Java

Курсовой проект

Sergey Proshchaev версия от 25-02-2023

Цели и задачи

- **Реализовать на практике знания, полученные в учебном курсе**

Сформировать в своем портфолио первый Java-проект с использованием Spring Framework и базой H2

Требования к проекту: стек разработки

- Проект выполняется в репозитории **GitHub**

Использование фреймворка **Spring Boot**, **Spring Data Jpa, Spring Web**

Сборка проекта: **Maven**

База данных: **встроенная H2** (рекомендуется, но можно и любую SQL СУБД)

Реализовать в проекте не менее трех сущностей со связями: **один ко многим, многие-к-одному**

Фронтальная часть: вывод главной сущности на **html**-странице с использованием **JavaServer Pages** (**JSP**)

Написать **JUnit-тесты**

Факультативно: использовать Thymeleaf, реализовать авторизацию с использованием **Spring Security**, добавить пользователей

Требования к проекту: оформление

Содержание проекта:

1. **Презентация**
2. **Ссылку на GitHub**
3. **Демонстрация работы проекта**

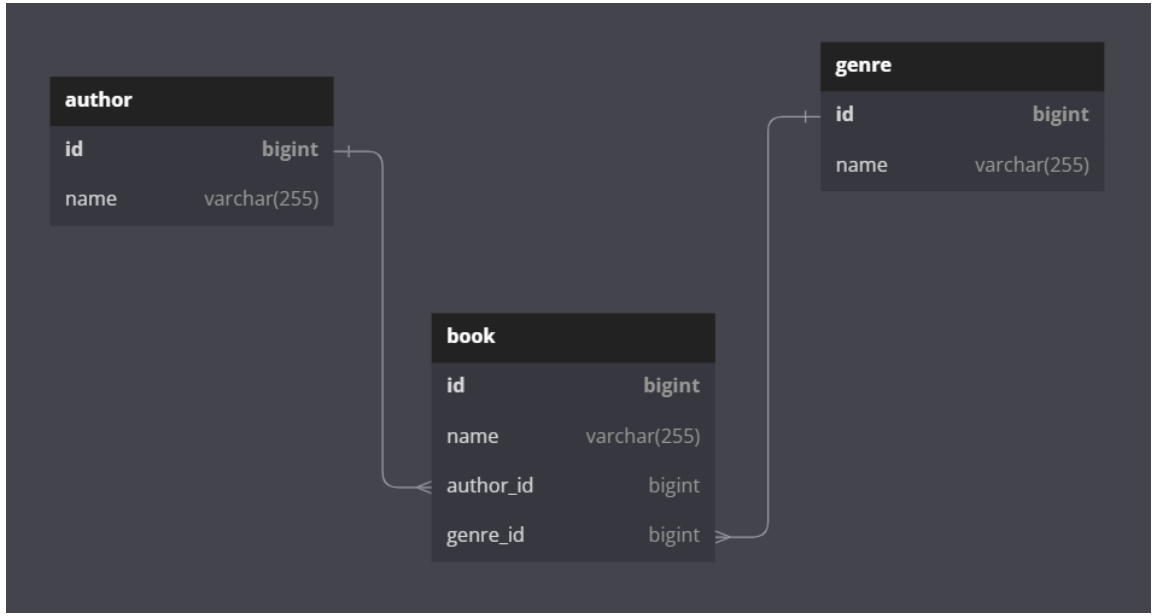
Рекомендованные темы: Онлайн библиотека

Тема 1: «Онлайн библиотека»

Минимальный набор сущностей и полей класса:

1) **Автор: идентификатор, имя**
2) **Жанр: идентификатор, имя**
3) **Книга: идентификатор, название, идентификатор автора, идентификатор жанра**

Реализовать в приложении CRUD-операции над всеми сущностями



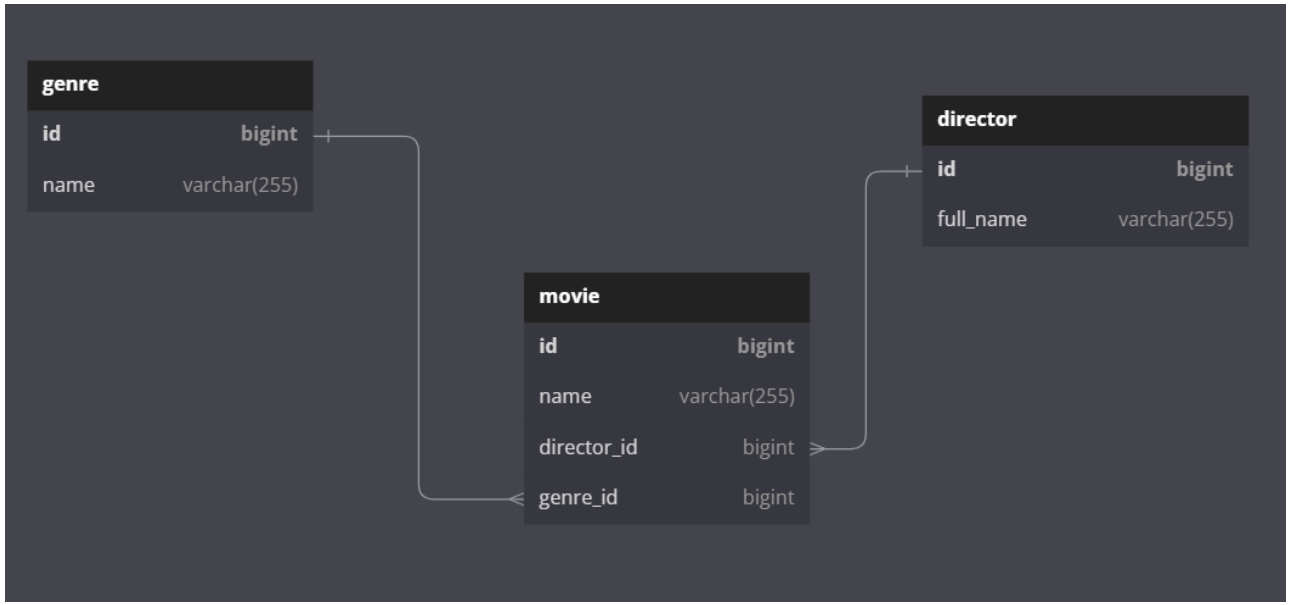
Рекомендованные темы: Прокат DVD-дисков

Тема 2: «Прокат DVD-дисков»

Минимальный набор сущностей и полей класса:

1) **Режиссер: идентификатор, полное имя**
2) **Жанр: идентификатор, имя**
3) **Фильм: идентификатор, название, идентификатор режиссера, идентификатор жанра**

Реализовать в приложении CRUD-операции над всеми сущностями



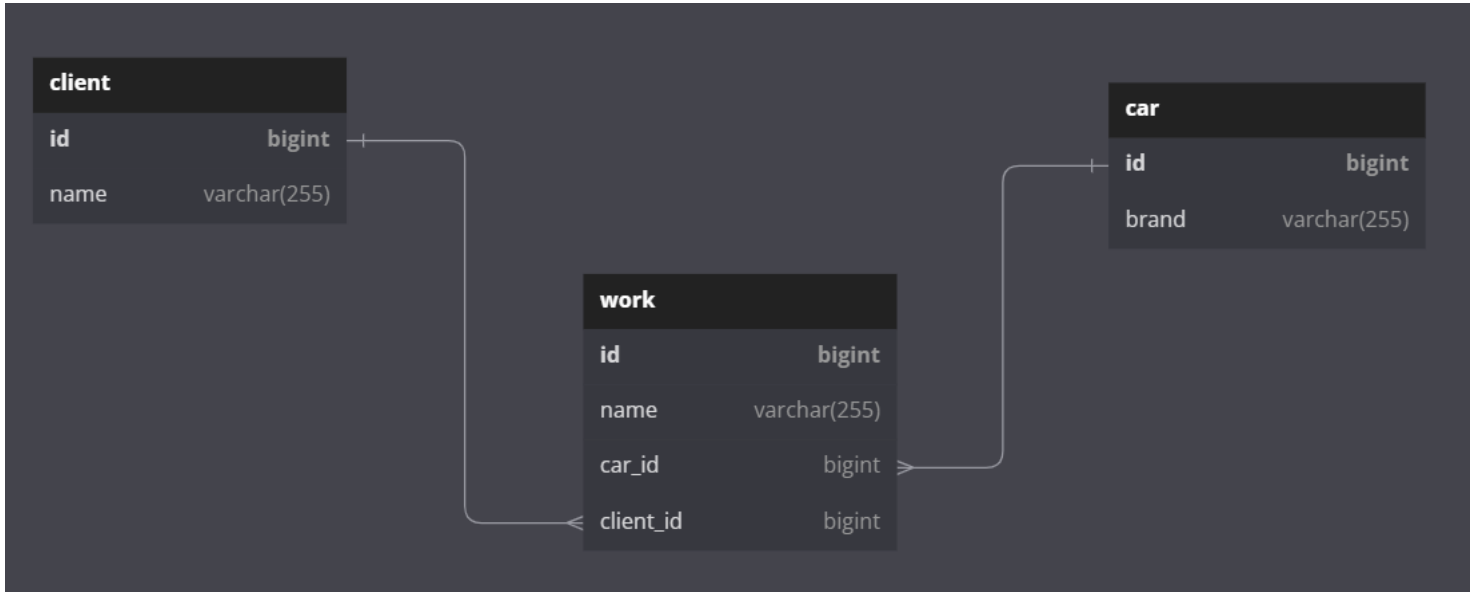
Рекомендованные темы: Автосервис

Тема 3: «Автосервис»

Минимальный набор сущностей и полей класса:

1) **Клиент: идентификатор, полное имя**
2) **Автомобиль: идентификатор, марка**
3) **Выполненная работа: идентификатор, название вида работ, идентификатор клиента, идентификатор автомобиля**

Реализовать в приложении
CRUD-операции над всеми сущностями



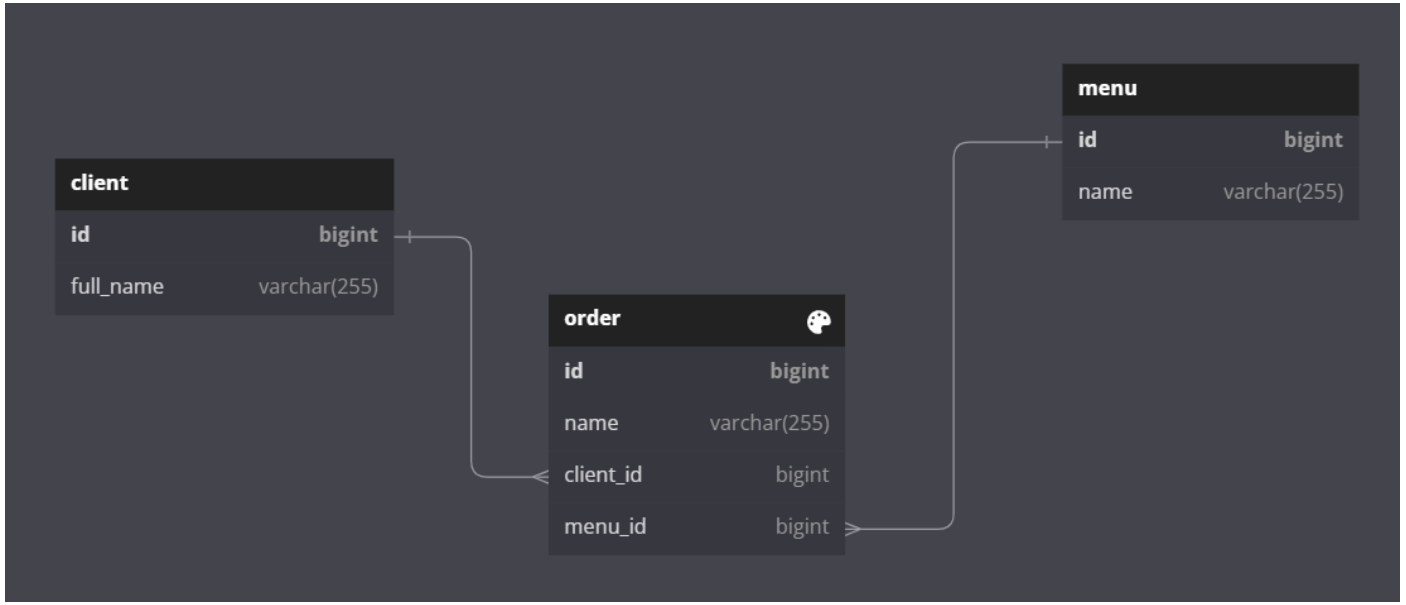
Рекомендованные темы: Сервис доставки

Тема 4: «Сервис доставки»

Минимальный набор сущностей и полей класса:

1) **Клиент: идентификатор, полное имя**
2) **Меню: идентификатор, наименование блюда**
3) **Выполненные работы: идентификатор, название, идентификатор клиента, идентификатор блюда из меню**

Реализовать в приложении CRUD-операции над всеми сущностями



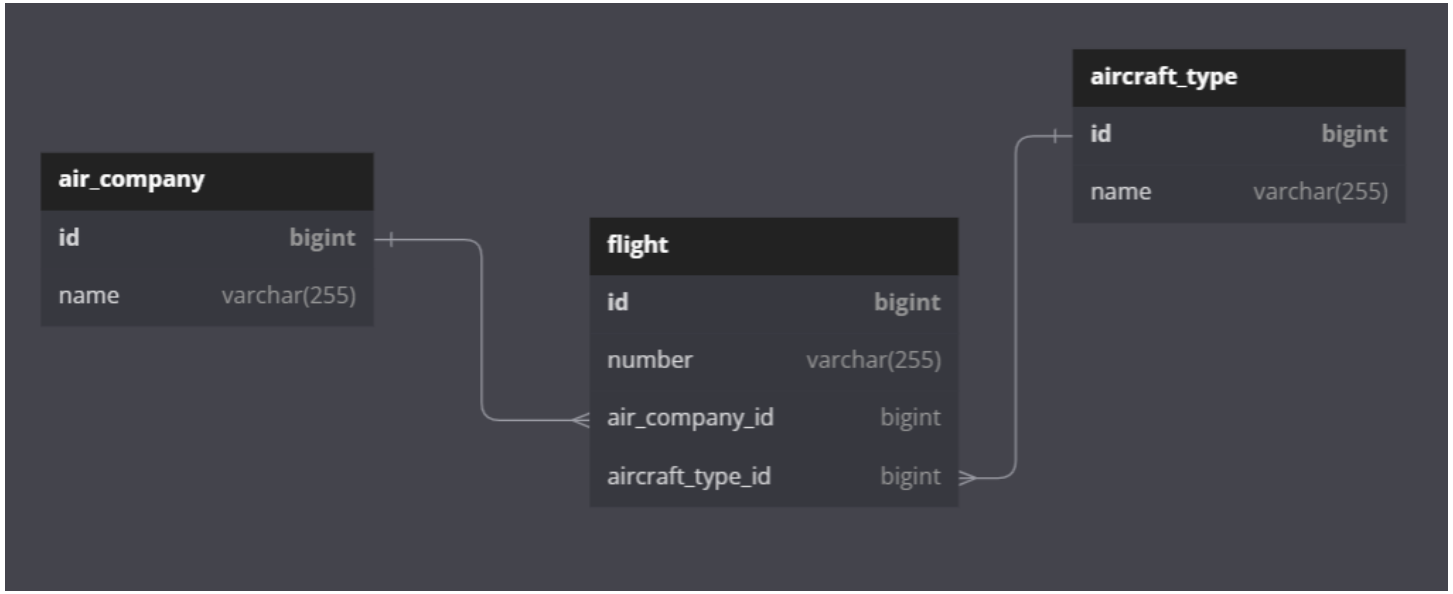
Рекомендованные темы: Расписание вылетов

