**REST** – это паттерн проектирования web-приложений. Он описывает, как по протоколу HTTP взаимодействовать с сервером для работы с данными (реализует концепцию **CRUD**). Плюс много чего другого.

**DAO** (**Data Access Object**) – это паттерн проектирования, когда логика взаимодействия с БД вынесена в отдельный класс.

Вывод списка в представлении + динамические ссылки на каждого человека:

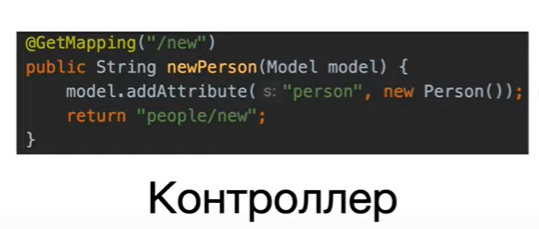
<div th:each=”person : ${people}”>

<a th:href=”@{/people/{id}(id=${person.getId()})}” th:text=”${person.getName()}”></a>

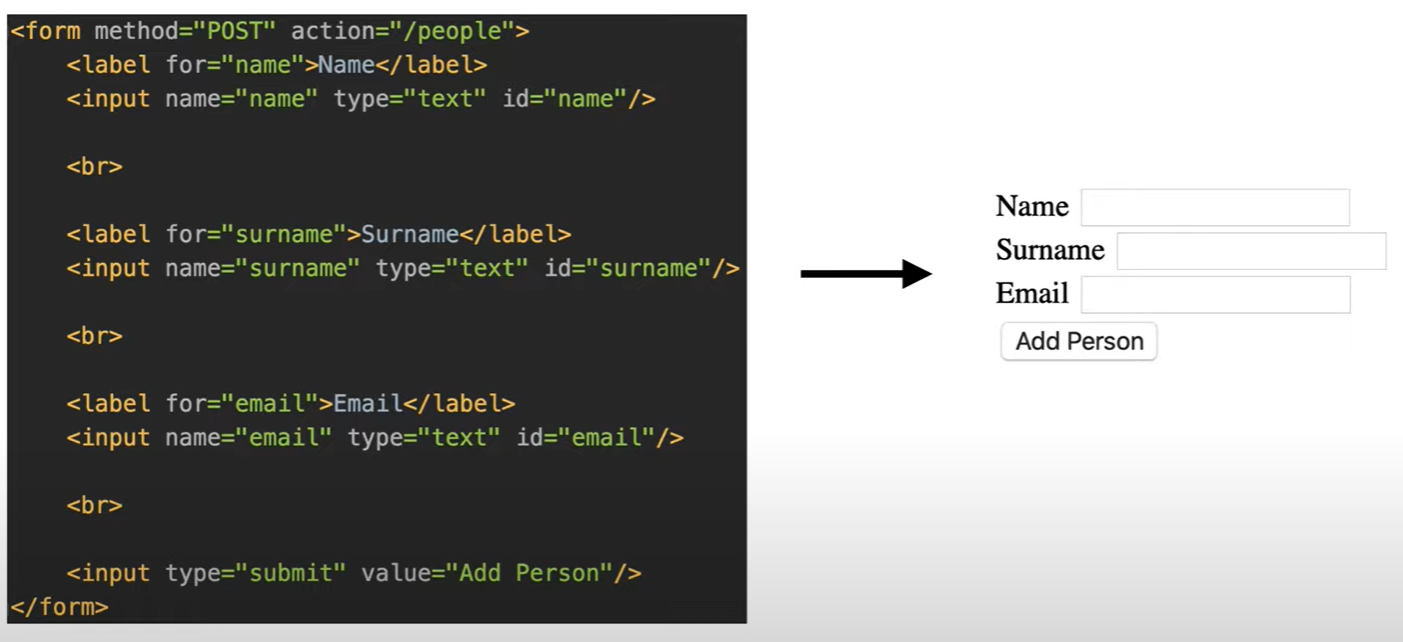
</div>

Создание формы POST-запроса:

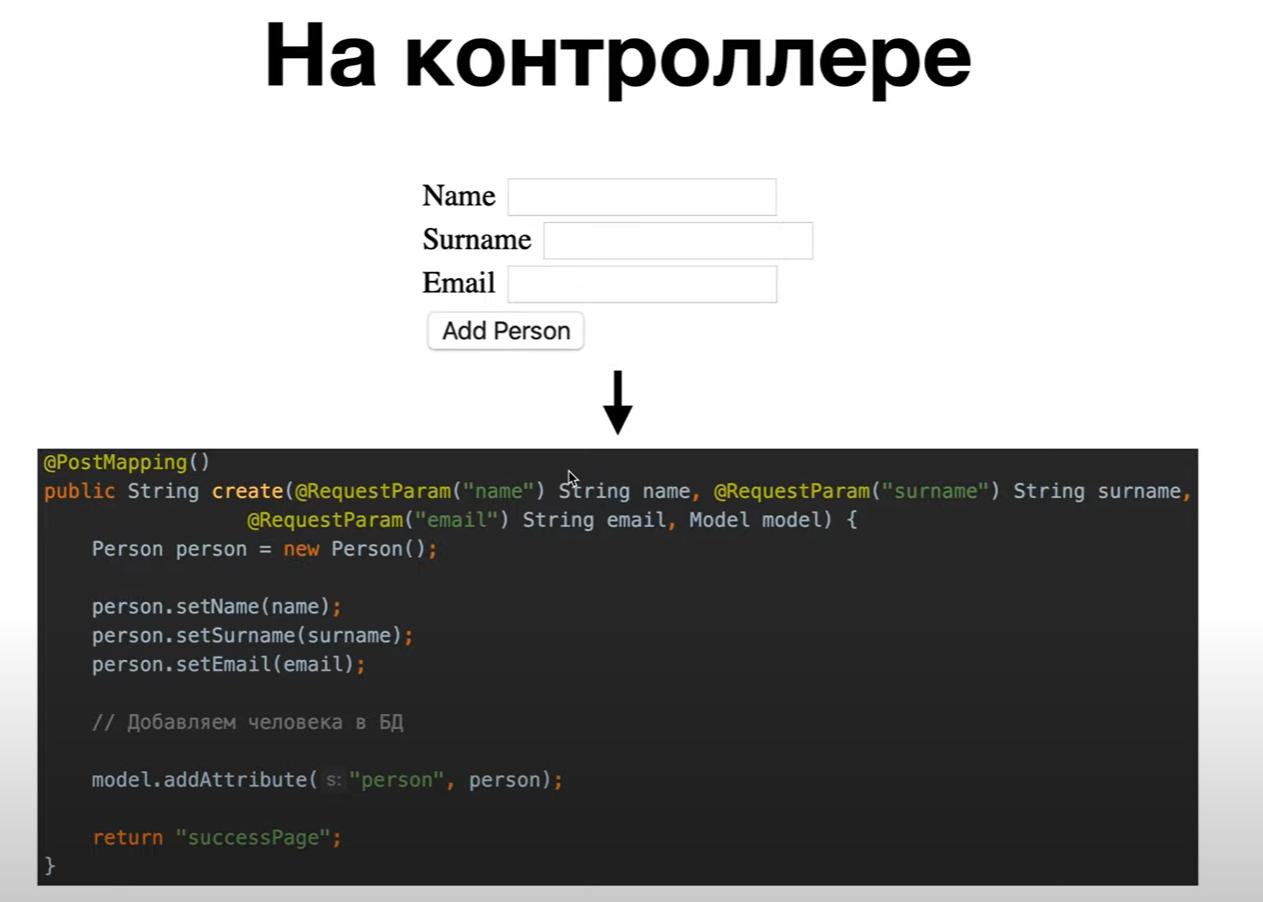
1. Создать контроллер перенаправляющий на форму



