CPAN-252 WEB APPLICATION DEVELOPMENT FOR CPAN DIPLOMA. PROF. ANTON KOVUNOV

PERSISTING DATA WITH SPRING DATA JPA

RECAP

- We used Spring Data JDBC for providing CrudRepository and using methods provided by repo to perform queries towards Fighter Entity
- We implemented CommandLineRunner to prepopulate some of the fighters to have in the database on application startup
- Now we are moving to Spring Data JPA

WHAT IS JPA

- The JPA specification defines the object-relational mapping internally, rather than relying on vendor-specific mapping implementations. JPA is based on the Java programming model that applies to Java Enterprise Edition (Java EE) environments, but JPA can function within a Java SE environment for testing application functions.
- JPA represents a simplification of the persistence programming model. The JPA specification explicitly defines the object-relational mapping, rather than relying on vendor-specific mapping implementations. JPA standardizes the important task of object-relational mapping by using annotations or XML to map objects into one or more tables of a database.

SPRING DATA JPA

- Spring Data JPA provides us set of tools to use with relational database.
- First of all, we have to add dependency to pom.xml file

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

- This dependency also brings Hibernate transitively as a JPA Implementation, so we can leverage it during development

SPRING DATA JPA

- First of all we need to annotate our fighter domain with @Entity to show that it's an entity to be used in the database, we use javax.persistence package for it.

```
import java.time.LocalDateTime;
import javax.persistence.Column;
import javax.persistence.Entity;
import org.springframework.data.annotation.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Fighter {
```

SPRING DATA JPA

- Regarding repository, JPA works equally well with Spring CrudRepository so we don’t need to change anything there
- More interesting part is implementing custom query that is not provided by CrudRepository
- Let’s see how we can create this query

CUSTOM QUERY IN JPA

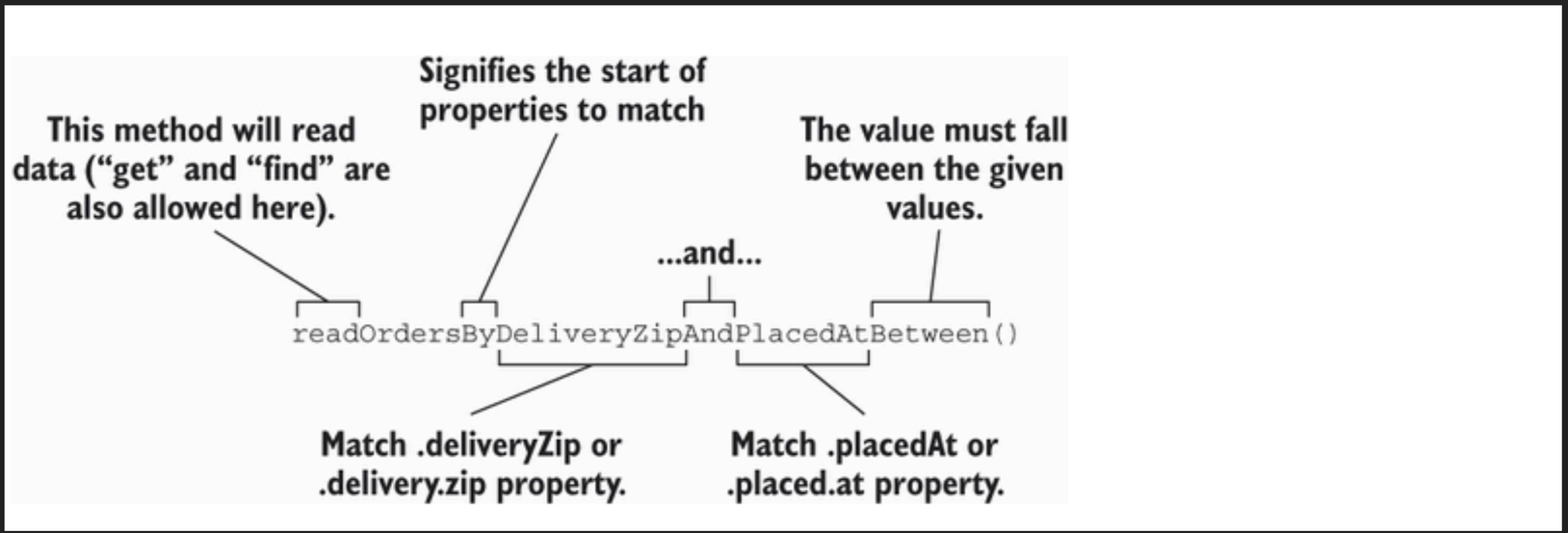
- Let’s create a simple custom query to get fighters by Anime
- We can achieve this by providing query like this in FighterRepository

```
public interface FighterRepository extends CrudRepository<Fighter, Long> {
    List<Fighter> findByAnimeFrom(Anime animeFrom);
}
```

- We need to break down our query semantically: we start it with **find** which shows that we are doing SELECT query, then using **By** we specify that we are going to search using some of the Fighter properties, and after that we provide property name

CUSTOM QUERIES IN JPA

- This query mechanism is pretty powerful, we don’t need to provide query body as JPA understands the semantics from the query name. Lower is more complex example to get orders by zip code and placed between some dates.



CUSTOM QUERIES IN JPA

- There are many ways to provide derived queries in Spring Data, here are some keywords that you can use:
  - IsAfter, IsGreaterThan, IsBefore, IsBetween, IsNull, IsStartingWith, IsEndingWith, ..and so on and so forth
  - Let’s create a custom query to find fighters by name ending with some suffix and createdAt between some dates.
  - I will recommend you try to do it yourself, but I will show the end result

RESULTING QUERY

- To reiterate: we start with **findBy** to indicate that we are searching by some conditions, then we must provide name prefix and two dates: from and to
- This is pretty flexible approach that covers 60% of cases that you may need in your application

```
public interface FighterRepository extends CrudRepository<Fighter, Long> {
    List<Fighter> findByAnimeFrom(Anime animeFrom);

    List<Fighter> findByNameStartsWithAndCreatedAtBetween(String namePrefix, String fromDate, String toDate);
}
```

QUESTIONS?

On the lab we are going to test our custom queries and create even more customized cases with @Query annotation, and we will play with PagingAndSorting repository to implement search in fightersView