Тема 5: «Расписание вылетов»

Минимальный набор сущностей и полей класса:

1) **Авиакомпания: идентификатор, наименование авиакомпании**
2) **Тип воздушного судна: идентификатор, тип воздушного судна**
3) **Номера рейсов: идентификатор, номер рейса, идентификатор авиакомпании, идентификатор типа воздушного судна**

Реализовать в приложении
CRUD-операции над всеми сущностями



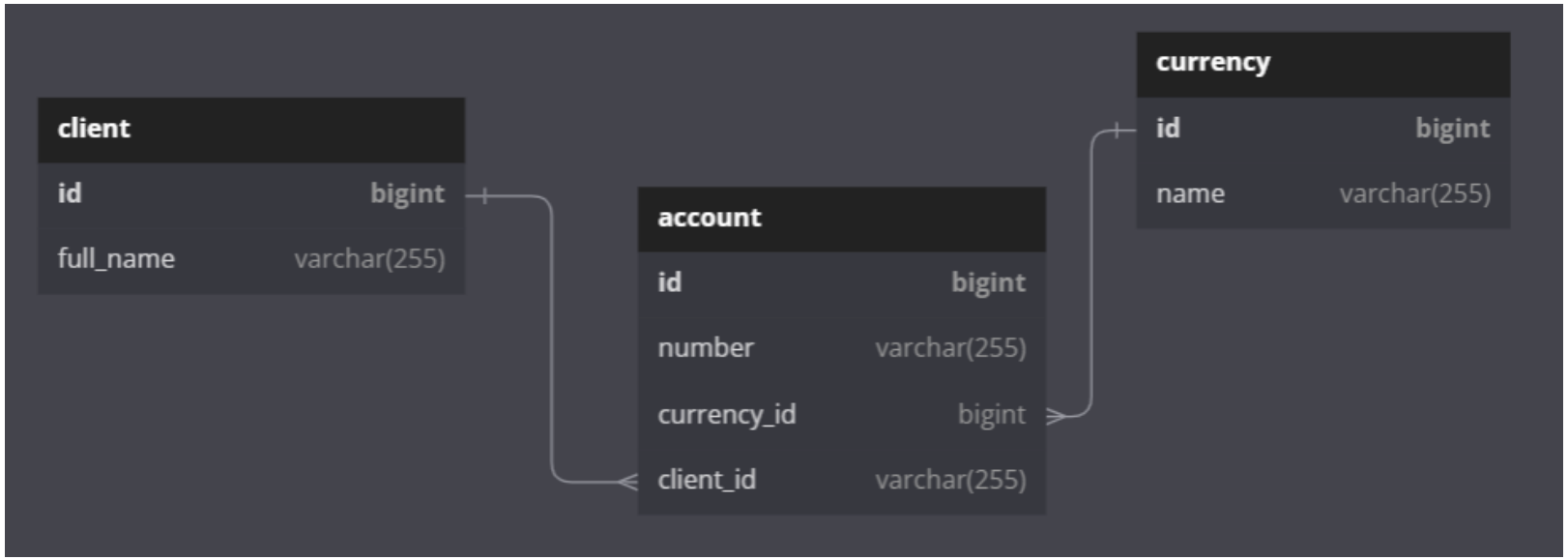
Рекомендованные темы: Банковский счет

Тема 6: «Банковский счет»

Минимальный набор сущностей и полей класса:

1) **Клиент: идентификатор, полное имя**
2) **Валюта: идентификатор, наименование валюты**
3) **Счет: идентификатор, номер счета, идентификатор валюты, идентификатор клиента**

Реализовать в приложении CRUD-операции над всеми сущностями



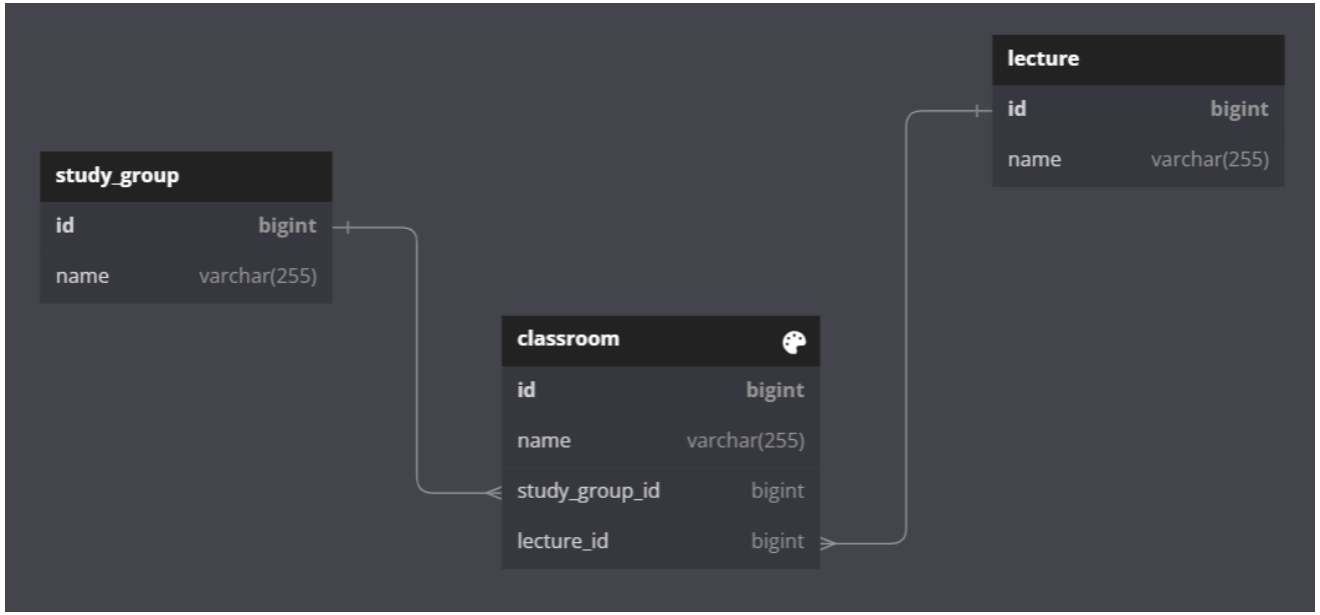
Рекомендованные темы: Расписание занятий

Тема 7: «Расписание занятий»

Сущности:

1) **Учебная группа: идентификатор, наименование группы**
2) **Учебная дисциплина: идентификатор, наименование предмета**
3) **Аудитория: идентификатор, номер аудитории, идентификатор учебной группы, идентификатор дисциплины**

Реализовать в приложении CRUD-операции над всеми сущностями



Принципы SOLID

Принципы SOLID - 5 правил разработки ПО которым нужно следовать при разработке программ чтобы их проще было масштабировать и поддерживать

S – Single Responsibility (Принцип единственной ответственности)

O — Open-Closed (Принцип открытости-закрытости)

L — Liskov Substitution (Принцип подстановки Барбары Лисков)

I — Interface Segregation (Принцип разделения интерфейсов)

D — Dependency Inversion (Принцип инверсии зависимостей)

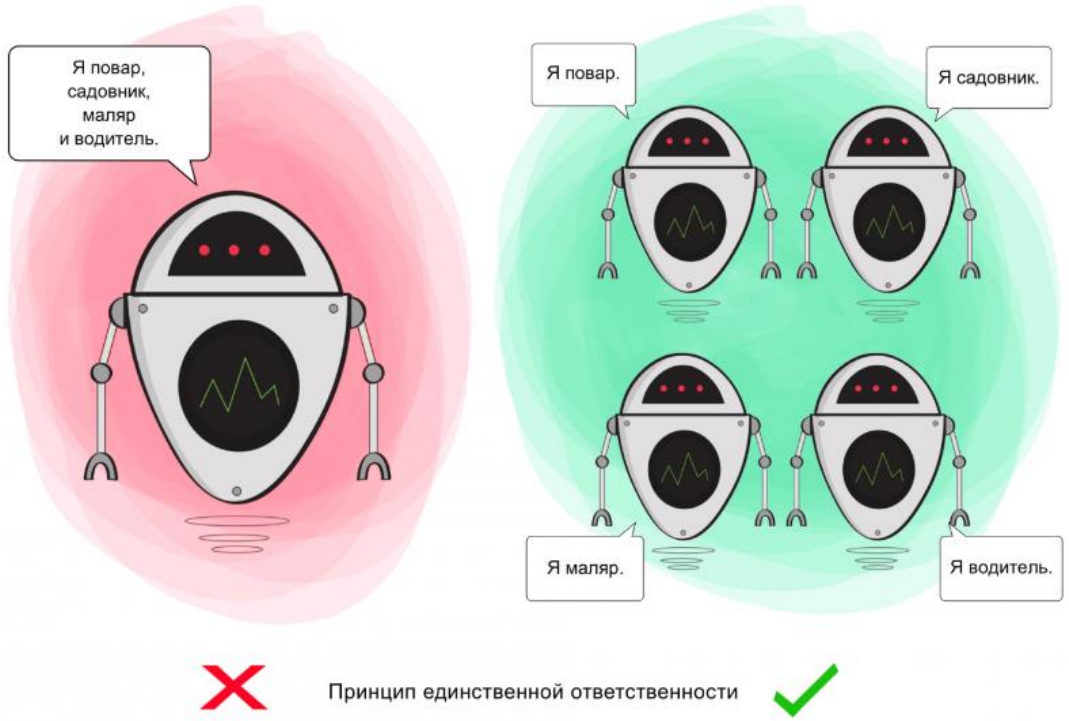
Принципы SOLID: Single Responsibility

S – Single Responsibility

Принцип единственной ответственности

Каждый класс должен отвечать только за одну операцию

Если класс отвечает за несколько операций сразу, вероятность возникновения багов возрастает – внося изменения, касающиеся одной из операций вы, сами того не подозревая, можете затронуть и другие



Назначение

Принцип служит для разделения типов поведения, благодаря которому ошибки, вызванные модификациями в одном поведении, не распространялись на прочие, не связанные с ним типы

(увеличить)

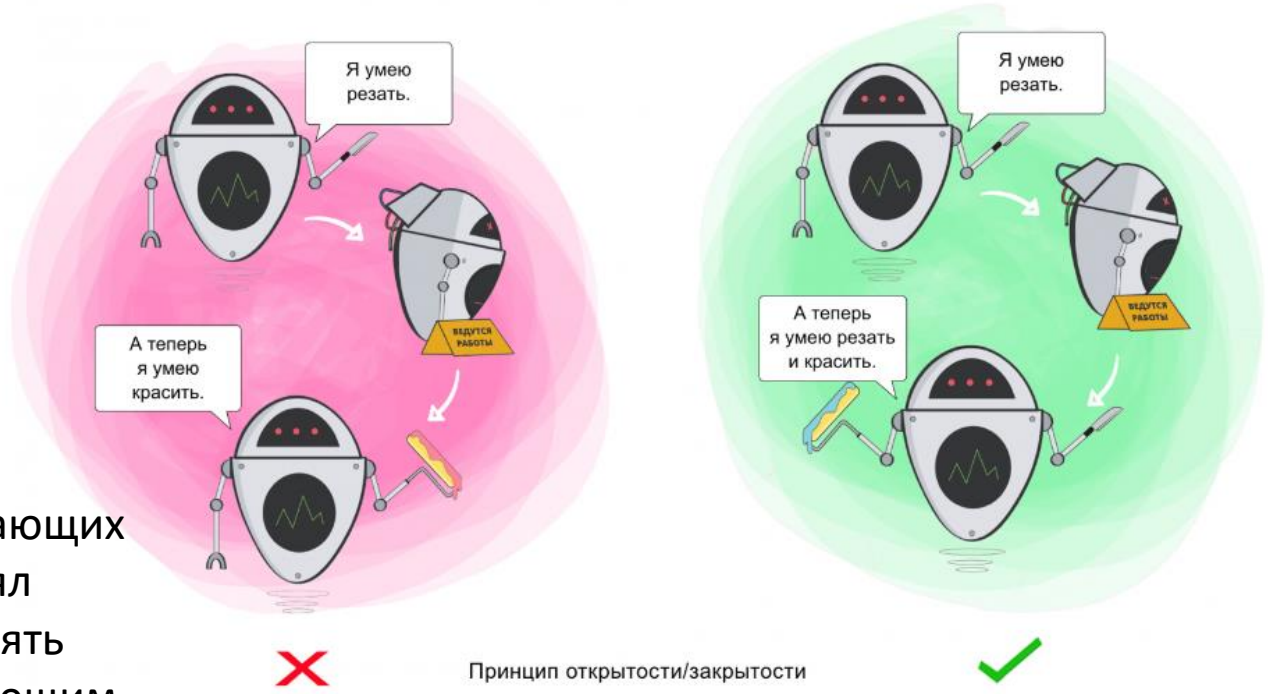
Принципы SOLID: Open-Closed

O — Open-Closed

Принцип открытости-закрытости

Классы должны быть открыты для расширения, но закрыты для модификации

Когда вы меняете текущее поведение класса, эти изменения сказываются на всех системах, работающих с данным классом. Если хотите, чтобы класс выполнял больше операций, то идеальный вариант – не заменять старые на новые, а добавлять новые к уже существующим



(увеличить)

Назначение

Принцип служит для того, чтобы делать поведение класса более разнообразным, не вмешиваясь в текущие операции, которые он выполняет. Благодаря этому вы избегаете ошибок в тех фрагментах кода, где задействован этот класс

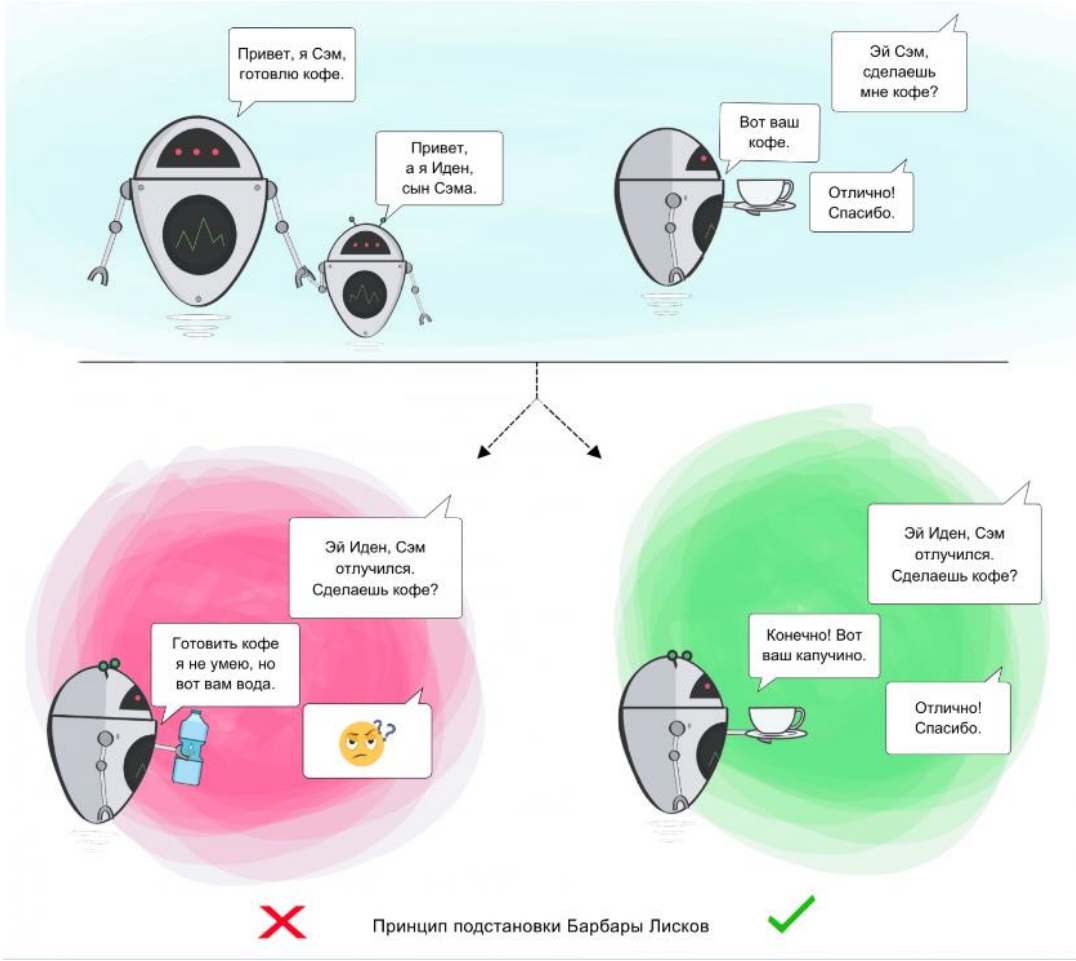
Принципы SOLID: Liskov Substitution

L — Liskov Substitution

Принцип подстановки Барбары Лисков

Если П является подтипом Т, то любые объекты типа Т, присутствующие в программе, могут заменяться объектами типа П без негативных последствий для функциональности программы