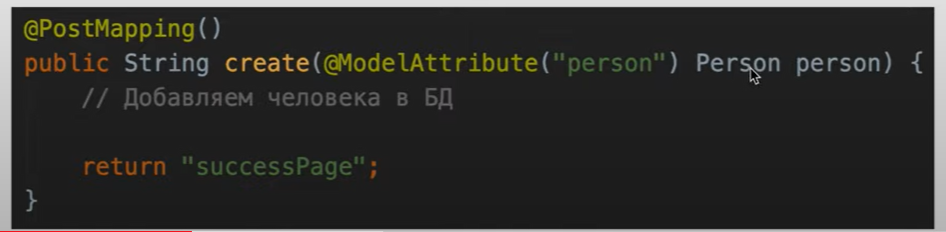
1. Создать представление, которое рисует форму.



1. Создать метод, вызывающийся при отправке формы:



@ModelAttribute – при использовании в аргументе метода, автоматизирует внедрение полученных данных в модель. Например, перепишем контроллер формы person выше:



Если не передать поля класса Person через форму, эти поля будут заполнены полями по умолчанию.

@ModelAttribute – при использовании на методе, внедряет во все создаваемые объекты модели данного контроллера пару ключ-значение. Например, внедрим “headerMessage”=”Welcome”:

@ModelAttribute(“headerMessage”)

public String populateHeaderMessage() {

return “Welcome”; }

Обращение к полю объекта, внедренного в модель, и присваивание ему id для сопоставления с надписью label:

<form th:method=”PATCH” th:action=”@{/people/{id}(id=${person.getID()})” th:object=”$person}”>

<label for=”name”>Enter name: </label>

<input th:”text” th:field=”\*{name}” id=”name”>

</form>

return “redirect:/people” - перенаправление на другую страницу.

HTML 5 поддерживает только методы GET и POST. Для использования других методов, которые поддерживает HTTP, на стороне представления используется скрытое поле:

<form method=”post” action=”/people/1”>

<input type=”hidden” name=”\_method” value=”patch”>

</form>

А на стороне контроллера используются фильтры, подключаемые в конфигурационных файлах.

Для включения валидации форм нужно внедрить зависимость Hibernate Validator и использовать аннотации на полях:

@NotEmpty(message = “Поле не может быть пустым”)

@Size(min = 2, max = 30, message = “Введите от 2 до 30 символов”)

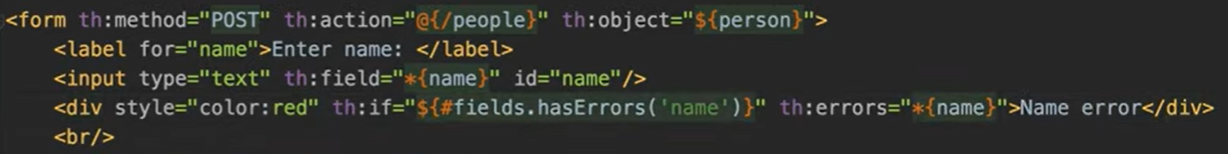
@Min(value = 0, message = “Число должно быть неотрицательным”)

