Java Pro 7.12

TEL-RAN
by Starta Institute

Хранение данных в приложении

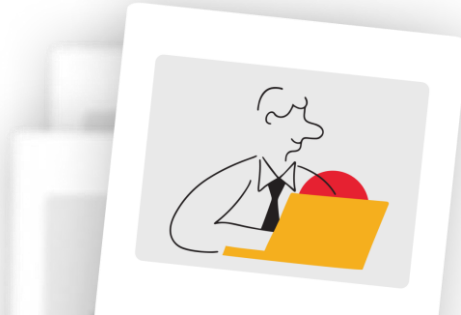
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

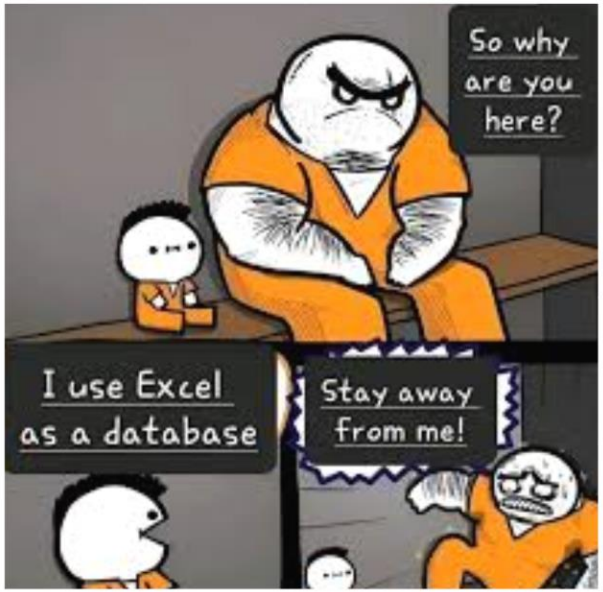
- CRUDучись
- Не JDBCись
- Инъекции зла
- Под сокращение

Проблема

TEL-RAN
by Starta Institute

Хранить данные в приложении – ненадёжно, потому что приложение хранит данные в оперативной памяти и при перезапуске приложения в случае ошибки или аппаратных причин данные будут утеряны. Приложение может записать данные в файл на жёсткий диск, но современные приложения работают с огромными массивами данных, хранить которые в виде обычных файлов крайне неудобно – приходится выдумывать структуру хранилища, способы записи и чтения, правила доступа к диску, способы резервирования и т.д. Всё это давно уже реализовано в системах управления базами данных (СУБД) – отдельных программах, с которыми наше приложение может взаимодействовать через API.

Каким образом можно подготовить Spring-приложение к работе с СУБД?

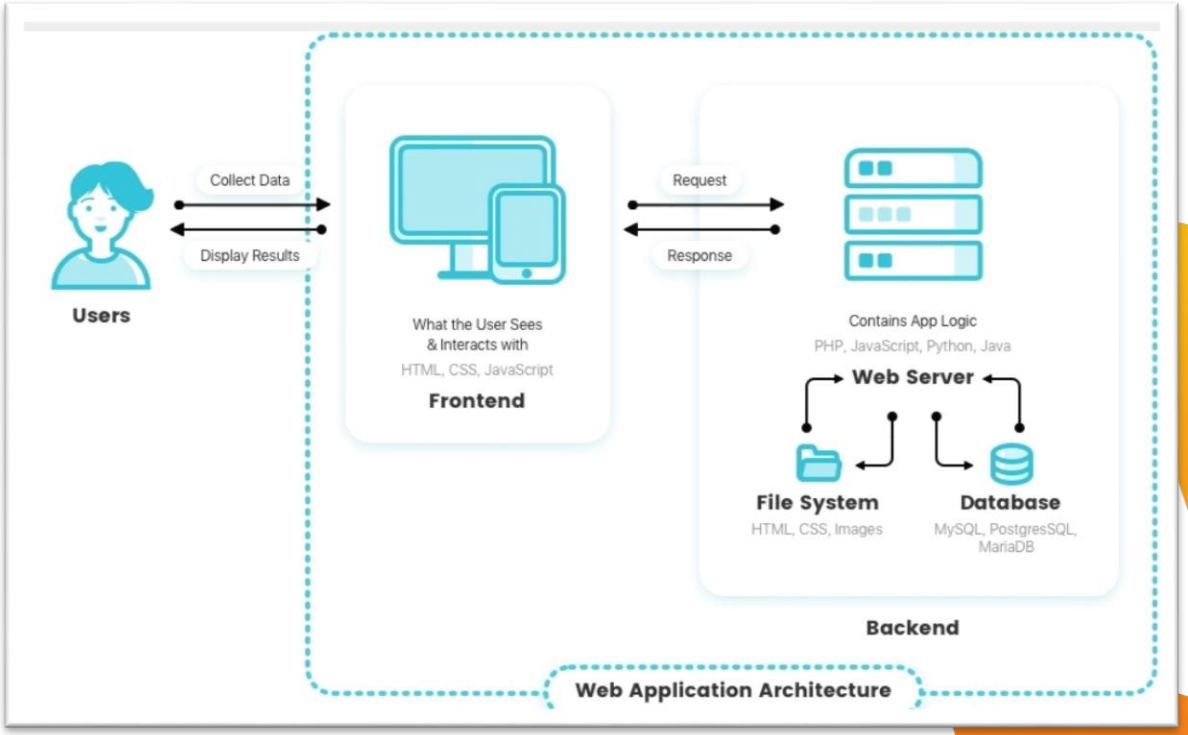


CRUDучись

Структура web-приложения с БД

Структура web-приложения предполагает, что с СУБД взаимодействует backend-часть.

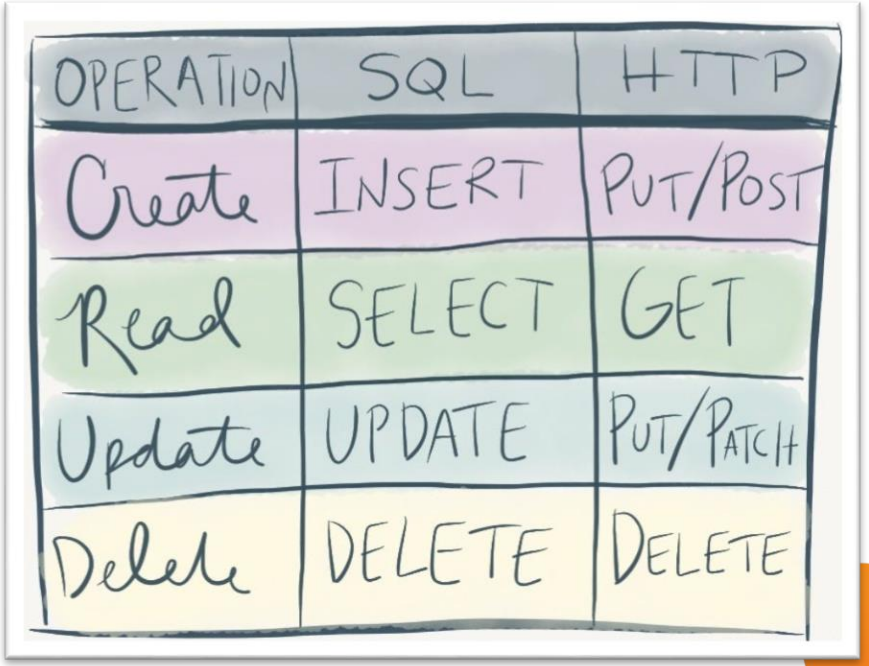
Можно говорить о сквозном соответствии запросов к API backend-приложения и запросов к СУБД.