Если есть класс и вы создаете на его базе другой класс, исходный класс становится родителем, а новый – потомком. Потомок должен производить такие же операции, как и родитель (это наследственность)



Назначение

Принцип служит для того, чтобы обеспечить постоянство: класс-родитель и класс-потомок могут использоваться одинаковым образом без нарушения работы программы

(увеличить)

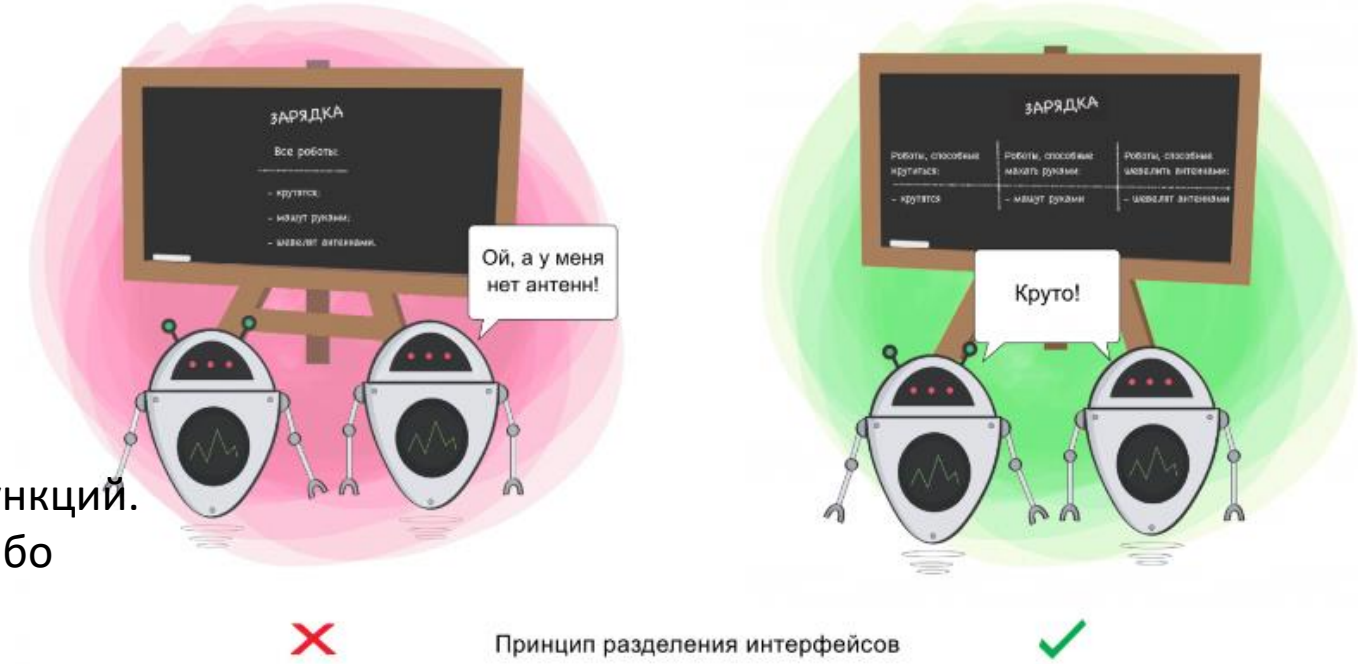
Принципы SOLID: Interface Segregation

I — Interface Segregation

Принцип разделения интерфейсов

Не следует ставить клиента в зависимость от методов, которые он не использует

Класс должен производить только те операции, которые необходимы для осуществления его функций. Все другие действия следует удалить совсем, либо переместить в другой класс



(увеличить)

Назначение

Принцип служит для того, чтобы раздробить единый набор действий на ряд наборов поменьше – таким образом, каждый класс делает то, что от него действительно требуется, и ничего больше

Принципы SOLID: Dependency Inversion

D — Dependency Inversion

Принцип инверсии зависимостей

Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций

Класс не должен соединяться с инструментом, который применяет для выполнения операции. Класс должен быть соединён с интерфейсом, который поможет установить связь между инструментом и классом. Ни интерфейс, ни класс, не обязаны вникать в специфику работы инструмента. Инструмент должен подходить под требования интерфейса

(увеличить)

Назначение

Этот принцип служит для того, чтобы устранить зависимость классов верхнего уровня от классов нижнего уровня за счёт введения интерфейсов

Принципы SOLID: итоги

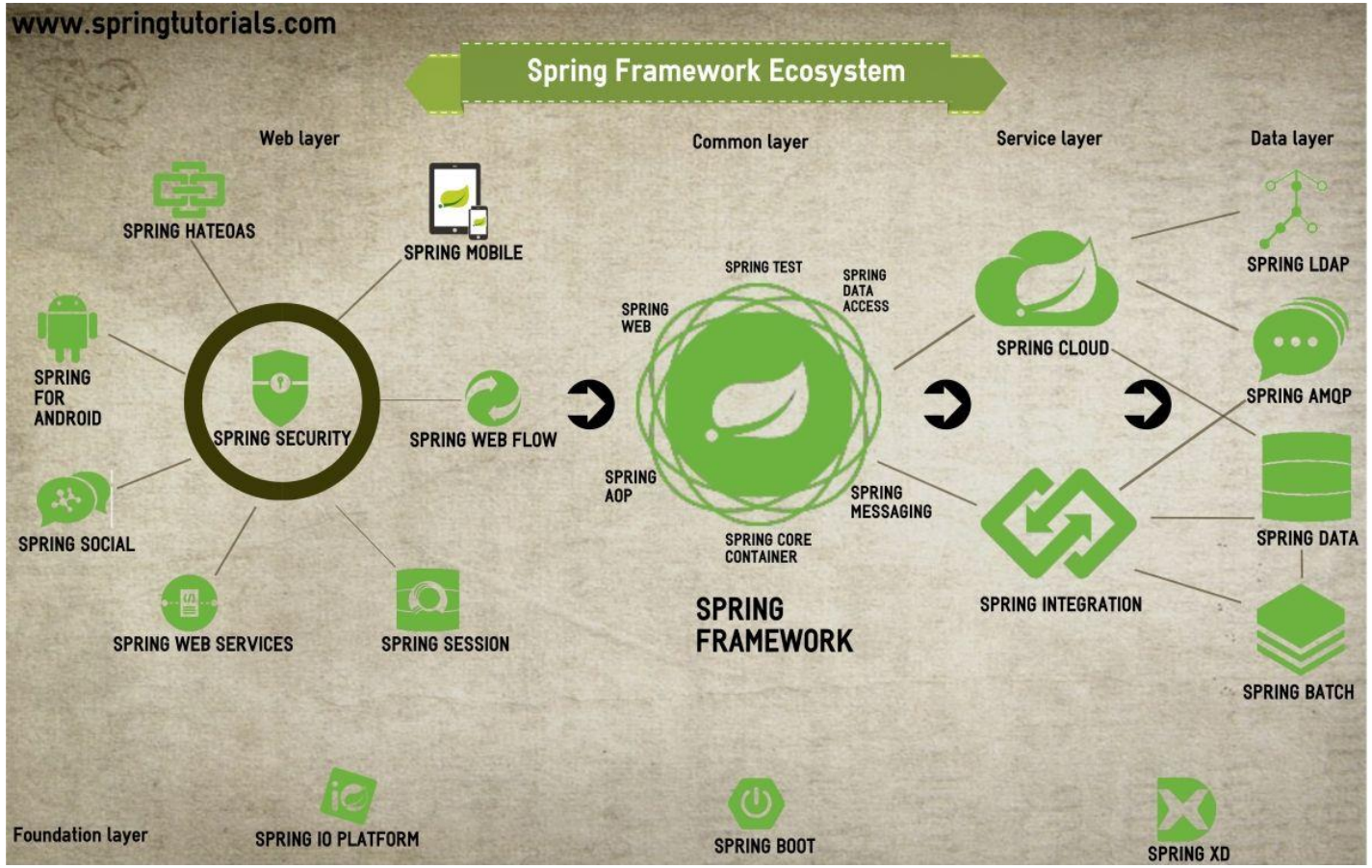
SOLID упрощенно:

- **Single Responsibility** – делай модули меньше (1 ответственность)
- **Open/Closed** – делай модули расширяемыми
- **Liskov Substitution** – наследники ведут себя так же, как родител
- **Interface Segregation** – дели слишком сложные интерфейсы
- **Dependency Inversion** – используй интерфейсы

https://bit.ly/Solid_ppt

Spring Framework

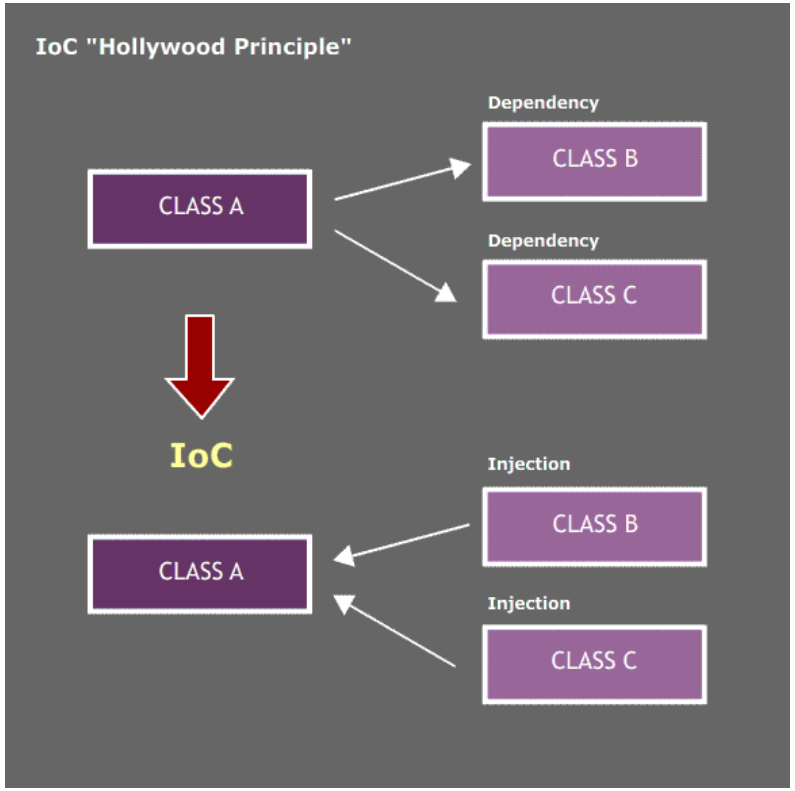
- **Spring Framework** — универсальный фреймворк с открытым исходным кодом для Java-платформы



Inversion of Control (IoC)

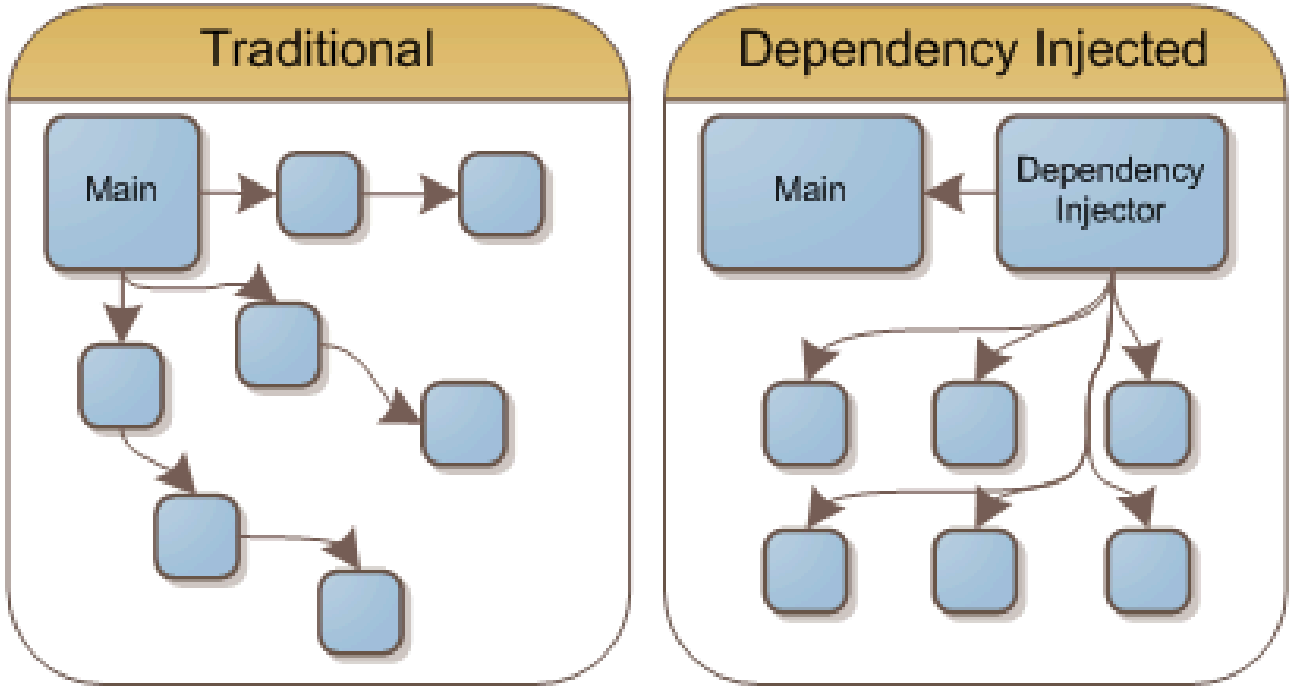
Инверсия управления (англ. *Inversion of Control, IoC*) — важный принцип объектно-ориентированного программирования, используемый для уменьшения зацепления в компьютерных программах .