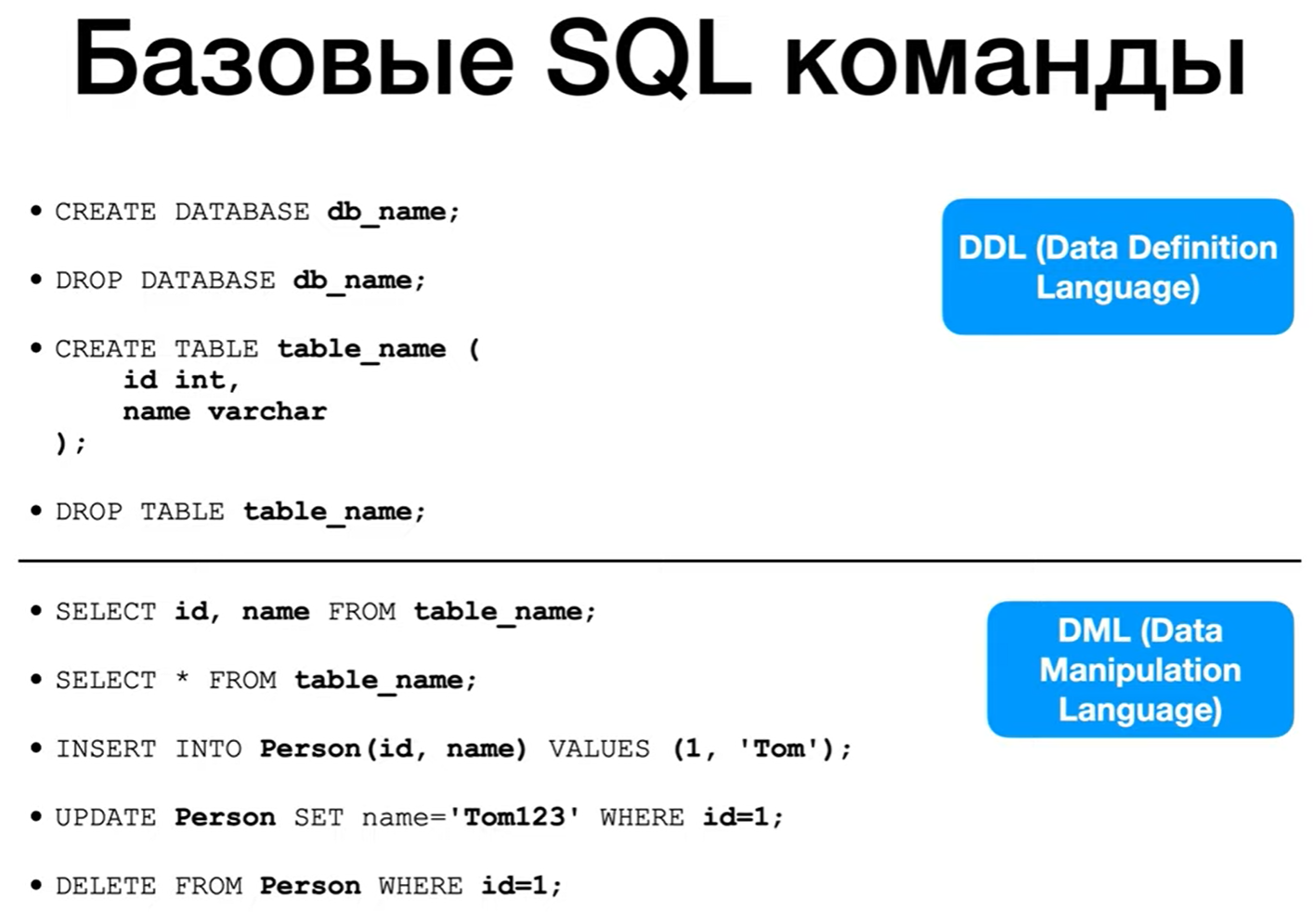
@Email(message = “Не валидная почта“)

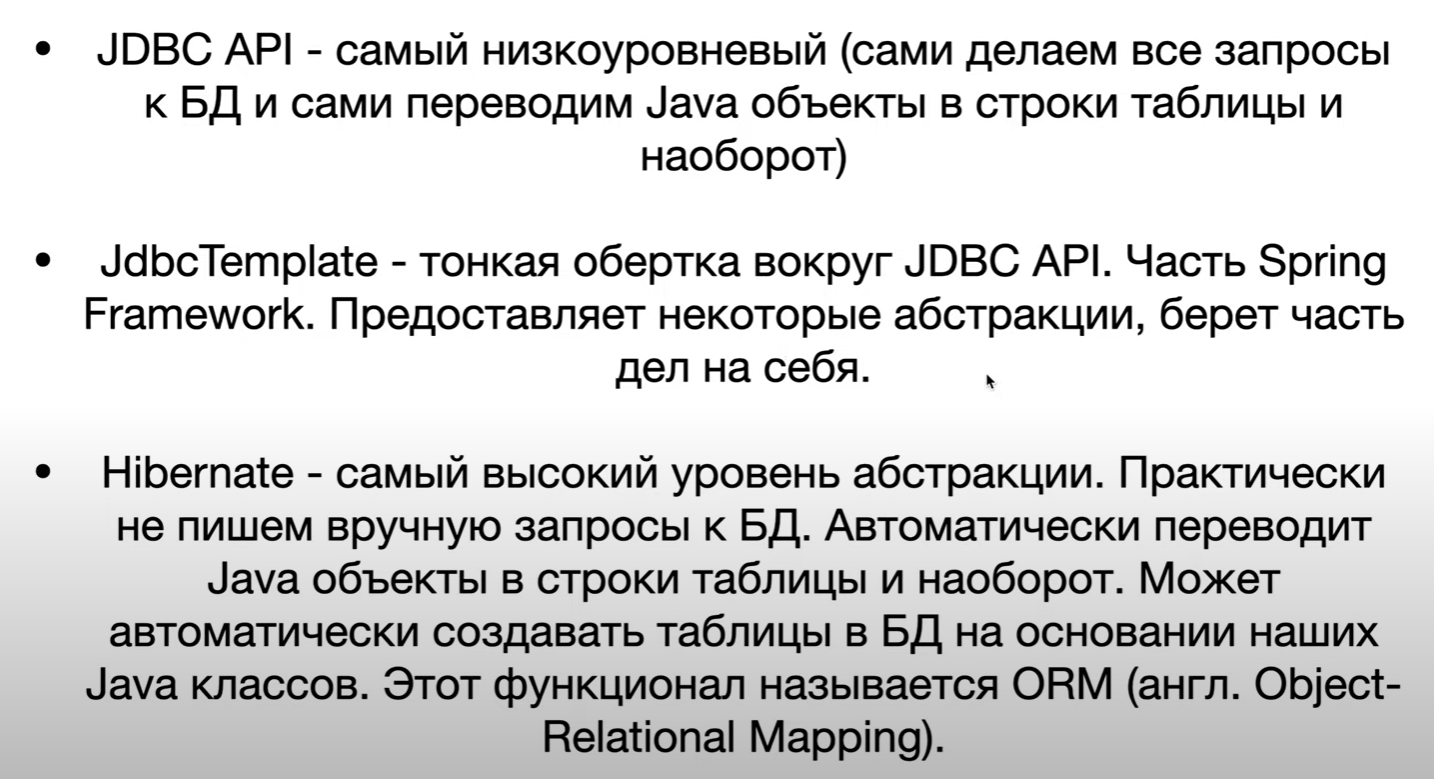
А в методе используем аннотацию @Valid для проверки объекта и его полей, ошибка валидации будет сохраняться в объект BindingResult:



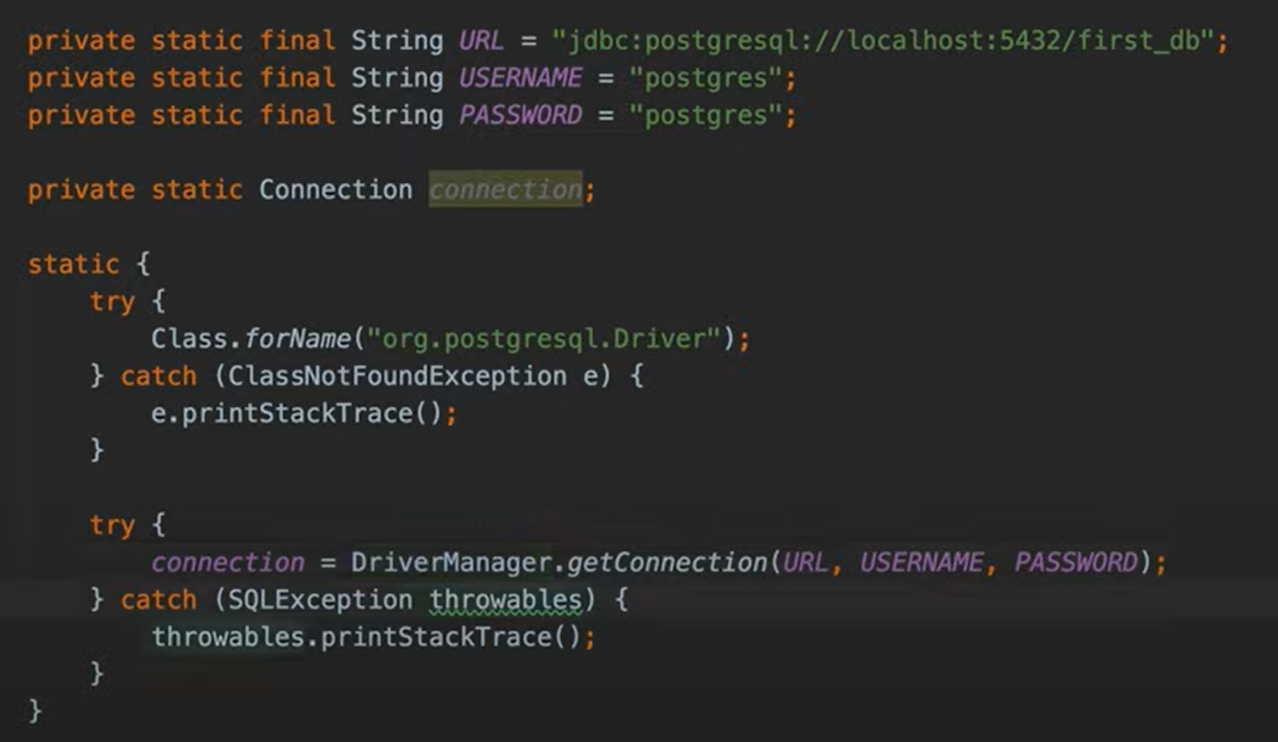
Динамически выведем ошибки валидации для каждого поля:



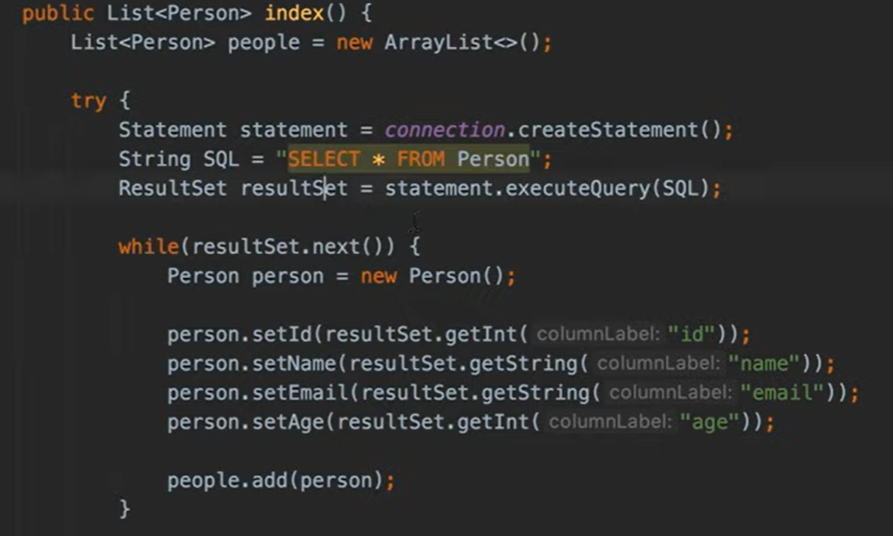




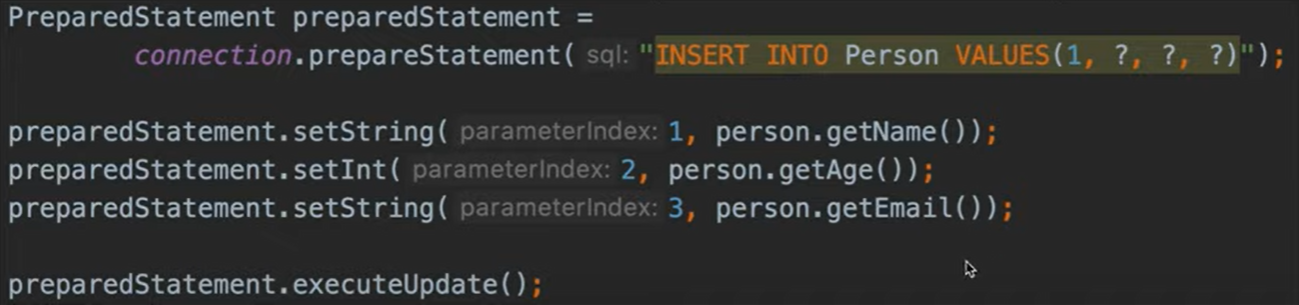
Рассмотрим подключение к БД через **JDBC**. Для подключения к БД нужно подключить зависимость для работы с JDBC и настроить параметры подключения:



И создадим запрос в БД:

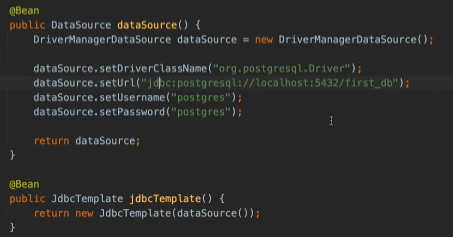


Чтобы защититься от SQL-инъекций, для всех запросов с данными, полученными от пользователя, используется **PreparedStatement**. В этом случае SQL запрос не сможет быть изменен, кроме вставляемых данных в подготовленные места.

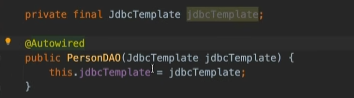


PreparedStatement в отличии от Statement компилируется один раз. В связи с этим он быстрее работает. Дополнительно он кэшируется на БД.

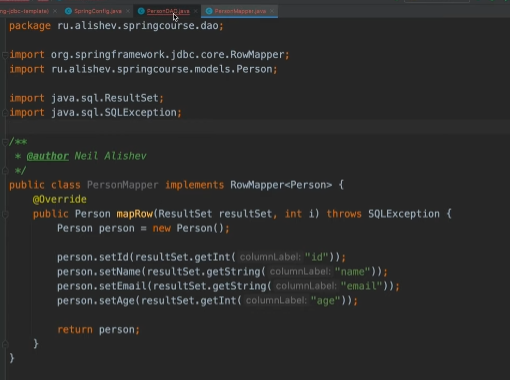
Рассмотрим подключение к БД через **JdbcTemplate**. Для подключения к БД нужно подключить зависимость для работы с JdbcTemplate и создать бины JdbcTemplate и DataSource с настройками подключения:



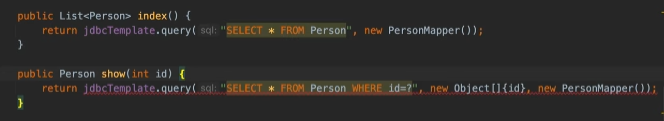
Внедрить бин JdbcTemplate в экземпляр класса, отвечающего за доступ к БД:



Создать класс, реализующий интерфейс RowMapper, для преобразования возвращаемой из БД строки в объект java класса:



И создать запрос к БД:



В JdbcTemplate по умолчанию используется PrepareStatement.

Также можно использовать уже готовый шаблон RowMapper (когда имена полей класса и таблицы совпадают). В этом случае запрос к БД будет таким:

