в состав обертки.



* **Обогащение событий** — добавление в события дополнительной информации, которая нужна потребителю.
* В результате потребители событий становятся более простыми, поскольку им больше не нужно запрашивать данные у сервиса, опубликовавшего событие.
* Агрегат Order может обогатить событие OrderCreated, включив в него информацию о заказе.
* Обогащение событий упрощает потребителей, но его обратной стороной является риск снижения стабильности классов событий.
* Эти классы потенциально могут меняться каждый раз, когда обновляются требования их потребителей. Это способно отрицательно сказаться на поддерживаемости, поскольку изменения такого рода могут затронуть несколько частей приложения.
* К тому же удовлетворение требований каждого потребителя может оказаться недостижимой целью.
* К счастью, во многих ситуациях довольно легко определить, какие свойства следует включить в событие.

Порождение событий  
Преимущества

* **Надёжная публикация доменных событий** о каждом изменении состояния агрегата. Каждое событие может хранить идентификатор пользователя, который внес изменение, что позволяет вести гарантированно корректный журнал аудита. Поток событий можно использовать для целого ряда других задач, включая уведомление пользователей, интеграцию приложений, аналитику и мониторинг.
* **Сохранение истории изменений агрегата**. Вы можете с легкостью реализовать временные запросы, которые извлекают агрегат в одном из его предыдущих состояний. Чтобы определить состояние агрегата в заданный момент времени, нужно свернуть события, которые произошли до этого момента. Например, вы можете легко рассчитать доступные кредитные средства клиента в какой-то момент в прошлом.
* **Отсутствие большинства проблем, связанных с объектно-реляционным разрывом**. Это вводит новый уровень опосредованности между агрегатом и его сериализованным представлением.
* **Машина времени для разработчиков**. Шаблон «Порождение событий» хранит историю всего, что происходило на протяжении жизненного цикла приложения.

Порождение событий  
Недостатки

* **Другая модель программирования** с высоким порогом вхождения.
* **Так же сложно**, как приложение, основанное на обмене сообщениями.
* **Меняющиеся события**. При использовании порождения событий структура событий (и снимков!) будет меняться со временем. Поскольку события хранятся вечно, агрегатам, возможно, придется сворачивать их с учетом нескольких версий структуры.
* **Усложняется удаление данных**. Мягкое удаление подходит для многих видов данных. Но одна из трудностей заключается в соблюдении общего регламента защиты данных.
* **Обращение к хранилищу событий связано с определенными трудностями**. Представьте, что вам нужно найти клиентов, которые исчерпали свой кредитный лимит. Поскольку у вас нет столбца с кредитными средствами, вы не можете написать простой SELECT. Вместо этого нужно использовать сложный и потенциально неэффективный запрос. Что еще хуже, хранилища событий на основе NoSQL обычно поддерживают только запросы по первичному ключу. Следовательно, вам необходимо реализовать запросы с помощью методики CQRS.

ЛЕКЦИЯ 02-05

## 19. Опишите основные принципы построения бизнес-логики на основе порождения событий. Укажите основные преимущества и недостатки этого подхода.

## 20. Опишите шаблоны реализации запросов. Дайте определения, приведите примеры.

## 21. Почему в микросервисной архитектуре нежелательно использовать распределённые транзакции? Какие используются приёмы для достижения согласованности данных? Опишите их.