Также архитектурное решение интеграции, упрощающее расширение возможностей системы, при котором контроль над потоком управления программы остается за каркасом



Dependency injection (DI)

Внедрение зависимости (англ. Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту. Является специфичной формой «инверсии управления» (англ. Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с принципом единственной ответственности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму



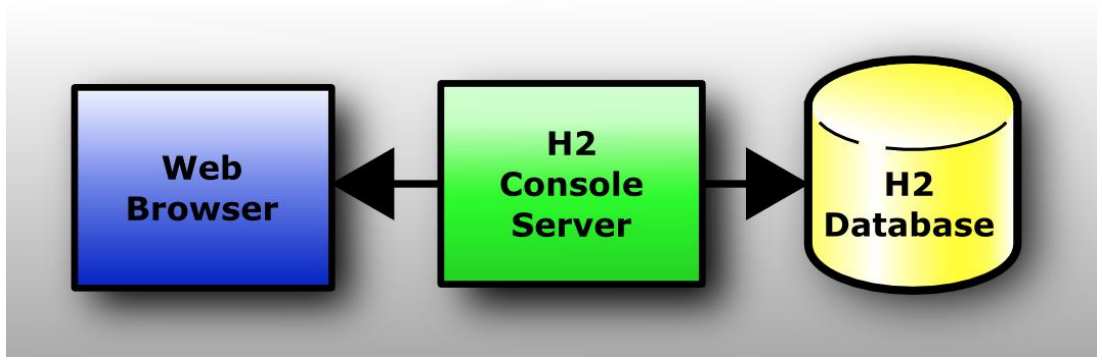
База данных H2

H2 – это легковесная база данных, может быть встроена в приложения Java или работать в режиме клиент-сервер. H2 может быть настроена для работы в качестве базы данных памяти, что означает, что данные не будут сохраняться на диске. В основном используется для разработки и тестирования

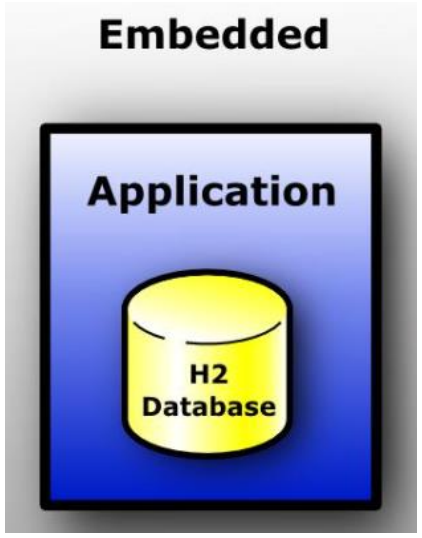
Особенности:

- Чрезвычайно быстрый движок базы данных
- Открытый исходный код, написанный на Java
- Поддерживает стандартные API SQL и JDBC, может использовать драйвер PostgreSQL ODBC
- Имеет встроенный и серверный режим
- Поддерживает **кластеризацию** и **многоверсионный параллелизм**
- Наличие сильных особенностей безопасности

1



2



Ключи SQL: первичный ключ

Столбец, который в базе данных должен быть уникальным помечают первичным ключом. Первичный ключ или **primary key** означает, что в таблице значение колонки primary key не может повторяться.

Таким образом данный ключ позволяет однозначно идентифицировать запись в таблице не боясь при этом, что значение столбца повториться

Клиенты

1 Код	Организация	Имя
1	Организация A	Ольга
2	Организация B	Григорий
3	Организация C	Владимир

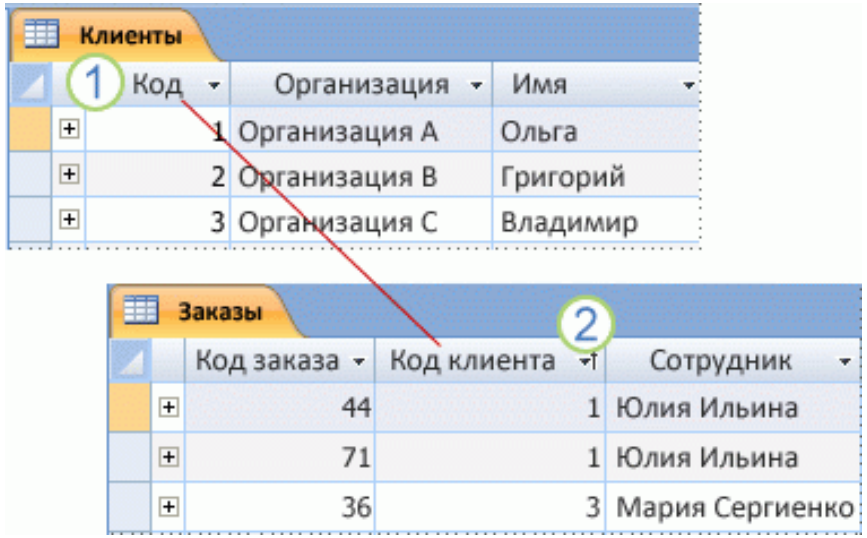
Заказы

Код заказа	2 Код клиента	Сотрудник
44	1	Юлия Ильина
71	1	Юлия Ильина
36	3	Мария Сергиенко

Ключи SQL: внешний ключ

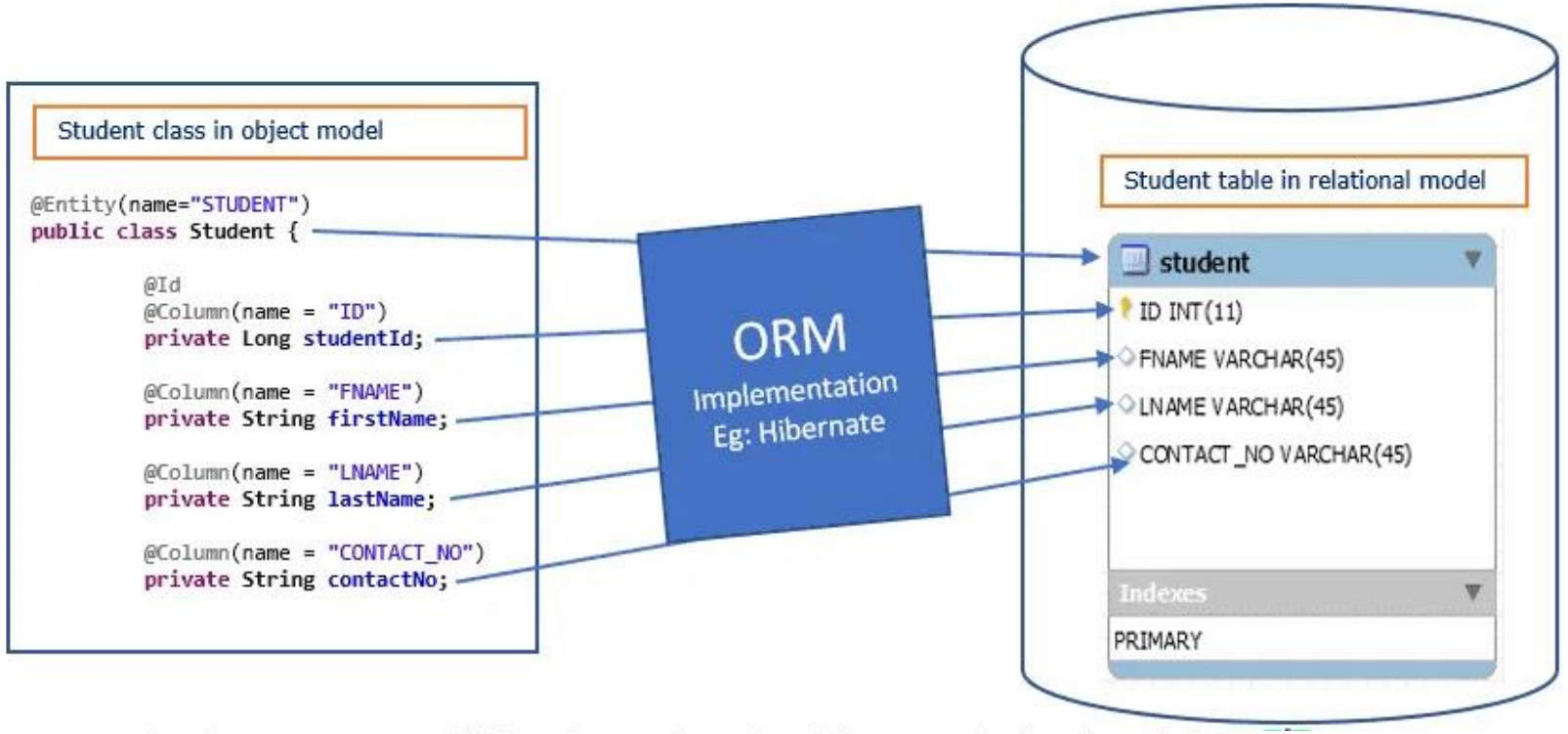
- Есть еще **внешний ключ** (*foreign key*). Его еще называют ссылочным. Он нужен для связывания таблиц между собой

2. Внешним ключем будет поле Код клиента в таблице **Заказы**. Как правило, при создании таблицы задают колонку уникальных целочисленных значений



Object-Relational Mapping (ORM)

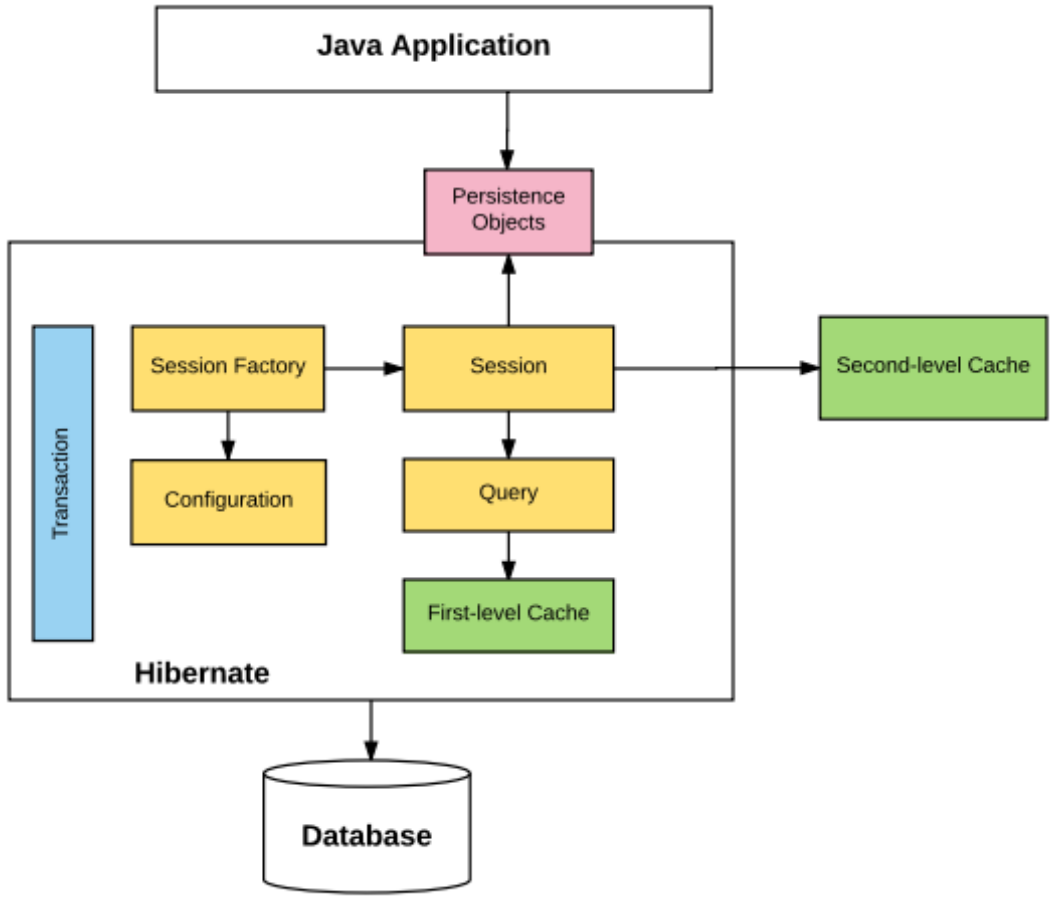
- Объектно-реляционная модель, или ORM, позволяет создать программную «виртуальную» базу данных из объектов



ORM implements responsibility of mapping the Object to Relational Model.

Hibernate

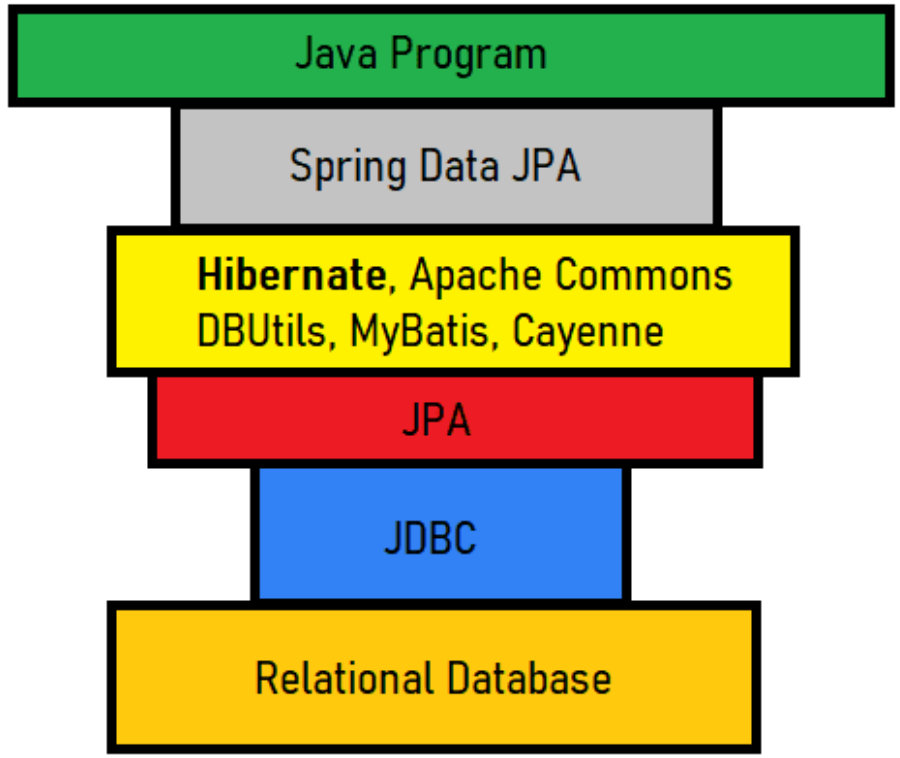
Hibernate — популярная реализация ORM модели, построеная на спецификации JPA 2.1 — наборе правил, который описывает взаимодействие программных объектов с записями в базах данных



Spring Data JPA

Spring Data JPA упрощает разработку JPA-приложений, за чет расширения поддержки JPA-слоя доступа к данным

Основное понятие в Spring Data — это репозиторий, представляющий собой несколько интерфейсов, использующих JPA Entity для взаимодействия



Оболочка Spring Shell

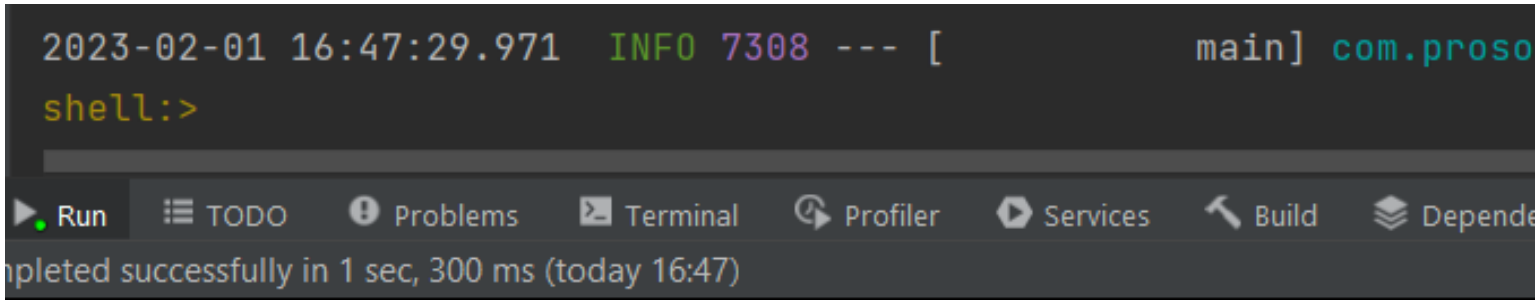
Если для вашего проекта Web UI или UI на Swing слишком сложен, но хочется использовать все возможности Spring, то подойдет **Spring Shell** для создания **CLI-интерфейса** (Command line interface - интерфейс командной строки)

Пример создания простейшего Spring Shell - метода:

```
@Component
public class RunCommands {
    @ShellMethod(value = "run", group = "general")
    public String run() {
        return "Run
    }
}
```

Spring Shell command must be declared inside a class annotated with '@ShellComponent'

Пример командной строки Spring Shell:



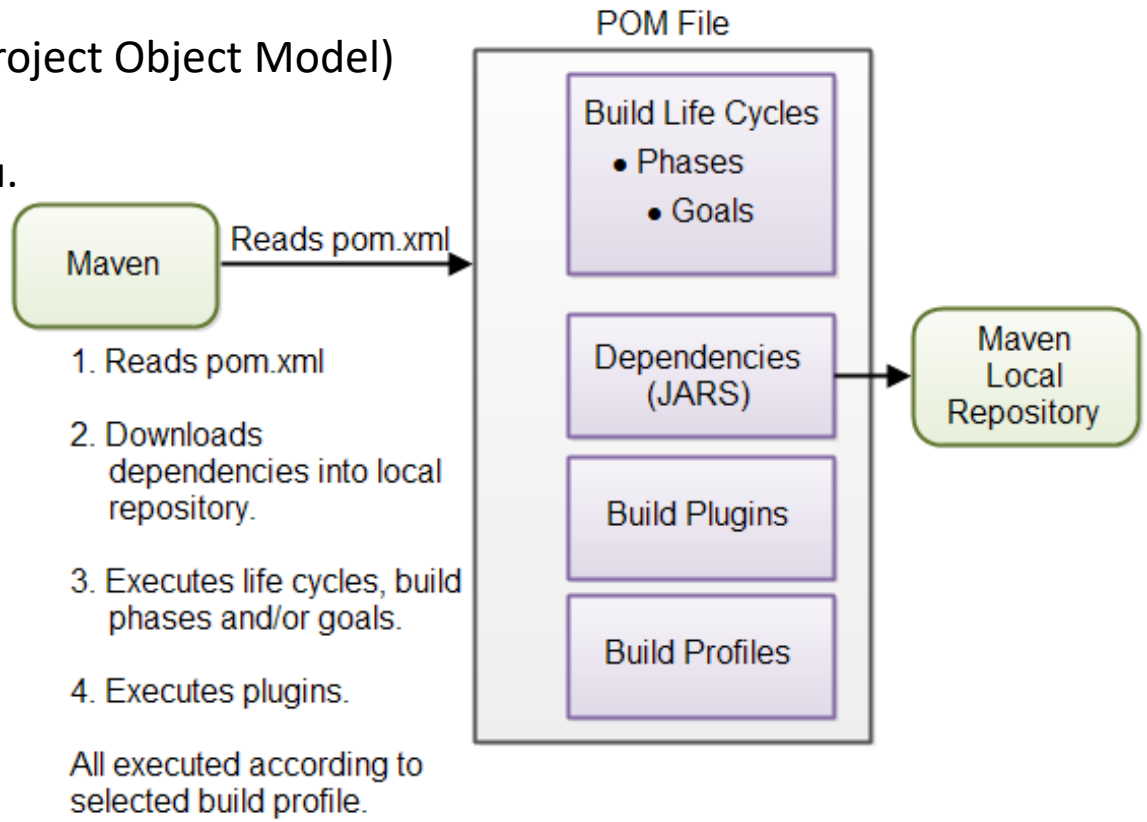
Сборщик проекта Maven

Структура проекта описывается в файле **pom.xml** (POM – Project Object Model)

Артефакт — любая библиотека, хранящаяся в репозитории. Это может быть какая-то зависимость или плагин

Зависимости — библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования

Плагины используются самим Maven'ом при сборке проекта или для каких-то других целей (деплоймент, создание файлов проекта и др.)

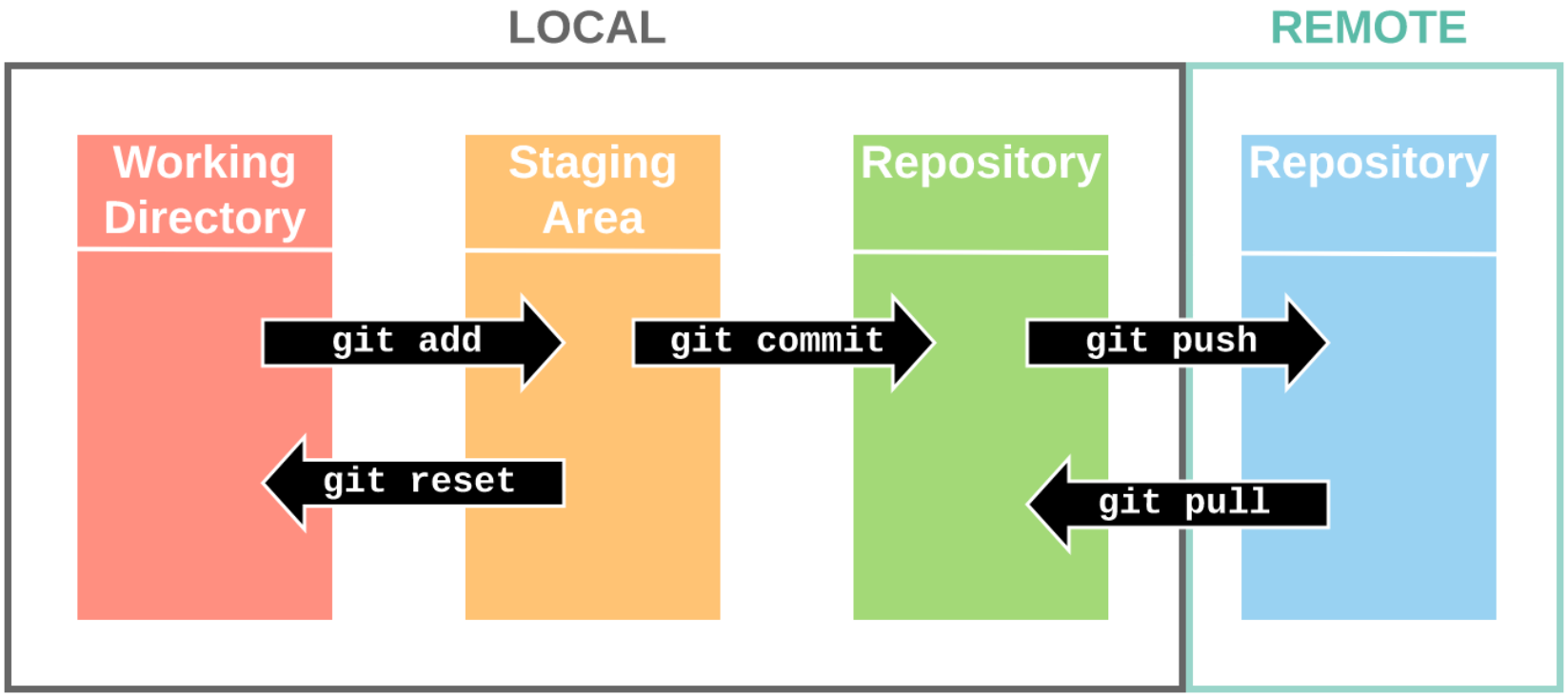


Архетип — стандартная компоновка файлов и каталогов в проектах различного рода (веб, swing-проекты и прочие). Maven знает, как обычно строятся проекты и в соответствии с архетипом создает структуру каталогов

Система управления версиями Git

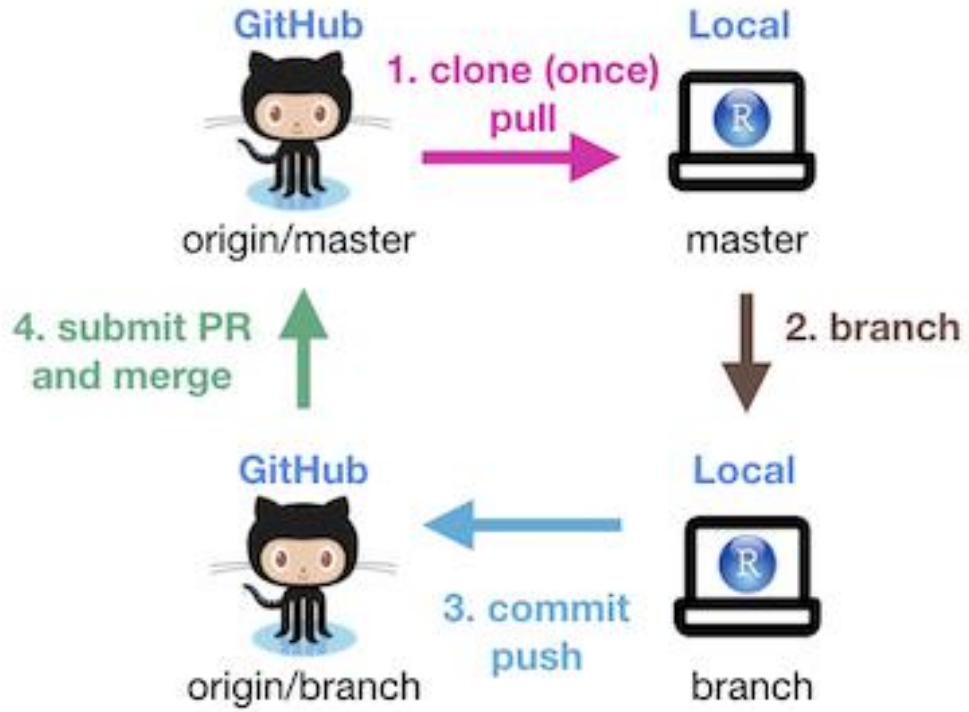
Git — система управления версиями с распределенной архитектурой

В Git каждая рабочая копия кода сама по себе является **репозиторием**, что позволяет всем разработчикам хранить историю изменений в полном объеме



Система управления версиями Git

Полный цикл работы с Git-репозиторием:

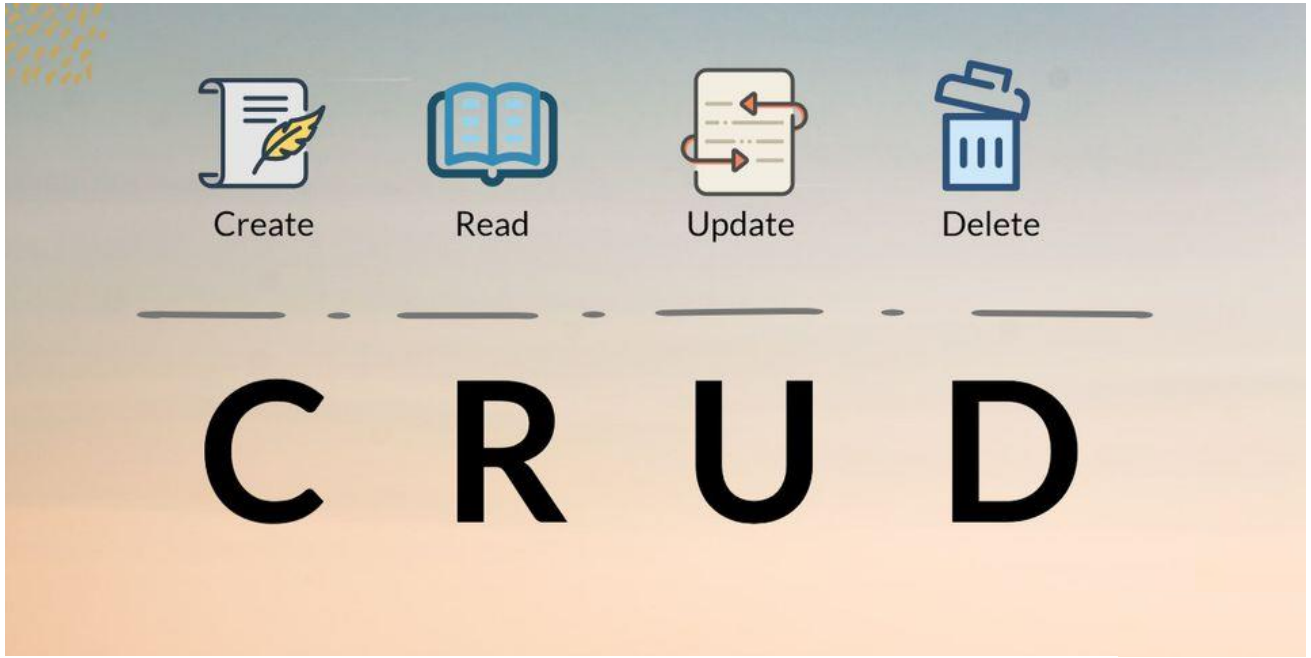


bit.ly/Git_manual

Создание проекта: CRUD операции

CRUD — акроним, обозначающий четыре базовые функции, используемые при работе с базами данных:

1) создание (англ. **create**),
2) чтение (**read**),
3) модификация (**update**),
4) удаление (**delete**)



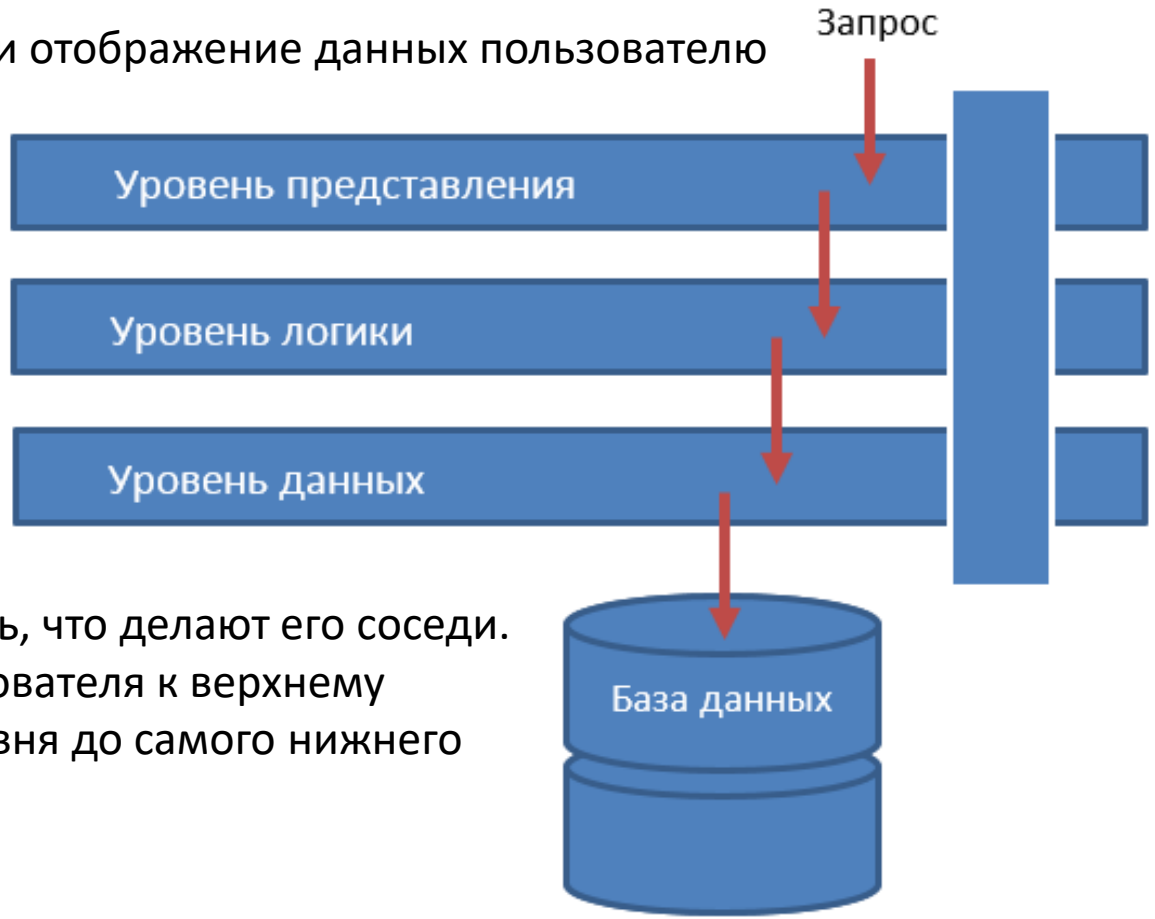
Архитектура приложения

Многоуровневая архитектура:

Уровень представления - пользовательский интерфейс и отображение данных пользователю

Уровень логики содержит алгоритмы работы прил.

Уровень данных обеспечивает доступ к базе данных



Разделение ответственности: слою не обязательно знать, что делают его соседи. Если все три слоя являются закрытыми, то запрос пользователя к верхнему уровню инициирует цепочку обращений с верхнего уровня до самого нижнего

Сущности Entity

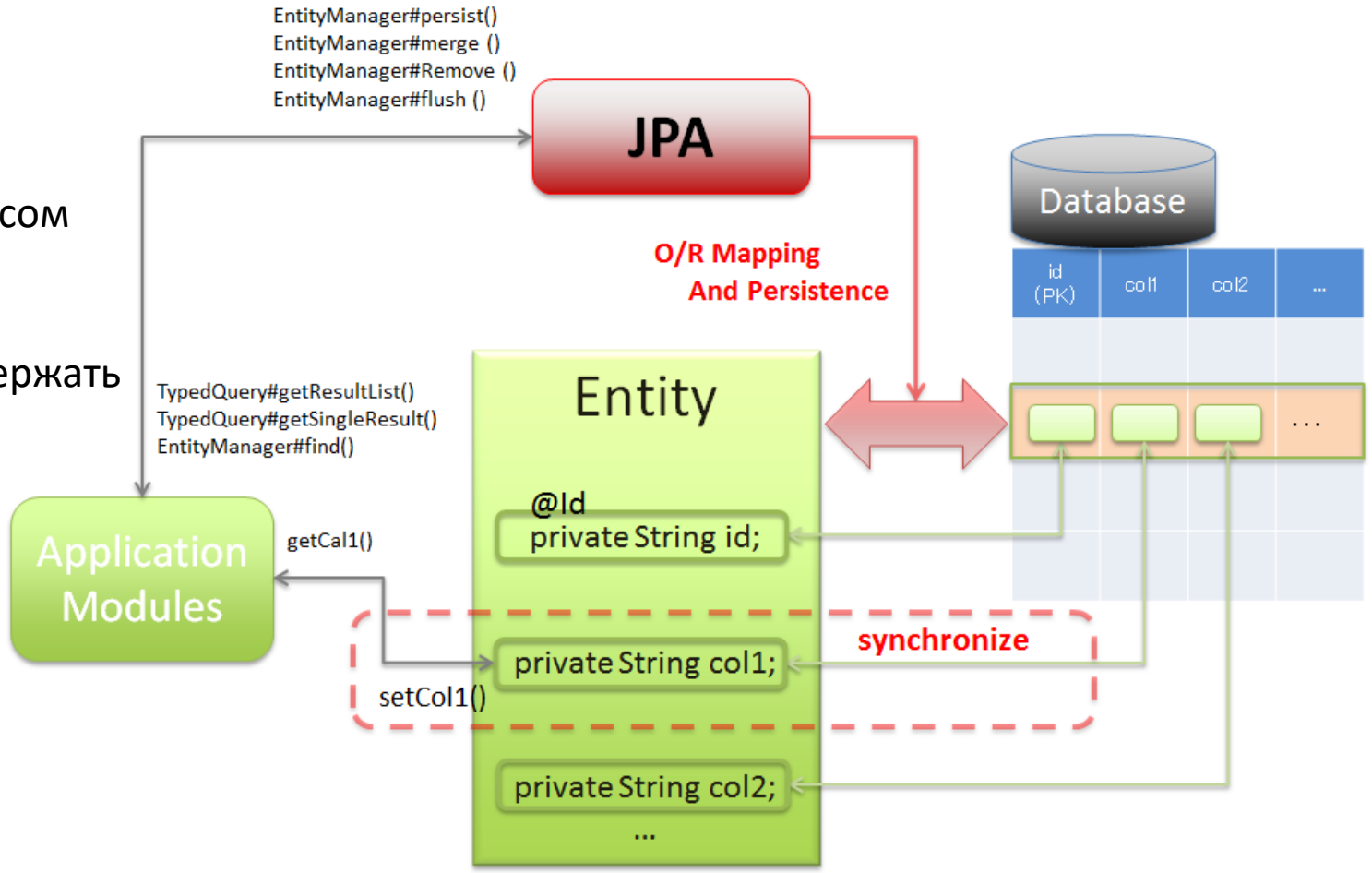
Entity (Сущность) — POJO-класс связанный с БД с помощью аннотации `@Entity`:

1) Должен иметь **пустой конструктор** (`public` или `protected`)

2) Не может быть вложенным, интерфейсом или enum

3) Не может быть `final` и не может содержать `final`-полей

4) Должно быть одно поле **`@Id`-поле**



bit.ly/3Y0OEO9

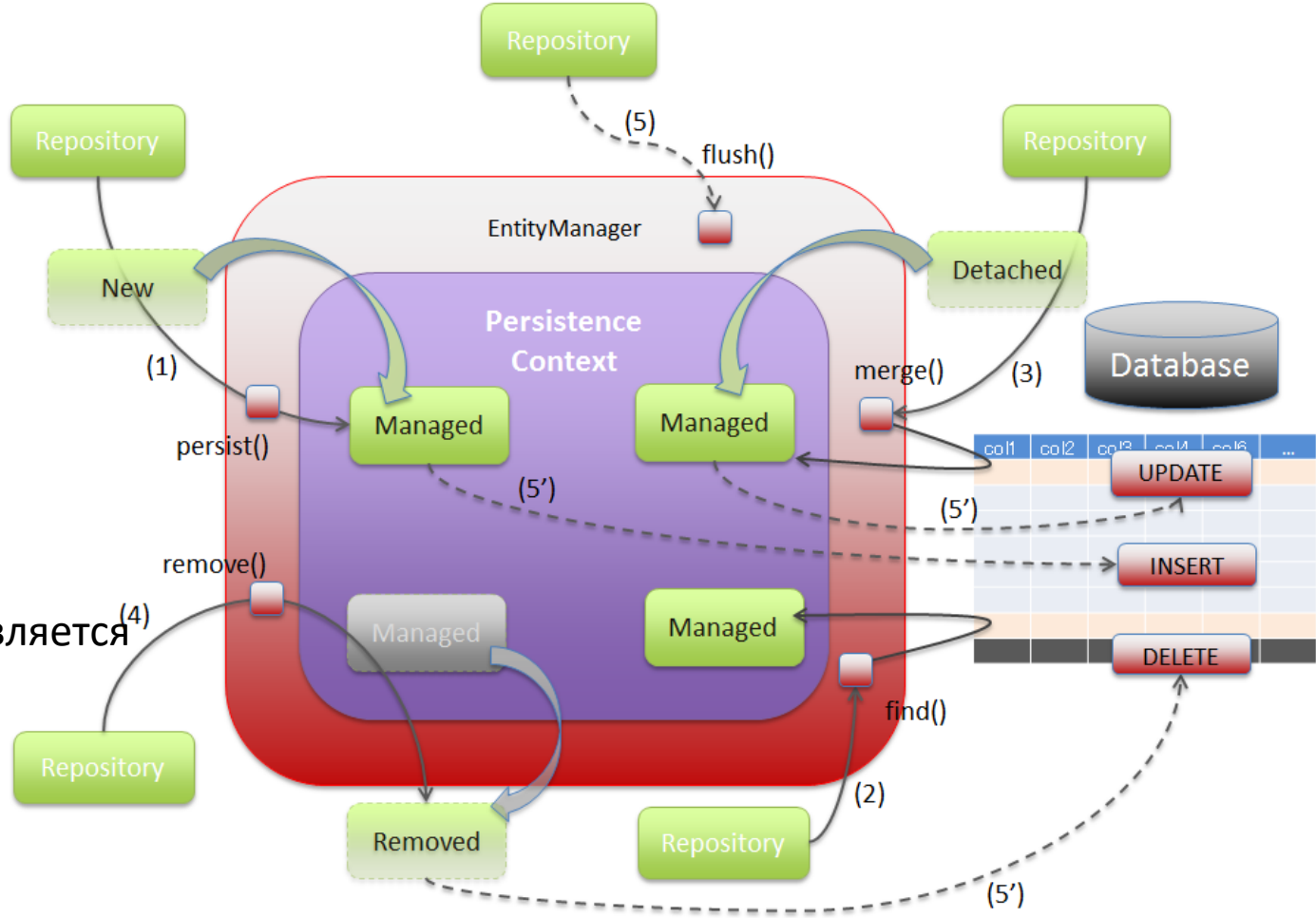
Жизненный цикл Entity и EntityManager

EntityManager – это стандартный интерфейс JPA для загрузки и сохранения сущностей, управляющий **жизненным циклом Entity**

Сущность в Hibernate может быть в одном из следующих состояний:

- *Transient (New)* — Новая
- *Managed (Persistent)* — Управляемая
- *Detached* — Отсоединенная
- *Removed (Deleted)* — Удаленная

Переход из одного состояния в другое осуществляется с помощью методов EntityManager или Session



JpaRepository

JpaRepository – это интерфейс фреймворка Spring Data предоставляющий набор стандартных методов JPA для работы с БД:

1 – Имя репозитория начинается с имени сущности **Name**Reposytory (*необязательно но рекомендуется*)

2 – Второй Generic должен быть оберточным типом того типа которым есть ID нашей сущности (*обязательно*)

3 – Первый Generic должен быть объектом нашей сущности для которой мы создали Repository (*обязательно*)

4 – Мы должны унаследовать свой интерфейс от JpaRepository (*обязательно*)

```
@Entity
@Table(name = "bank")
public class Bank {

    @Id
    @GeneratedValue(generator = "increment")
    @GenericGenerator(name= "increment", strategy= "increment")
    @Column(name = "id", length = 6, nullable = false)
    private long id;
```

```
package com.devcolibri.dataexam.repository;

import com.devcolibri.dataexam.entity.Bank;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BankRepository extends JpaRepository<Bank, Long> {
}
```

Класс отмечается аннотацией @Repository

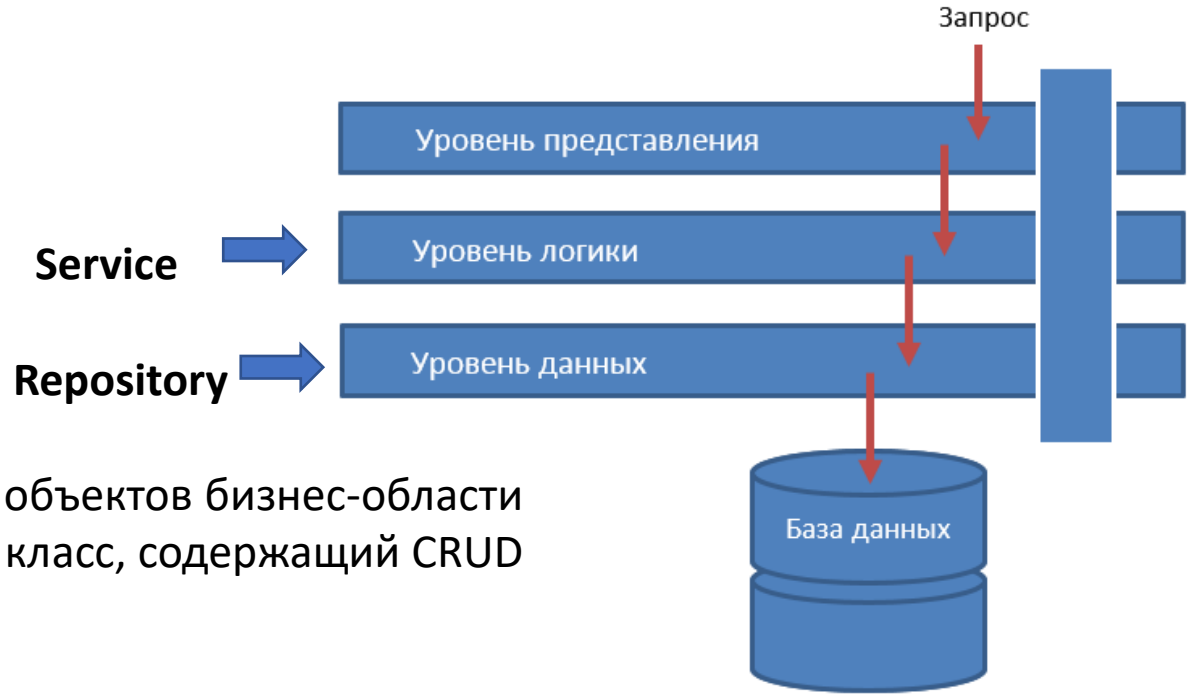
В классе конфигурации необходимо указать аннотацию **@EnableJpaRepositories**

Services

Напрямую использовать **Repositories** для получение данных в *Пользовательский Интерфейс* не принято и считается плохим тоном, для этого были придуманы **Services**

Service – это Java класс, который предоставляет с себя основную (*Бизнес-Логику*). В основном сервис использует готовые **DAO*/Repositories** или же другие сервисы, для того чтобы предоставить конечные данные для пользовательского интерфейса

Класс отмечается аннотацией @Service



***Data Access Object (DAO)** — паттерн для сохранения объектов бизнес-области в базе данных. В самом широком смысле, DAO — это класс, содержащий CRUD методы для конкретной сущности

Лучшим решением будет использование паттерна **Repository**!

Аннотации в Spring

Некоторые аннотации, использующиеся в Spring:

@Component
@Service
@Repository

@Entity
@Table
@Column
@Id
@GeneratedValue

@Transactional
@Autowired

bit.ly/3Y0OEO9

bit.ly/3Yk0rXp

Способы внедрения зависимостей (Dependency Injection) в Spring

Если в классе есть конструктор, то можно внедрить зависимость через конструктор. При создании класса контейнер Spring вызовет конструктор и передаст зависимость в качестве аргумента конструктора:

```
@Component
public class CarWithConstructor {

    private Engine engine;

    @Autowired
    public CarWithConstructor(Engine engine) {
        this.engine = engine;
    }
}
```

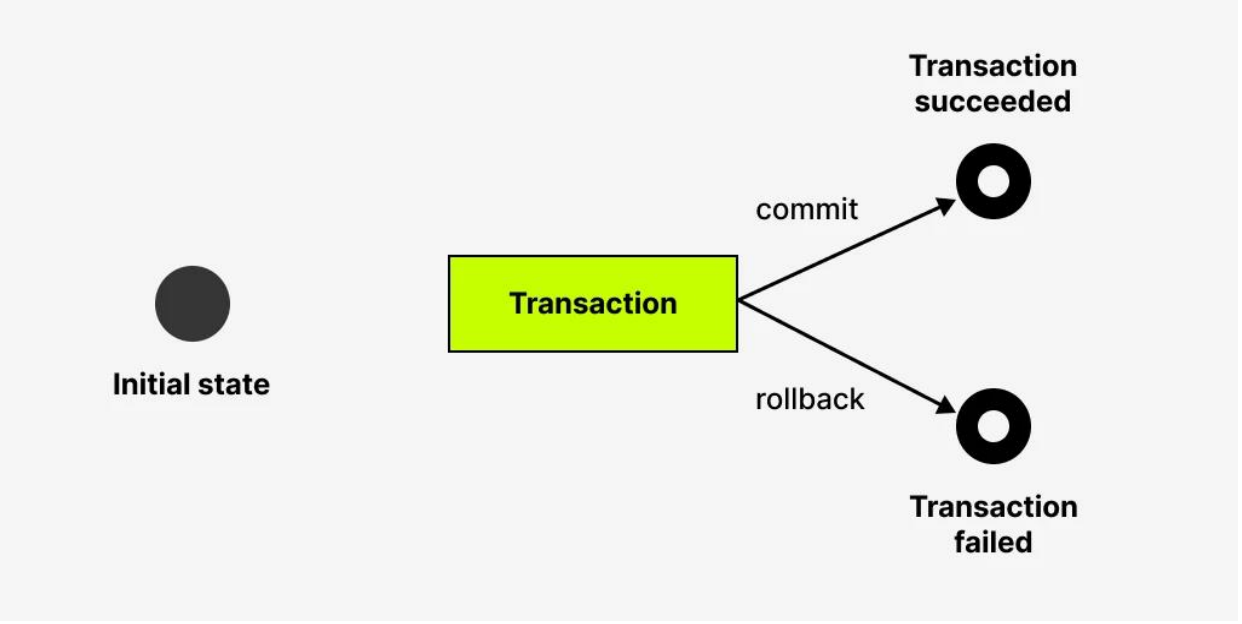
Начиная с версии Spring 4.3 аннотацию *@Autowired* можно опустить, если у класса всего один конструктор. О том, что в конструкторе надо внедрить бин, фреймворк догадается сам

Транзакции

Транзакция — это набор операций по работе с базой данных (БД), объединенных в одну атомарную пачку

Есть три команды:

1) **begin/start** — выполняется перед началом логической группы действий
2) **commit** — выполняется после группы действий транзакции
3) **rollback** — запускает процесс возврата системы из failed state в initial state



Транзакции

Пример использования транзакции в сервисе проекта:

```
public class UserService {

    @Transactional
    public Long registerUser(User user) {
        // выполнить некоторый SQL, который, например,
        // вставляет пользователя в базу данных
        // и извлекает автогенерированный id
        // userDao.save(user);
        return id;
    }
}
```

В классе-конфигурации необходимо указать аннотацию **@EnableTransactionManagement**
Если транзакция не изменяет данные, рекомендуется указывать **@Transactional(readOnly = true)**

Создание проекта: проектирование базы данных

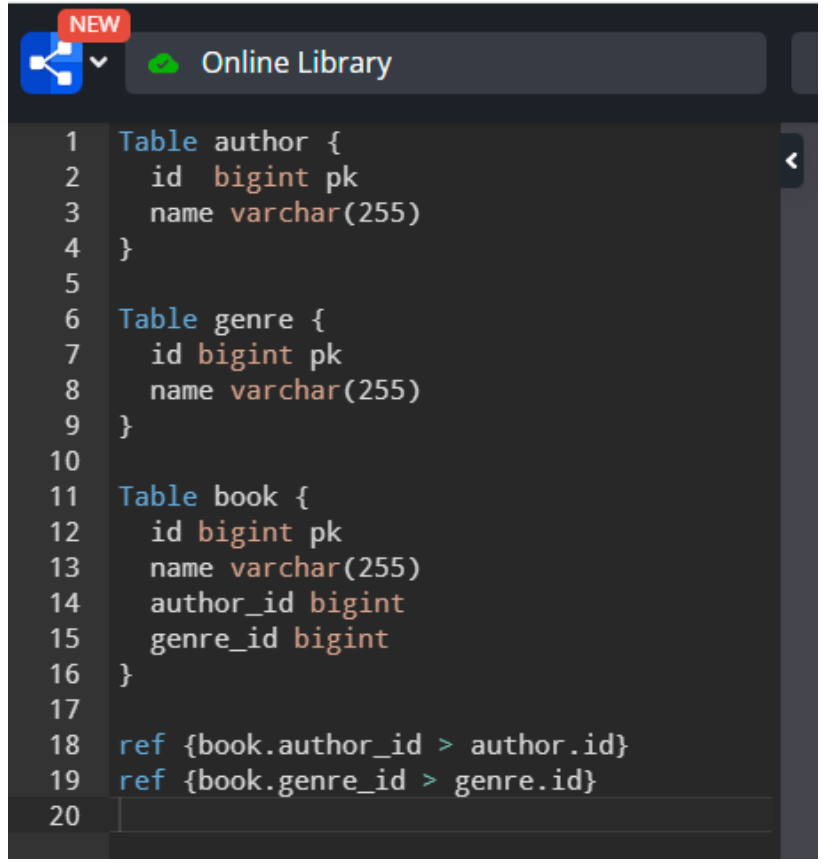
Зарегистрироваться в сервисе https://dbdiagram.io/

1. **Создать модель** таблиц, определить типы полей, полей

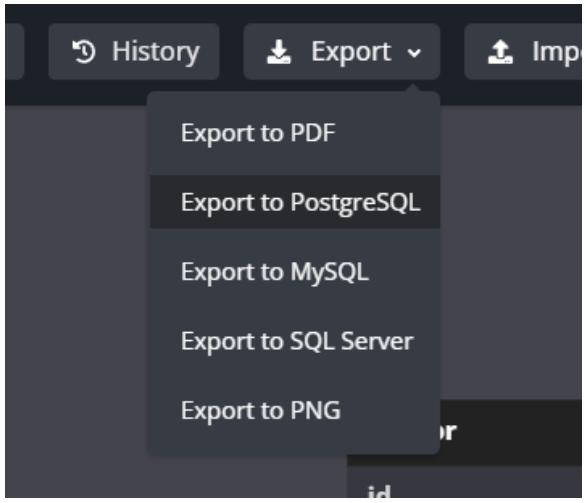
Поля H2 http://www.h2database.com/html/datatypes.html

2. **Выгрузить SQL запрос***:

1



2



*Сформированный SQL запрос затем можно использовать при проектировании Entity-классов

Создание проекта: регистрация репозитория на GitHub

Регистрация репозитория

1. **Зарегистрироваться на GitHub**
2. **Создать репозиторий проекта**
3. **Клонировать репозиторий GitHub на локальный компьютер**

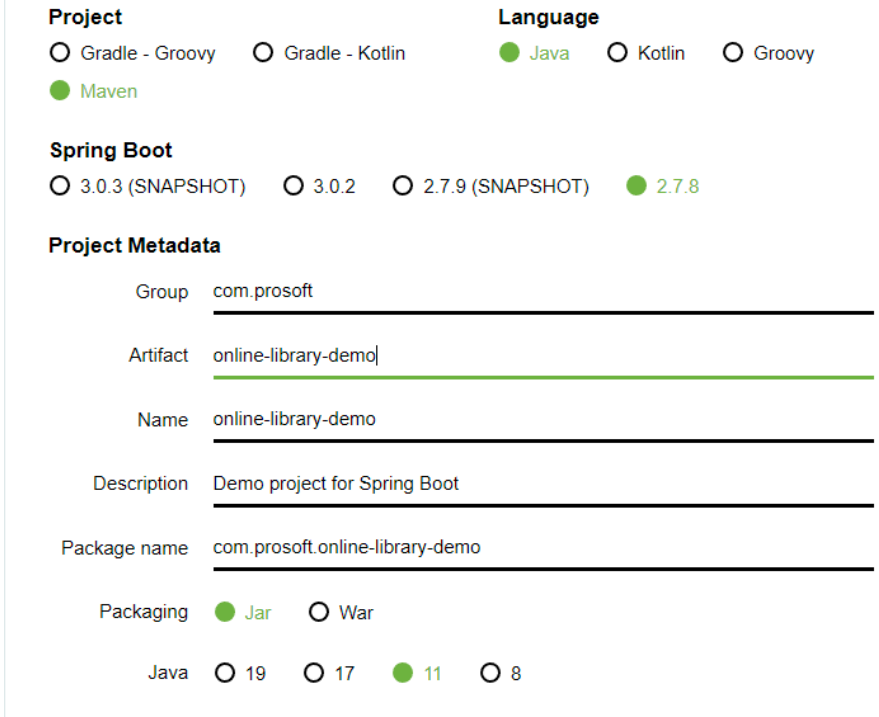
Выполнить пункты с 1-11 инструкции: https://disk.yandex.ru/i/pmAZBG_NEnC5Sw

При необходимости использовать **памятку для работы с Git**: https://disk.yandex.ru/i/C0p8RZpi5UhByw

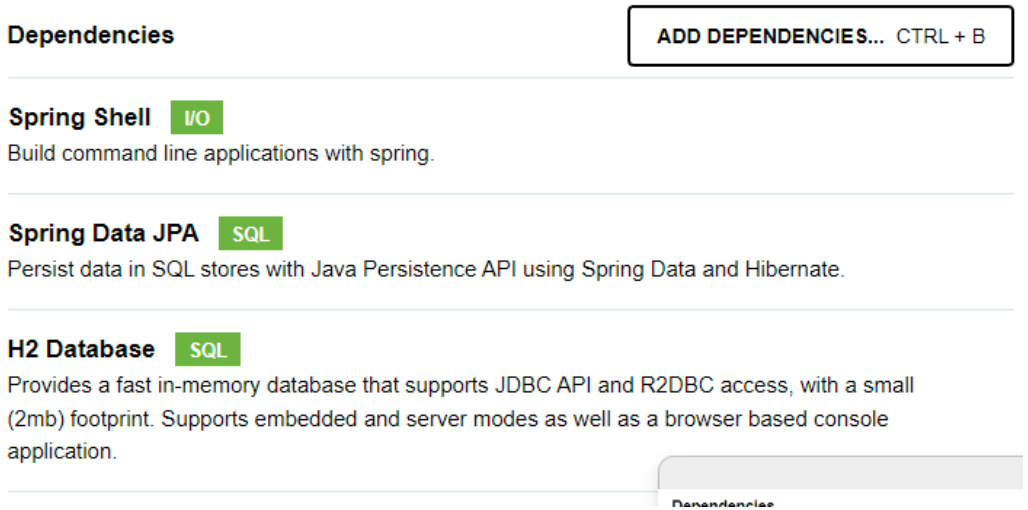
Создание проекта: Spring Initializr

Зайти на сервис https://start.spring.io/

1 **задать параметры и название проекта:**



2 **выбрать зависимости:**

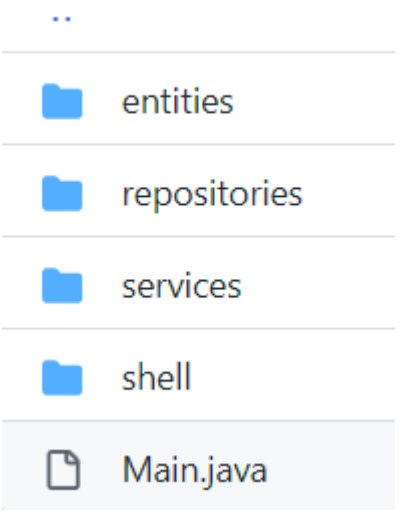


3 **развернуть проект в локальном репозитории**

Создание проекта: работа в IntelliJ IDEA

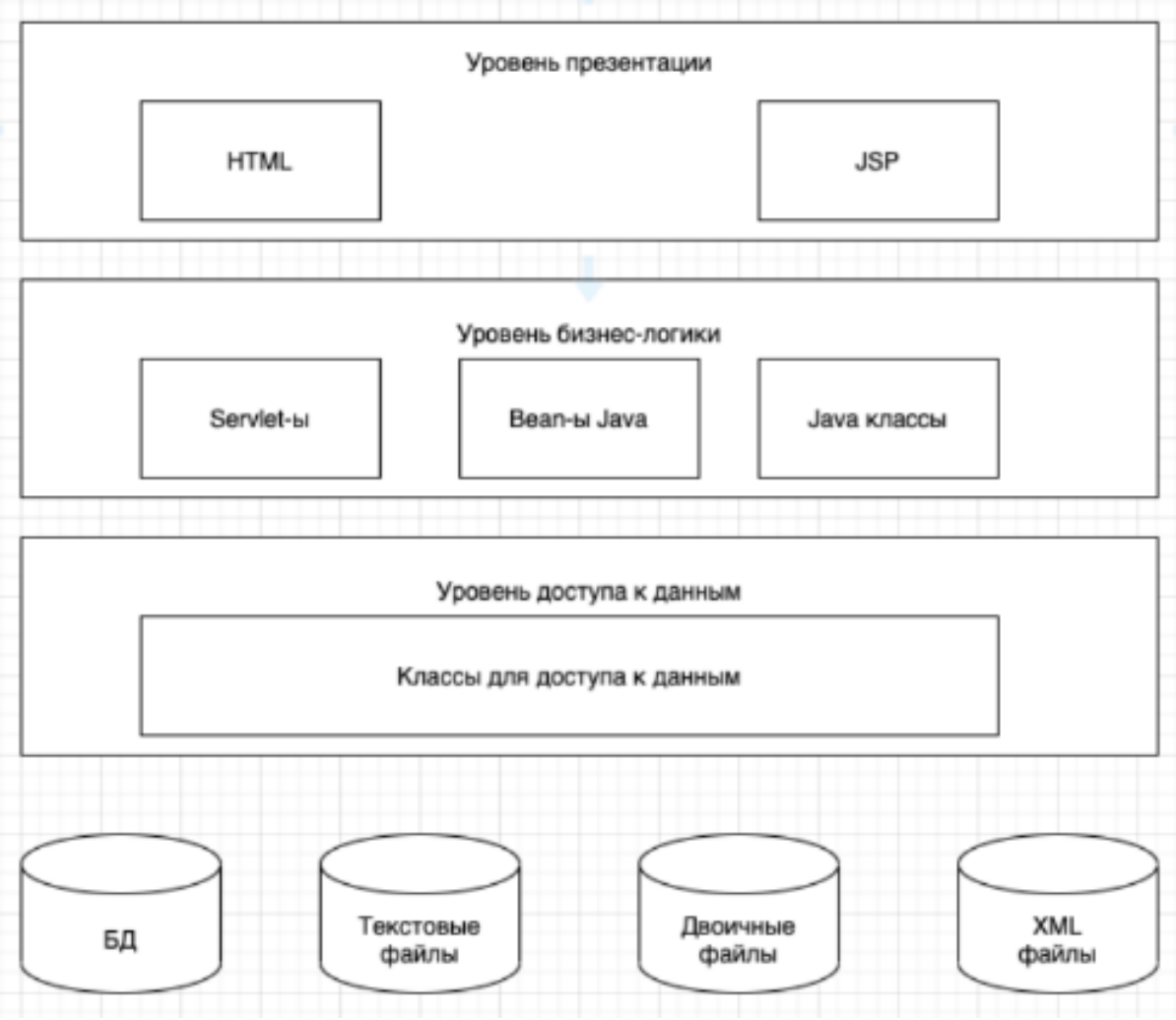
Открыть проект в IntelliJ IDEA:

1) **Переименовать класс Main**
2) **Настроить файл** .gitignore
3) **Создать и настроить в src/main/resources/** файлы **data.sql** и **schema.sql** см. пример https://bit.ly/40e12fj
4) **Настроить файл application.properties**
5) **Создать пакет shell и класс AppEventsCommands внутри которого создать метод вызова консоли H2**
6) **Создать структуру проекта:** entities, repositories, services, shell
7) **Создать в пакете entities классы сущностей**
8) **Создать в пакете repositories классы репозиториев**
9) **Создать в пакете services классы логики**
10) **Создать в пакете shell класс и методы работы с консолью**



Архитектура web-приложения

Схема архитектуры web:



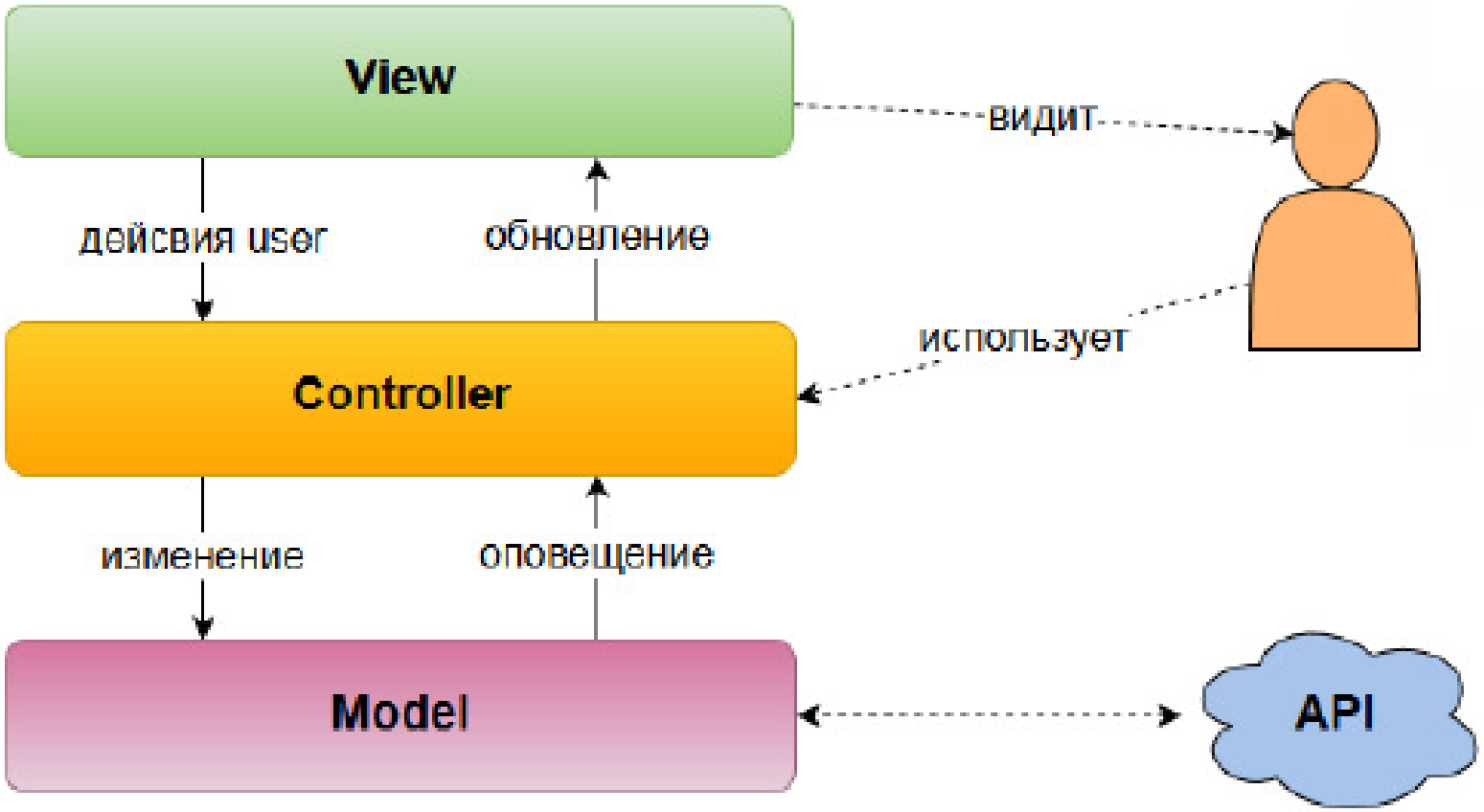
Model View Controller

Model View Controller (MVC) — паттерн, используемый в разработке для отделения бизнес-логики приложения от пользовательского интерфейса, паттерн MVC имеет три уровня:

1) **Модель** определяет бизнес-уровень приложения

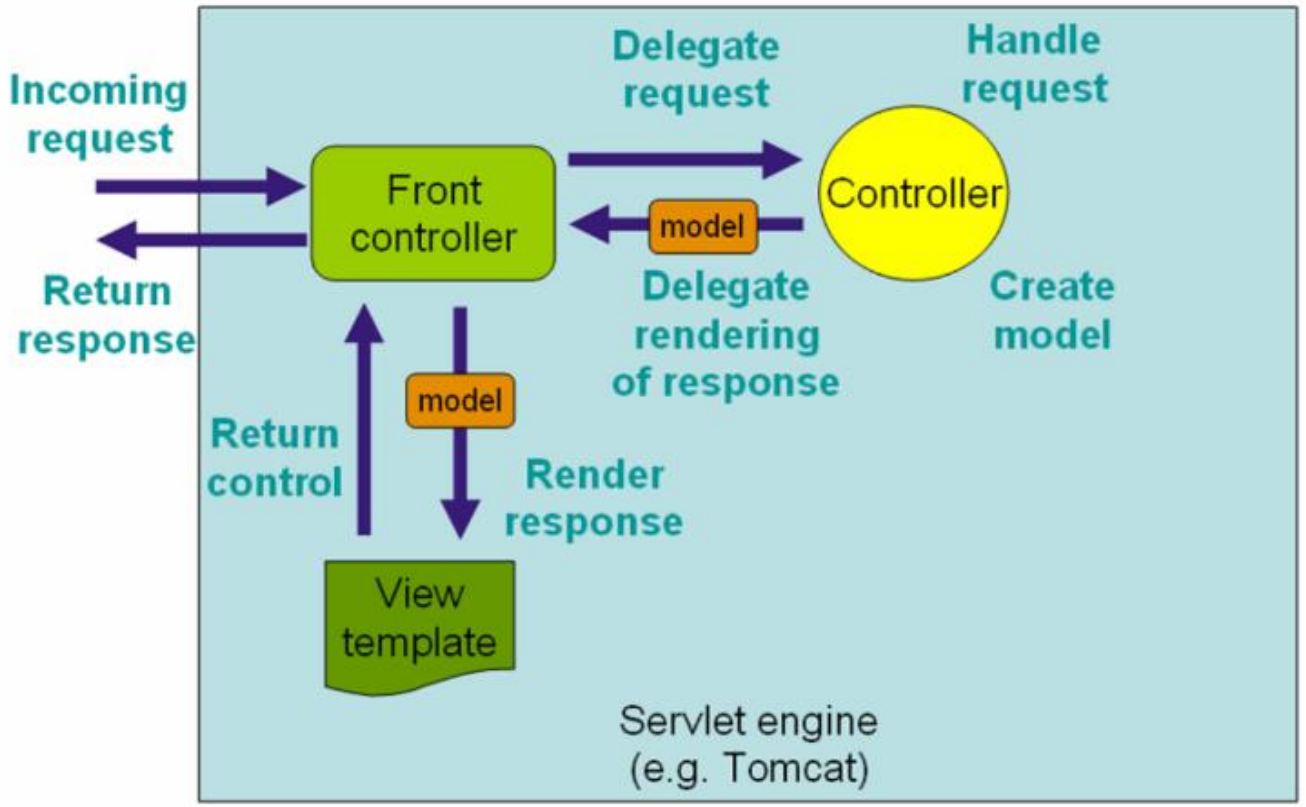
2) **Контроллер** управляет потоком приложения

3) **Представление** определяет уровень визуализации



Spring MVC

Модель Spring MVC



Thymeleaf

Thymeleaf — современный серверный механизм Java-шаблонов, который заменил JSP

Thymeleaf и JSP

Thymeleaf Template

```
<!DOCTYPE HTML>
<html

xmlns:th="http://www.thymeleaf.org">

...

<span th:text="Hello, ${name}">
    Hello, Username
</span>
```

page.jsp (JSP)

```
<%@page ...>
<%@taglib ...>

...

<span>
    Hello, ${name}
</span>
```

Thymeleaf

- Простая конструкция вывода переменной на html-странице:

```
<p th:text=" 'Authors ' + ${authors.size()}">Authors _</p>
```

Результат для коллекции из трех авторов:

Authors 3

Thymeleaf

Конструкция для вывода коллекции на html-странице:

```
<tr th:each="author : ${authors}">
        <td th:text="${author.getId()}"></td>
        <td th:text="${author.getFullName()}"></td>
        </br>
 </tr>
```

Результат для коллекции из трех авторов:

1 John Bunyan
2 Gianni Rodari
3 Daniel Defoe

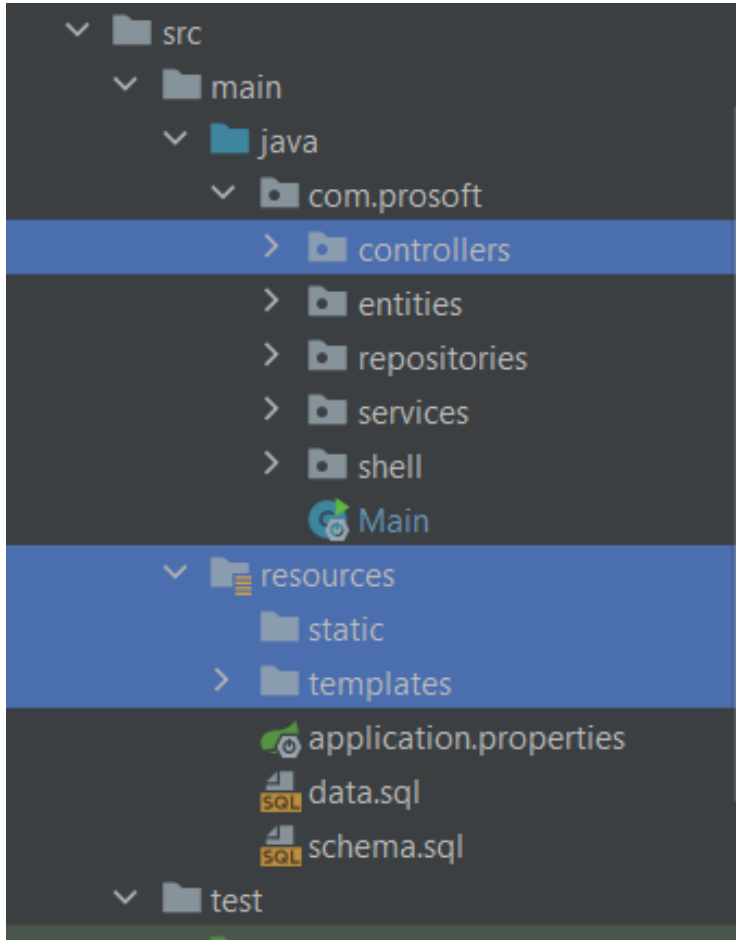
Как сделать web-приложение?

- **Добавить зависимости в pom.xml**

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Как сделать web-приложение?

- **Добавить packadge controllers и в resources: static, templates**



Как сделать web-приложение?

Добавить классический контроллер для доменной модели:

```
@Controller
public class AuthorController {
    2 usages
    private final AuthorService authorService;

    no usages
    @Autowired
    public AuthorController(AuthorService authorService) { this.authorService = authorService; }

    no usages
    @GetMapping("/authors")
    public String booksPage(Model model) {
        List<Author> authorList = authorService.getAllAuthors();
        model.addAttribute( attributeName: "authors", authorList);
        return "authors";
    }
}
```

Как сделать web-приложение?

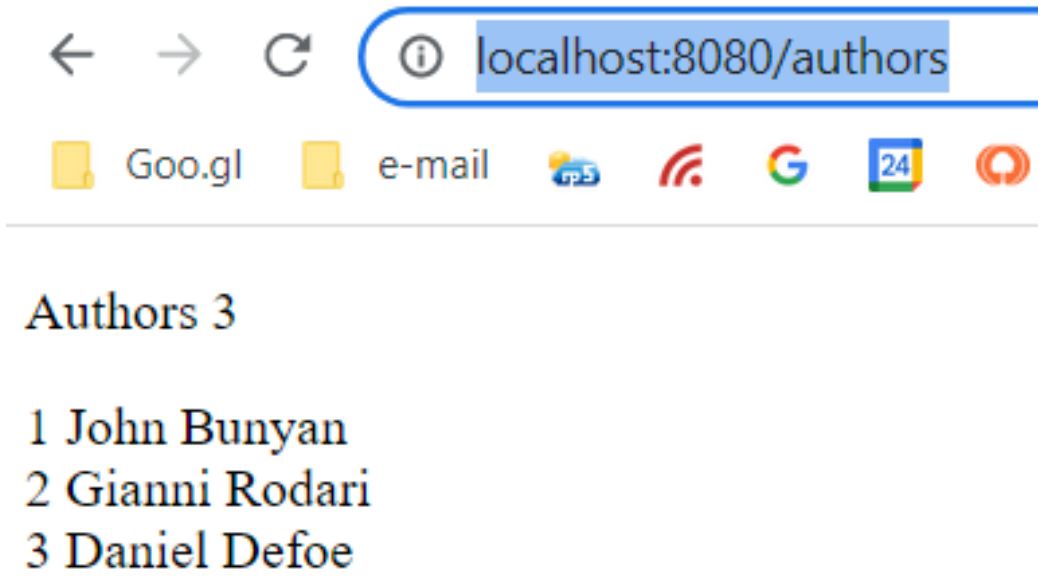
Настроить html страницу:

authors.html

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html">
<head>
    <meta charset="UTF-8">
    <title>Authors</title>
</head>
<body>

<p th:text="'Authors ' + ${authors.size()}">Authors _</p>

<tr th:each="author : ${authors}">
    <td th:text="${author.getId()}"></td>
    <td th:text="${author.getFullName()}"></td>
    </br>
</tr>

</body>
</html>
```

Как сделать web-приложение?

- **Запустить приложение и вызвать страницу по адресу:** http://localhost:8080/authors



Источники

- **Презентация SOLID** https://ppt-online.org/543101

Принципы SOLID в картинках https://habr.com/ru/company/productivity_inside/blog/505430/

Работа с Git https://disk.yandex.ru/i/C0p8RZpi5UhByw

H2 Database Engine https://metanit.com/java/tutorial/8.5.php

Поля H2 http://www.h2database.com/html/datatypes.html

Ключи SQL http://bit.ly/3YgnzpN

Apache Maven — основы https://habr.com/ru/post/77382/

Инверсия управления - Inversion of Control (IoC) http://bit.ly/3Hu0JUN

Spring Data JPA https://habr.com/ru/post/435114/

Источники

- **Что такое Git?** https://www.atlassian.com/ru/git/tutorials/what-is-git

Spring Data JPA. Работа с БД http://bit.ly/3X0GWSI

Database Access (JPA) http://bit.ly/3X0kCbY

Аннотации Java для работы с базой данных https://bit.ly/3Y0OEO9

Транзакции при работе с базой данных http://bit.ly/3X9nnIc

Управление транзакциями в Spring: @Transactional в деталях https://habr.com/ru/post/682362/

Способы внедрения зависимостей (Dependency Injection) в Spring https://bit.ly/3wW9S3P

Spring Framework Guide https://bit.ly/3Yk0rXp

Учебник: Использование Thymeleaf https://habr.com/ru/post/350862/