CRUDучись

Операции с данными

Ранее мы говорили, что основные операции с данными можно описать принципом CRUD (Create, Read, Update, Delete). Согласно этим принципам http-запросы соответствуют действиям с БД. Например, для реляционных БД:

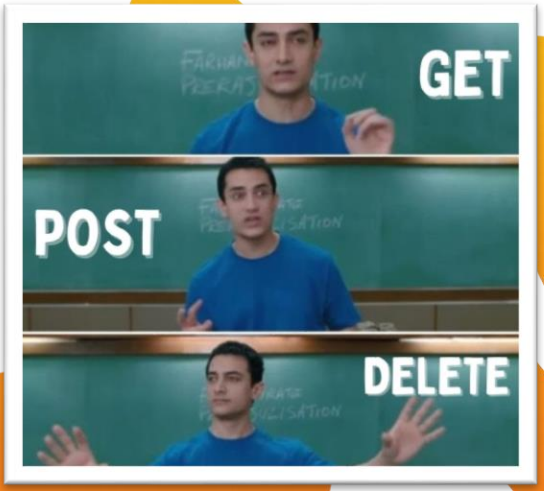


CRUDучись

CRUD API

Существуют некоторые правила составления API приложения для соответствия CRUD. Для сущности User:

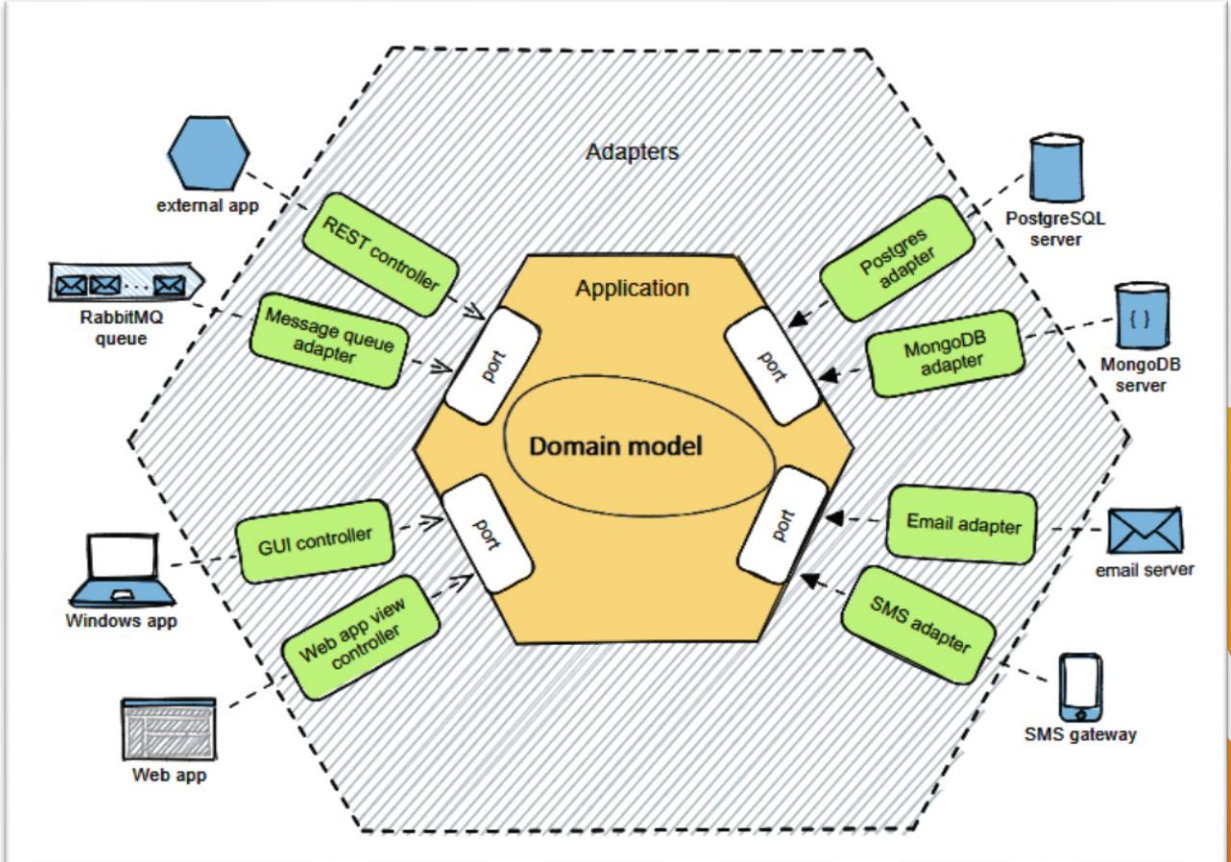
- GET /users - Получаем все записи сущностей user (READ)
- POST /users - Создаем новую запись user (CREATE)
- GET /users/new - Получаем HTML-форму для создания user
- GET /users/{id}/edit - Получаем HTML-форму для редактирования user
- GET /users/{id} - Получаем одну запись user (READ)
- PATCH /users/{id} - Обновляем запись user (UPDATE)
- DELETE /users/{id} - Удаляем запись user (DELETE)



CRUDучись

Изоляция приложения

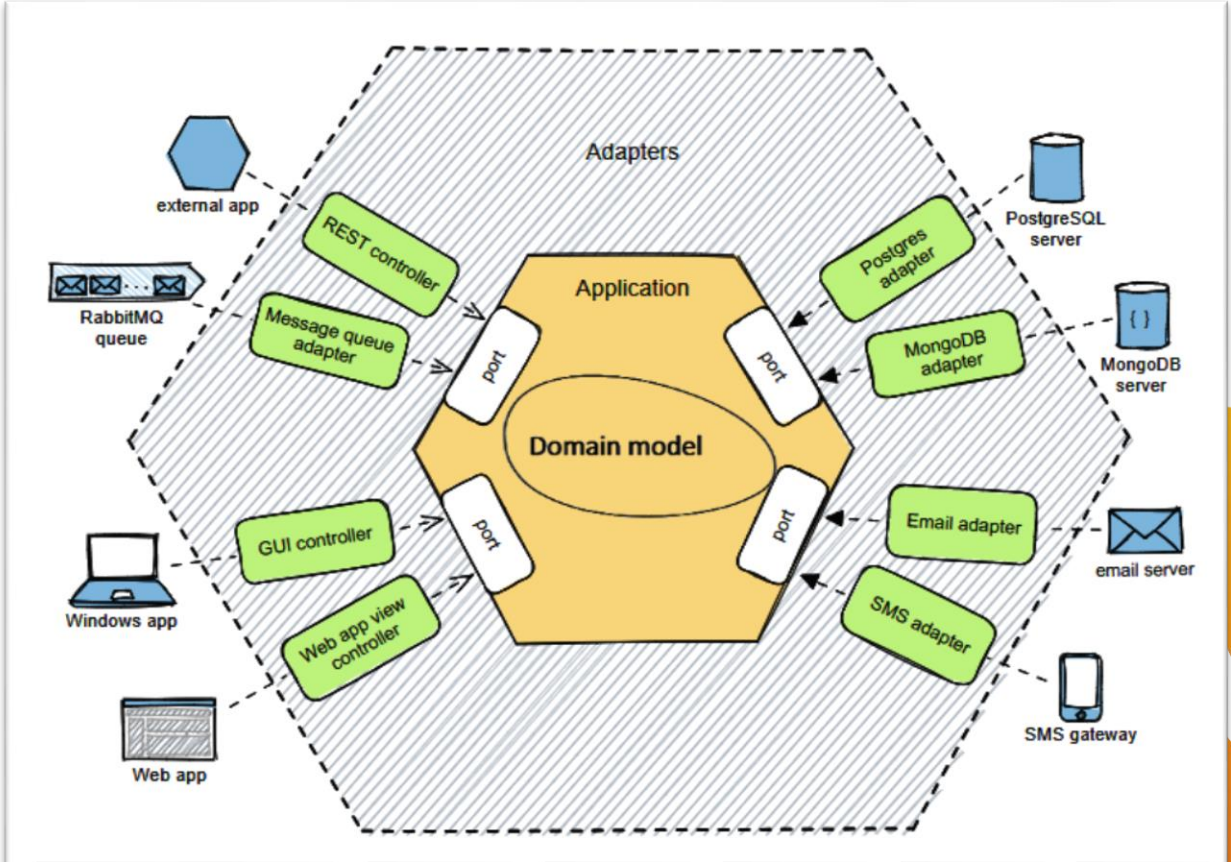
Ранее мы говорили об изоляции приложения от внешних технологий с помощью адаптеров, которые представляют собой классы, которые имплементируют интерфейсы, которые предоставляют порты приложения.



CRUDучись

Изоляция приложения

Но для обмена данными с внешними системами потребуются также дополнительные классы: внутри доменной модели есть сущности бизнес-логики. Их задача – содержать данные бизнеса и следовать логике, которую требует бизнес. На этом их единая ответственность (см. **S**OLID) заканчивается.

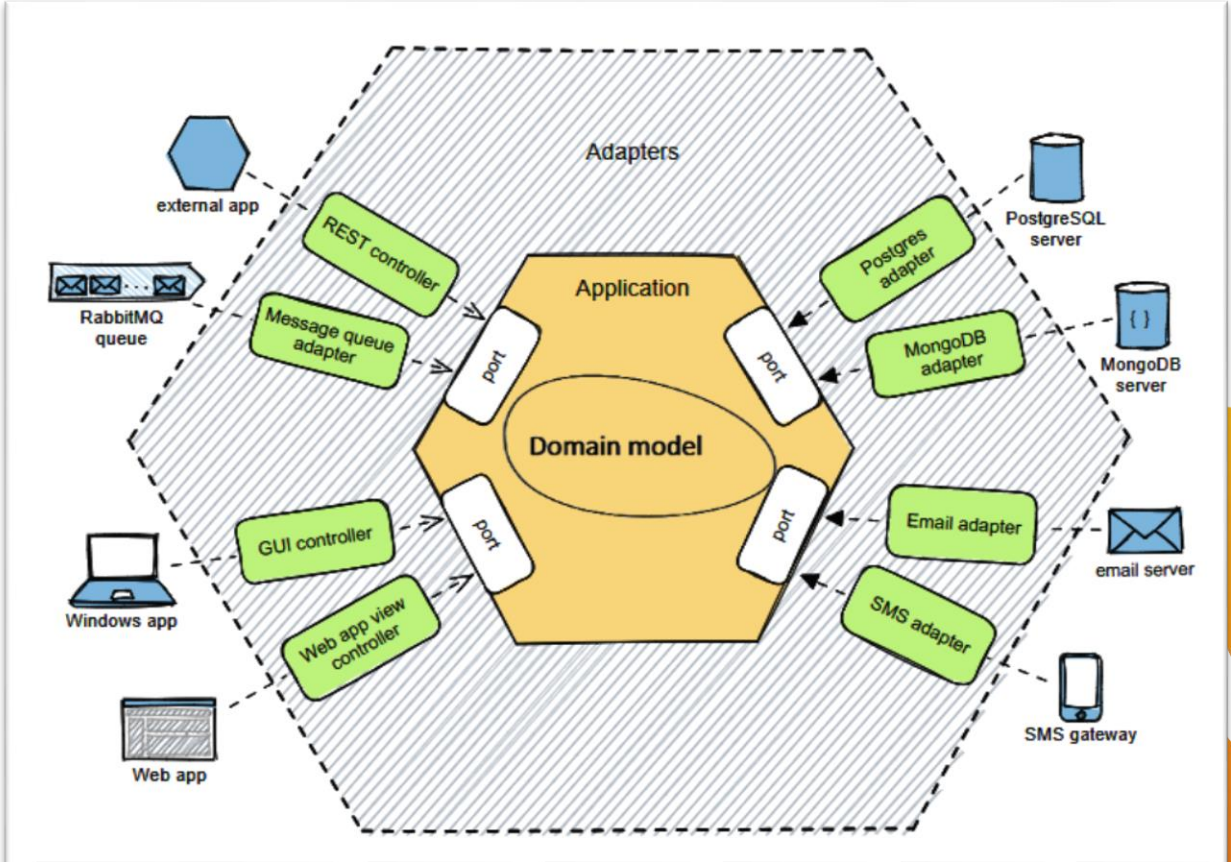


CRUDучись

Изоляция приложения

Это означает, что они не обязаны быть удобными для пересылки по сети или для записи в БД.

Для того, чтобы отделить бизнес-сущности от объектов, пересылаемых по сети и объектов, взаимодействующих с БД, используются паттерны проектирования **DTO** и **DAO**.



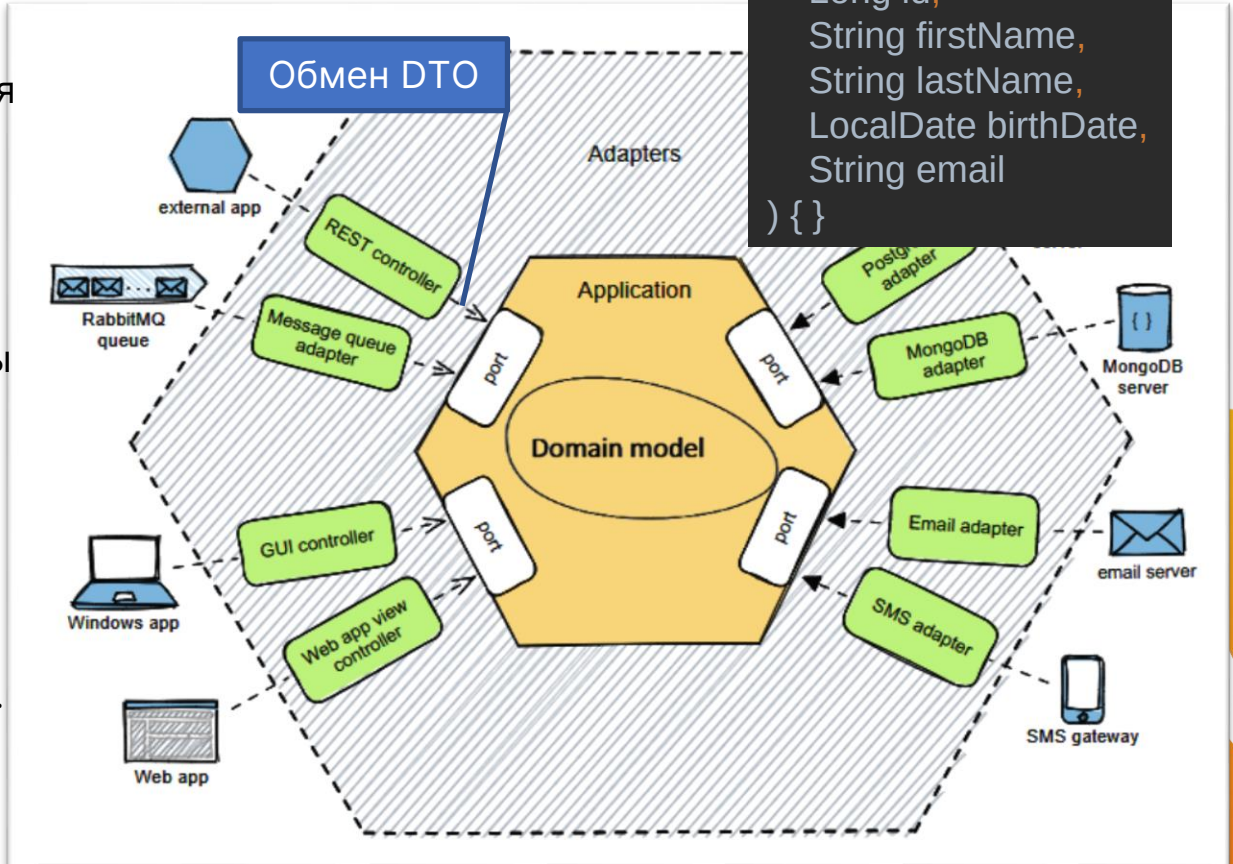
CRUDучись

Изоляция приложения

DTO (*Data Transfer Object*) – объекты, предназначенные для передачи между отдельными слоями приложения (между Service и Controller). Controller подготавливает такие объекты для отправки или восстанавливает из принятого сообщения.

DTO содержат приватные поля, геттеры и конструкторы. Могут содержать сеттеры и статические методы.

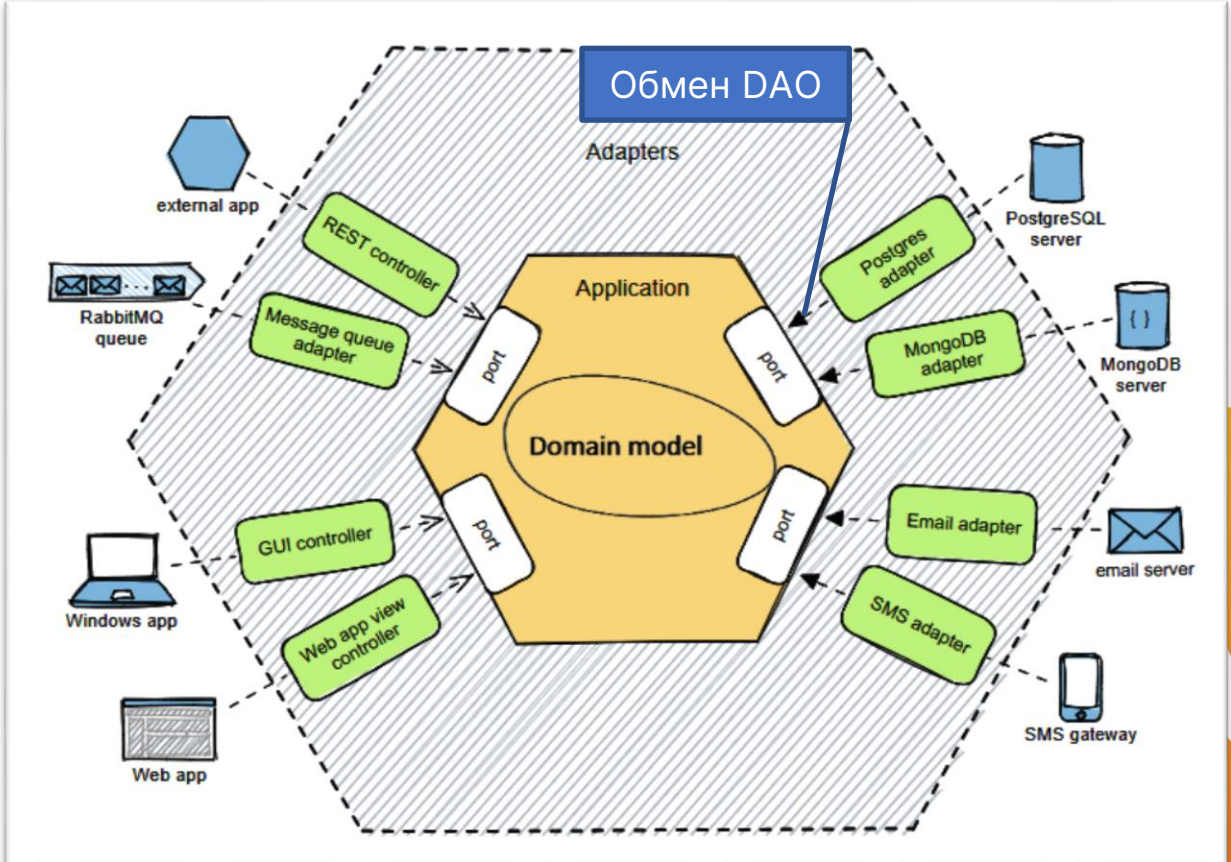
```
public record UserDto(
    Long id,
    String firstName,
    String lastName,
    LocalDate birthDate,
    String email
) { }
```



CRUDучись

Изоляция приложения

DAO (*Data Access Object*) – объекты, помогающие отделить бизнес-логику от слоя персистентности (хранения данных). *DAO* предоставляют методы для вставки, удаления, обновления и поиска данных в БД (CRUD). *DAO* содержат связи полей класса со столбцами таблиц реляционной БД.



CRUDучись

Изоляция приложения

DAO (*Data Access Object*) – объекты, помогающие отделить бизнес-логику от слоя персистентности (хранения данных). *DAO* предоставляют методы для вставки, удаления, обновления и поиска данных в БД (CRUD). *DAO* содержат связи полей класса со столбцами таблиц реляционной БД.

```
public interface UserDao {
    void create(User user);
    User read(Long id);
    void update(User user);
    void delete(Long id);
}

public class UserDaoImpl implements UserDao {
    @Override
    public void create(User user) {
        // save user to storage
    }
    @Override
    public User read(long id) {
        // read user from storage
    }
    // etc
}
```

CRUDучись

Изоляция приложения

Репозиторий (*Repository*) – класс, являющийся промежуточным обработчиком между приложением и *DAO*. Обычно репозиторий подготавливает данные и отправляет их в *DAO* для сохранения в БД, либо извлекает данные из одного или нескольких *DAO* для передачи в приложение.

В простейшем исполнении репозиторий и DAO совпадают.

CRUDучись

Изоляция приложения

В примере приведён репозиторий, задача которого получать данные пользователя и логи о действиях, выполненных этим пользователем.

```
public interface UserRepository {
    void create(User user);
    User read(Long id);
    void update(User user);
    void delete(Long id);
}

public class UserWithLogs extends User {
    private List<Logs> logs;
}

public class UserRepositoryImpl implements UserRepository {
    private UserDaoImpl userDaoImpl;
    private UserWithLogs userWithLogsImpl;
    @Override
    public UserWithLogs get(Long id) {
        UserWithLogs user = (UserWithLogs) userDaoImpl.read(id);
        Logs logs = logsDaoImpl.read(user.getId());
        user.setLogs(logs);
        return user;
    }
}
```

CRUDучись

Пример использования DTO и DAO

SpringMvcDtoDaoExample.zip

Задание

TEL-RAN
by Starta Institute

Создайте RESTful-приложение для приёма заказов от клиентов кафе. Бэкенд должен

1 отдавать список доступных сегодня блюд (меню)

2 принимать заказ: какие блюда и в каком количестве, комментарии к блюду (например, «Без добавления молока» или «Подать кофе первым» т.д.).

3 методы удаления старых блюд из меню и добавление новых, изменение данных о блюде.

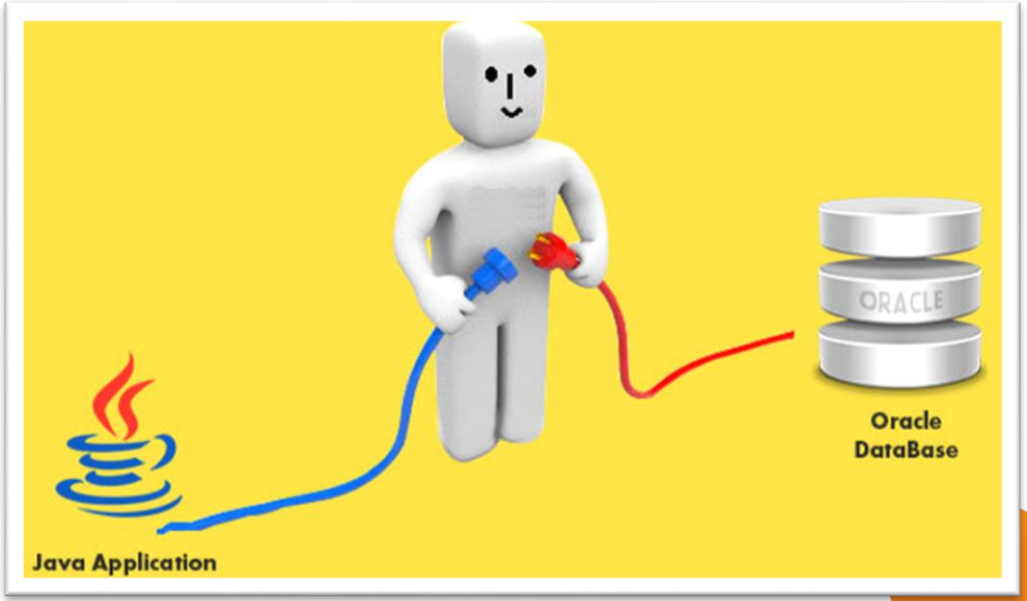
Приложение должно использовать сущности доменной модели, DTO и DAO (используйте внутреннюю коллекцию вместо подключения к БД).

Проблема

TEL-RAN
by Starta Institute

Реляционные базы данных и Java существуют давно, поэтому технология их взаимодействия уже отработана.

Каким образом Java взаимодействует с СУБД?



Не JDBCись

Классы, объекты, таблицы

На языке Java мы оперируем объектами классов и работаем в парадигме ООП. В реляционных БД класс можно соотнести с таблицей, поля класса – с заголовками столбцов, а объекты класса – с записями в таблице.

```java
public class User {
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDate birthDate;
    private String email;
}
```

```java
User tommy = new User(
    1L,
    "Tom",
    "Hardy",
    LocalDate.of(1977, 9, 15),
    null
);
```

id (BIGINT)	first_name (VARCHAR)	last_name (VARCHAR)	birth_date (DATE)	email (VARCHAR)
0	Joel	Edgerton	23.06.1974	joelEd@gmail.com
1	Tom	Hardy	15.09.1977	NULL
…	…	…	…	…

Не JDBCись

Java Database Connectivity

JDBC (*Java Database Connectivity*) – это низкоуровневый API Java, который позволяет выполнять операции с базами данных в Java-приложениях и предоставляет механизм для установления соединения с БД, выполнения SQL-запросов и обработки результатов. Поддерживает практически все СУБД.

При работе с БД с помощью *JDBC* программисту приходиться писать запросы в БД самостоятельно, а результат мапить в объекты Java (например, вызывая сеттеры). *JDBC* является основой для всех остальных технологий (*JPA, Hibernate и Spring Data*), предоставляя низкоуровневый доступ к базам данных.

Не JDBCись

Подключение приложения к БД

СУБД является отдельной программой, которая имеет сетевой адрес, подобно web-приложению.

Для подключения к БД потребуется знать

- сетевой адрес (ip или host);
- сетевой порт;
- название БД;
- логин и пароль пользователя БД;

Кроме того, потребуется JDBC-драйвер, подходящий для БД.

Задание

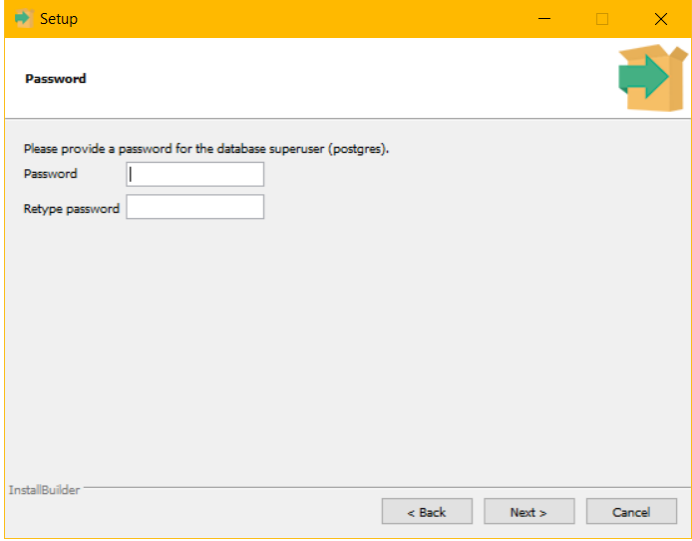
1 Скачайте установщик PostgreSQL для своей ОС (EDB-установщик) https://www.postgresql.org/download/

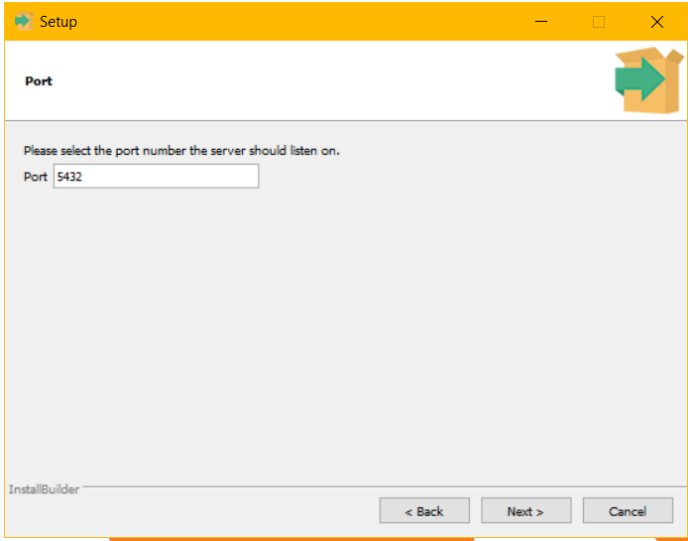
2 Запустите установку БД от имени администратора.

3 При установке в качестве пароля супер-пользователя (postgres) БД укажите 1, оставьте порт по умолчанию (5432).

4 Остальное можно указывать «Далее»

5 После окончания установки уберите галочку в финальном окне.





Задание

6 Postgres предлагает свою графическую оболочку для работы с БД (pgAdmin), которая уже будет установлена вместе с СУБД. Опционально можно установить другу программу - https://dbeaver.io/download/

7 Создайте новую БД *UserDB* с помощью *pgAdmin* или *DBeaver*.

Не JDBCись

Запросы в БД

При работе с СУБД нужно будет понимать, как составляются запросы на языке SQL. Все запросы можно разделить на два основных типа:

- **DDL** (*Data definition language*) – создание и удаление сущностей (баз данных, схем, таблиц и т.д.);
- **DML** (*Data manipulation language*) – операции над данными по *CRUD* ()

```sql
SELECT id, name FROM table_name;

SELECT * FROM table_name;

INSERT INTO User(id, first_name, last_name)
    VALUES (1, 'Tomas', 'Hardy');

UPDATE User SET first_name='Tom' WHERE id=1;

DELETE FROM User WHERE id=1;
```

```sql
DROP DATABASE IF EXISTS postgres;

CREATE DATABASE postgres
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'Russian_Russia.1251'
    LC_CTYPE = 'Russian_Russia.1251'
    LOCALE_PROVIDER = 'libc'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

COMMENT ON DATABASE postgres
    IS 'default administrative connection database';
```

Задание

1 Откройте ранее созданный проект с реализованным паттерном DAO.

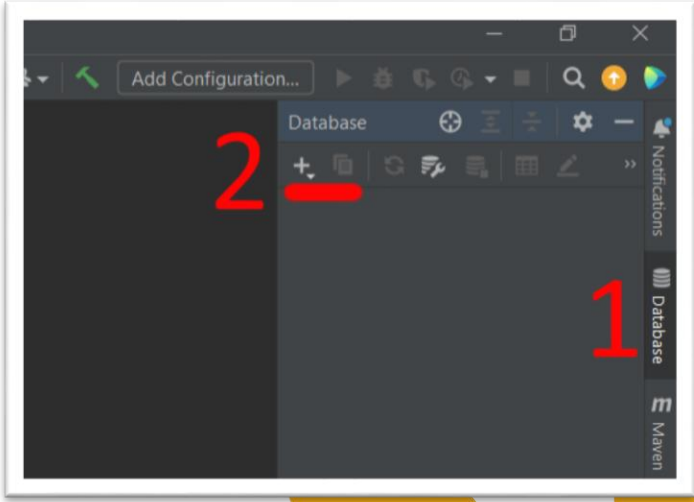
2 Создайте подключение к БД (правая панель Database -> “+” -> в списке выберите PostgreSQL -> в настройках введите параметры и, при необходимости, скачайте драйвер). Нажмите Test Connection. Если показал зелёную галочку, то соединение настроено.

3 Откроется консоль для ввода SQL-запросов. Создайте таблицу users со столбцами, соответствующими полям класса User.

4 Заполните таблицу тремя записями о пользователях.

Пример:

```
INSERT INTO users
values (0, 'Tom', 'Hardy', '1997-09-15', NULL);
```




```
CREATE TABLE users(
    id BIGINT NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    birth_date DATE,
    email VARCHAR(50)
);
```

Не JDBCись

Драйвер JDBC

Каждой СУБД соответствует свой драйвер (подпрограмма), который нужно добавить в зависимости проекта. Таким образом, разработчику не надо знать, как выполнить низкоуровневое подключение к СУБД, т.к. обычно драйвер поставляется самим разработчиком СУБД. При замене одной СУБД на другую достаточно изменить используемый драйвер и приложение сможет работать с новой СУБД без каких-либо изменений.

Не JDBCись

Классы JDBC

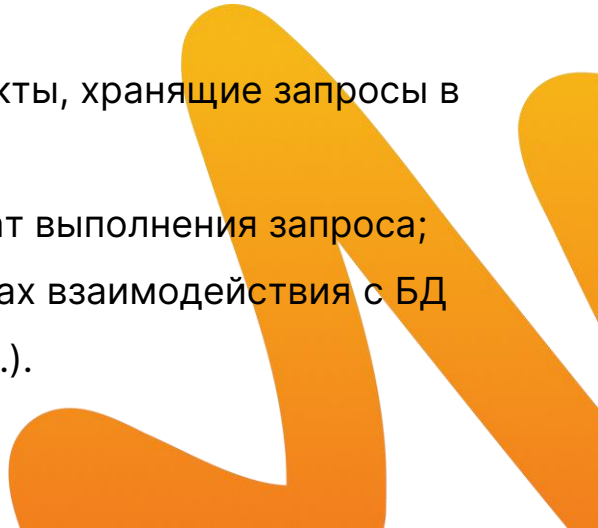
xxx.yyy.Driver (например, *org.postgresql.Driver*) – класс подгружаемого драйвера, обеспечивающего обмен данными с БД. В старых версиях Spring требовалось загружать драйвер с помощью вызова метода *Class.forName()*;

java.sql.Connection – интерфейс, обеспечивающий создание подключения к БД. Обычно такие объекты создаются не через оператор *new*, а с помощью вызова *DriverManager.getConnection(DB_URL, USERNAME, PASSWORD)*;

java.sql.Statement – интерфейс, который имплементируют объекты, хранящие запросы в БД;

java.sql.ResultSet – интерфейс для объектов, хранящих результат выполнения запроса;

java.sql.SQLException – исключение, выбрасываемое при ошибках взаимодействия с БД (при невозможности подключения, неправильных запросах и т.д.).



Не JDBCись

Пример использования JDBC

SpringJDBCExample.zip

Задание

1 Добавьте в проект приложения для кафе зависимость для драйвера, работающего с СУБД https://mvnrepository.com/artifact/org.postgresql/postgresql

2 Создайте подключение к БД (правая панель Database -> “+” -> в списке выберите PostgreSQL -> в настройках введите параметры и, при необходимости, скачайте драйвер). Нажмите Test Connection. Если показал зелёную галочку, то соединение настроено.

3 Создайте БД MenuApp.

4 Создайте подключение к БД в IntelliJ IDEA.

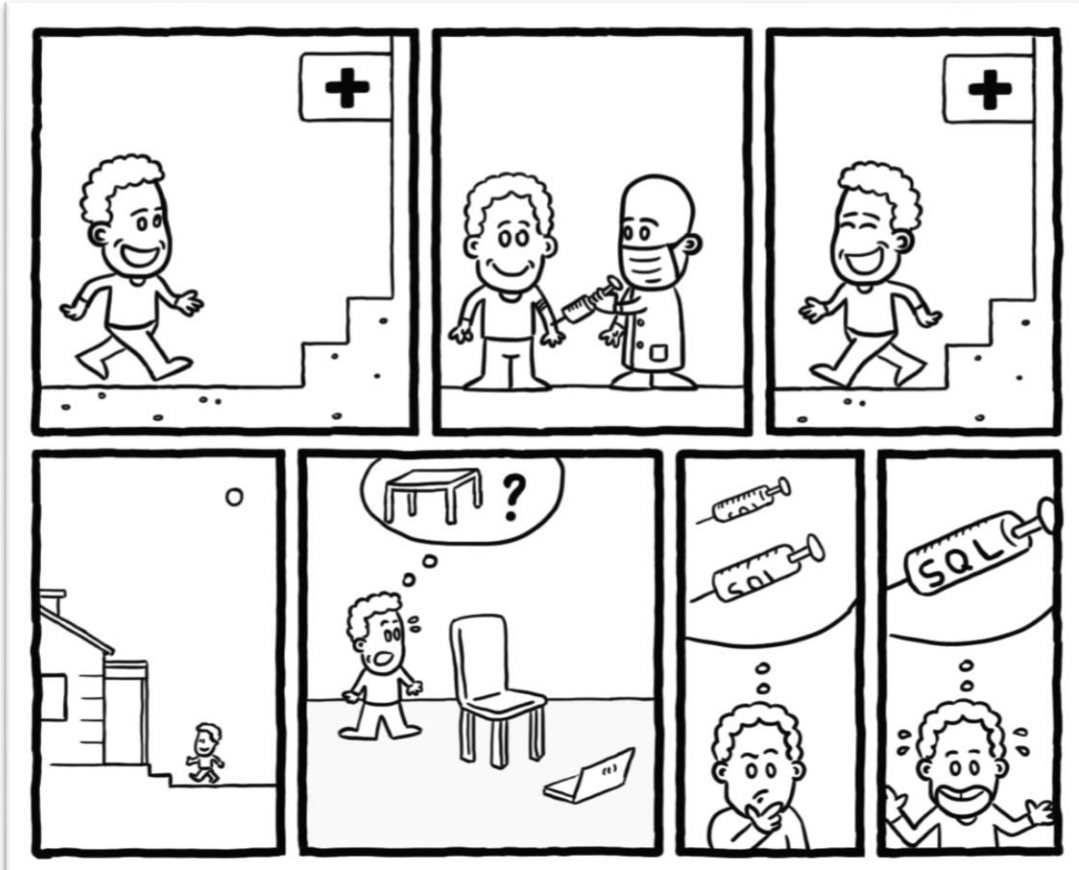
5 После создания откроется консоль для ввода SQL-запросов. Создайте таблицу dishes со столбцами, соответствующими полям класса Dish.

6 Заполните таблицу тремя записями о блюдах.

7 Дополните проект так, чтобы DAO работали с реальной БД.

Проблема

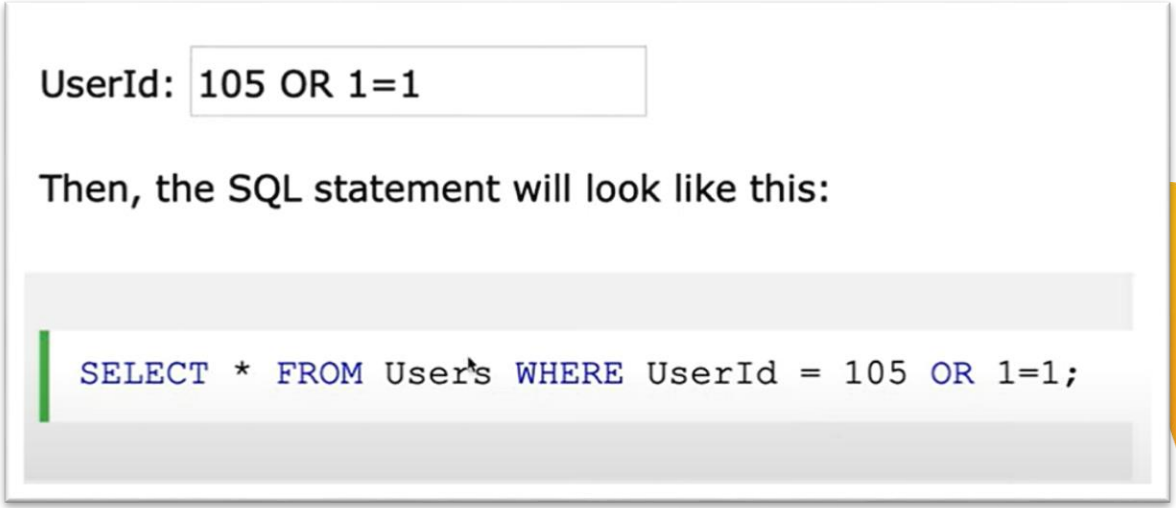
Из-за того, что мы делали конкатенацию запроса с использованием данных, направленных пользователем, есть риск, что злоумышленник сделает SQL-инъекцию в наш запрос. Это может привести к удалению отдельной таблицы (DROP TABLE) или даже всей базы данных (DROP DATABASE) Достаточно написать правильный SQL-запрос в поле формы ввода данных при регистрации.



Проблема

TEL-RAN
by Starta Institute

Другое применение SQL-инъекции – это кража данных из БД. В примере ниже можно получить все записи о пользователях из БД вместо одной.



Как защитить приложение от SQL-инъекций?

Инъекции зла

PreparedStatement

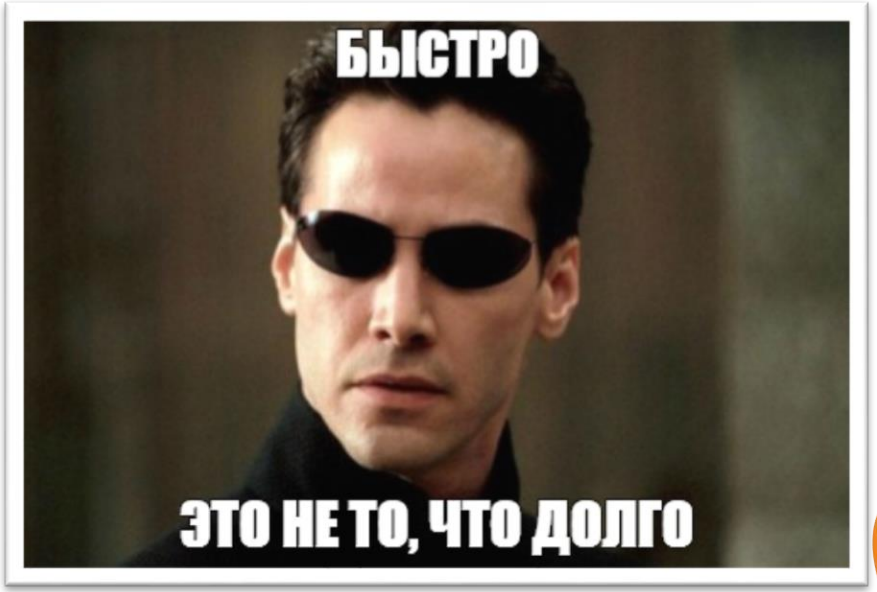
PreparedStatement – это интерфейс для объектов, хранящих запрос в БД, который позволяет создавать запросы без конкатенации (он делает конкатенацию строк за нас), при этом его механизм объединения строк позволяет избежать SQL-инъекций.

```
try {
    PreparedStatement preparedStatement = CONNECTION.prepareStatement(
            "INSERT INTO users (first_name, last_name, birth_date, email) VALUES (?, ?, ?, ?);"
    );
    preparedStatement.setString(1, user.getFirstName());
    preparedStatement.setString(2, user.getLastName());
    preparedStatement.setDate(3, Date.valueOf(user.getBirthDate()));
    preparedStatement.setString(2, user.getEmail());
    preparedStatement.executeUpdate();
} catch (SQLException e) {
    throw new RuntimeException(e);
}
```

Инъекции зла

PreparedStatement

Отдельно стоит отметить, что *PreparedStatement* работает быстрее обычного *Statement* особенно при большом количестве запросов в БД. *PreparedStatement* компилируется один раз и при запросе просто подставляет нужные значения. Объект *Statement* компилируется заново при каждом запросе.



Инъекции зла

Пример использования PreparedStatement

SpringJDBCPreparedStatementExample.zip

Задание

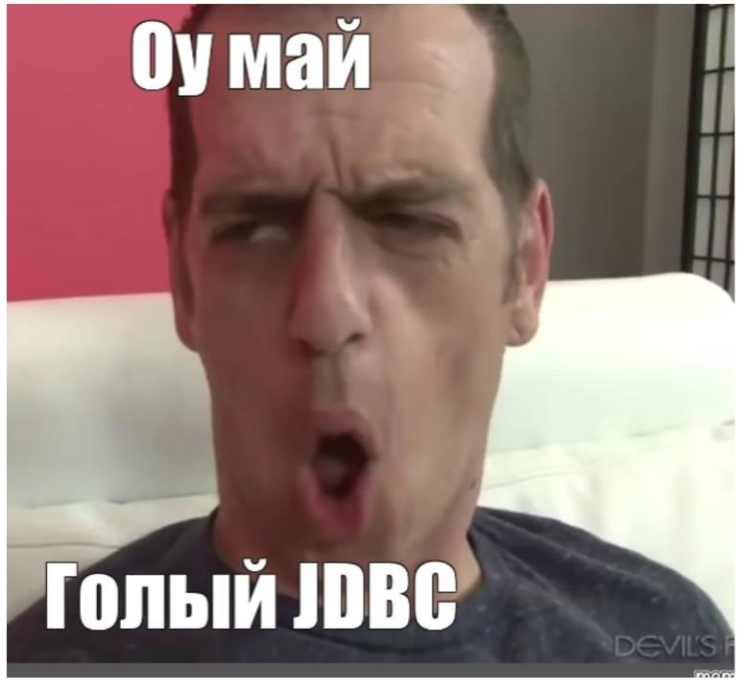
Перепишите DAO приложения для кафе, используя PreparedStatement.

Проблема

TEL-RAN
by Starta Institute

Из-за своей низкоуровневой реализации *JDBC* требует написание большого количества кода:

- создавать *Connection* и *Statement/PreparedStatemant* на каждый запрос;
- обрабатывать результат *ResultSet*, чтобы замапить данны в объект приложения;
- закрывать Connection и *ResultSet* после использования;
- на каждом шаге обрабатывать малоинформативное *SQLException*;
- код дублируется из метода к методу, из *DAO* к *DAO*.



Как сократить эту рутинную работу?

Под сокращение

JdbcTemplate

JdbcTemplate – тонкая обёртка над *JDBC* в составе *Spring Framework*, которая предоставляет некоторые абстракции и берёт часть дел на себя.

JdbcTemplate – центральный класс в базовом пакете *JDBC*:

1 обрабатывает создание и освобождение ресурсов (закрытие соединения);

2 выполняет основные задачи рабочего процесса JDBC (создание и выполнение запросов, извлечение результатов);

3 перехватывает исключения JDBC и преобразует их в общую, более информативную иерархию исключений, определенную в пакете org.springframework.dao.

Под сокращение

JdbcTemplate

При использовании *JdbcTemplate* нужно только реализовать интерфейсы, предоставив им четко определенный контракт.

PreparedStatementCreator и **CallableStatementCreator** - интерфейсы, которые создают запрос и соединение.

RowCallbackHandler – интерфейс, который извлекает значения из каждой строки результирующего набора.

JdbcTemplate может использоваться как реализация *DAO* посредством прямого создания экземпляра со ссылкой на источник данных или быть сконфигурирован в контексте *Spring* и внедрён в *DAO*.

Запросы в виде *SQL*, выданные из *JdbcTemplate*, логируются на уровне *debug*.

Под сокращение

Шаги перехода к JdbcTemplate

1 Добавить зависимость https://mvnrepository.com/artifact/org.springframework/spring-jdbc

2 В конфигурации Spring добавить бин *DataSource* с параметрами подключения к БД.

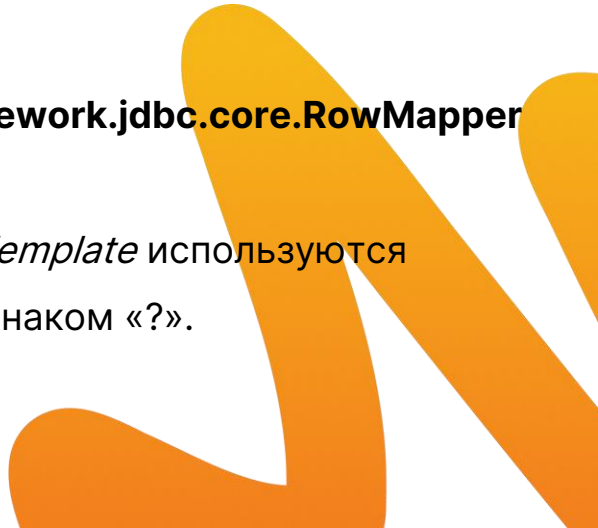
3 В конфигурации Spring добавить бин *JdbcTemplate*, принимающий ранее созданный *DataSource*.

4 Внедрить бин *JdbcTemplate* в *DAO*-классы.

5 В *DAO* удалить ранее написанный код подключения к БД.

6 Создать класс, имплементирующий интерфейс **org.springframework.jdbc.core.RowMapper** для маппинга данных между объектами и таблицами БД.

7 Переписать запросы с использованием *JdbcTemplate.* В *JdbcTemplate* используются *PreparedStatement,* поэтому можно использовать синтаксис со знаком «?».



Под сокращение

Очевидное приемущество

```
private static final String DB_URL = "jdbc:postgresql://localhost:5432/UserDB";
private static final String USERNAME = "postgres";
private static final String PASSWORD = "1";
private static final String DRIVER_CLASS_NAME = "org.postgresql.Driver";
private static final Connection CONNECTION;

static {
    try {
        Class.forName(DRIVER_CLASS_NAME); // Требовалось в старых версиях Spring
        CONNECTION = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
    } catch (ClassNotFoundException e) {
        throw new RuntimeException("Cannot load " + DRIVER_CLASS_NAME, e);
    } catch (SQLException e) {
        throw new RuntimeException("Cannot create connection to " + DB_URL, e);
    }
}

public StatisticsDaoImpl() { }

@Override
public Map<String, Long> getStatistics() {
    try {
        Statement statement = CONNECTION.createStatement();
        ResultSet result = statement.executeQuery("SELECT * FROM client_app_statistics;");
        Map<String, Long> clientAppToCount = new HashMap<>();
        while (result.next()) {
            clientAppToCount.put(
                    result.getString("client_app_name"),
                    result.getLong("client_app_count")
            );
        }
        return clientAppToCount;
    } catch (SQLException e) {
        throw new RuntimeException("Cannot get statistics", e);
    }
}
```

До

```
private final JdbcTemplate jdbcTemplate;

@Autowired
public StatisticsDaoImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

@Override
public List<StatisticsItem> getStatistics() {
    return jdbcTemplate.query("SELECT * FROM client_app_statistics;", new StatisticsMapper());
}
```

После

Под сокращение

Пример использования JdbcTemplate

SpringJDBCTemplateExample.zip

Задание

Перепишите DAO приложения для кафе, используя JdbcTemplate.

TEL-RAN
by Starta Institute

2 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

Напишите RESTful web-приложение для создания, изменения, удаления и просмотра записей школьного журнала (класс, ФИО ученика, предметы и соответствующие им оценки по датам получения). В приложении должен присутствовать слой persistence с DAO. Просмотр оценок можно делать по всем ученикам класса, передав название класса, или по отдельному ученику (передаётся имя класса, и ФИО ученика).

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute