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

SETTING UP DATABASE SCHEMA AND USING SPRING DATA JDBC

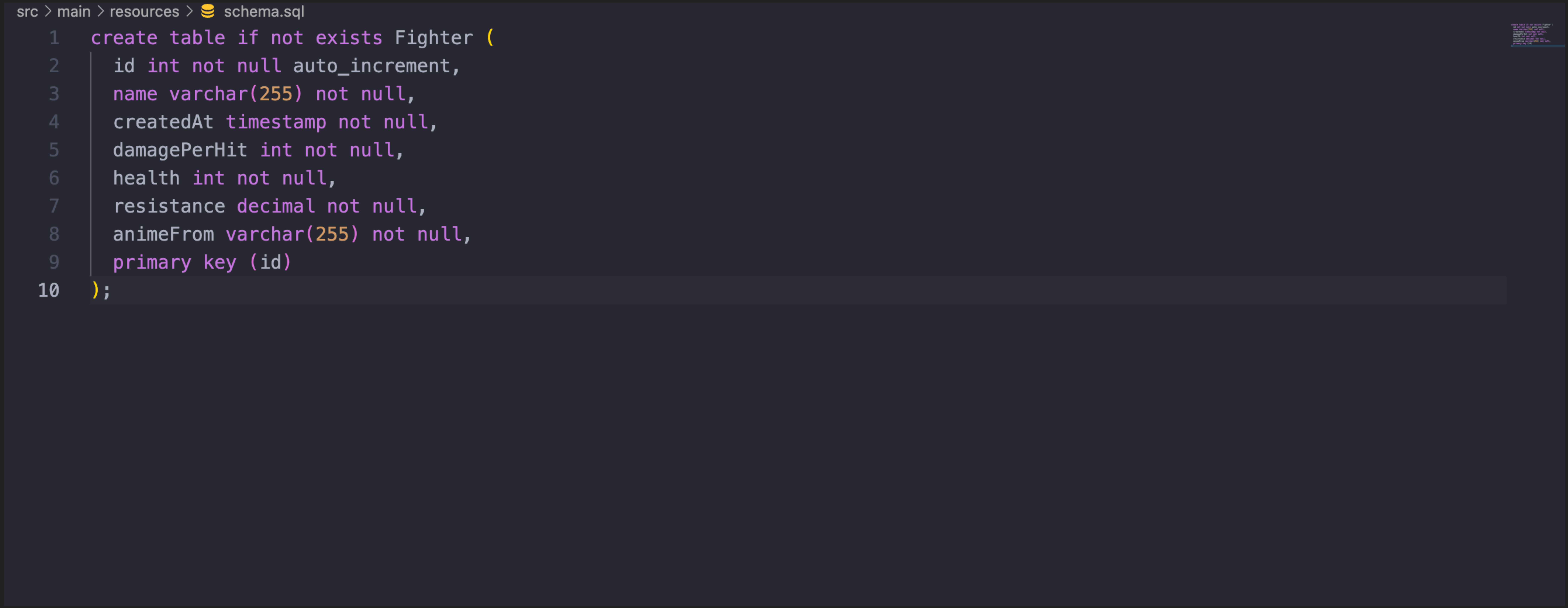
TO REITERATE

- We have implemented repository with methods to find all fighters, get fighter by id and save one on form submission to the database
- Though if you will try to do it now, you will fail with some weird issue: Table Fighter does not exist.
- It seems no good, so let’s do something with it.

SCHEMA FILE TO POPULATE DB

- We implemented database queries and insertion, but we didn’t actually create a table.
- We can do this by providing schema.sql file in src/main/resources folder to create a table
- This will execute sql code on start of the app and give us proper setup to do interactions

SCHEMA SQL FILE



src > main > resources > schema.sql

```sql
create table if not exists Fighter (
  id int not null auto_increment,
  name varchar(255) not null,
  createdAt timestamp not null,
  damagePerHit int not null,
  health int not null,
  resistance decimal not null,
  animeFrom varchar(255) not null,
  primary key (id)
);
```

CHECKING TABLE EXISTENCE IN H2 CONSOLE

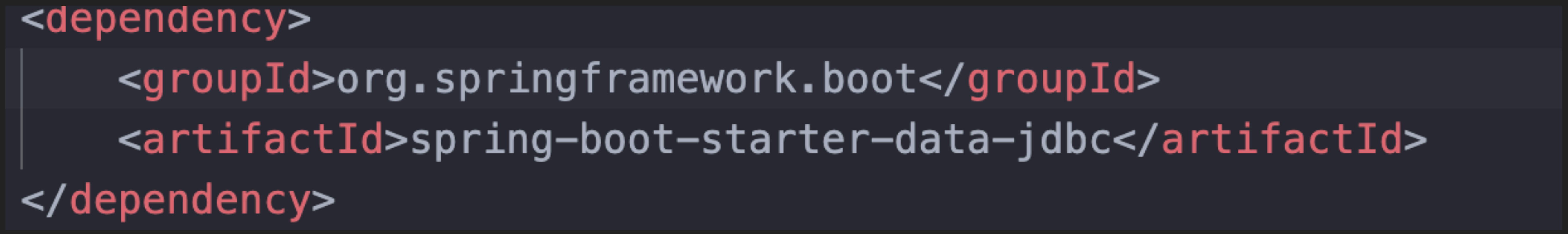
- Now if you go to H2 console after you start up the application, you can see that table was created and you can execute some queries towards it to see that it's actually working
- Though we don't have any entries available for us, let's actually create a fighter through the design page and check the result in the console.

PLEASE TRY TO CREATE A FIGHTER AND CHECK IF IT'S LOCATED IN THE CONSOLE

Anton Kovunov

SPRING DATA JDBC

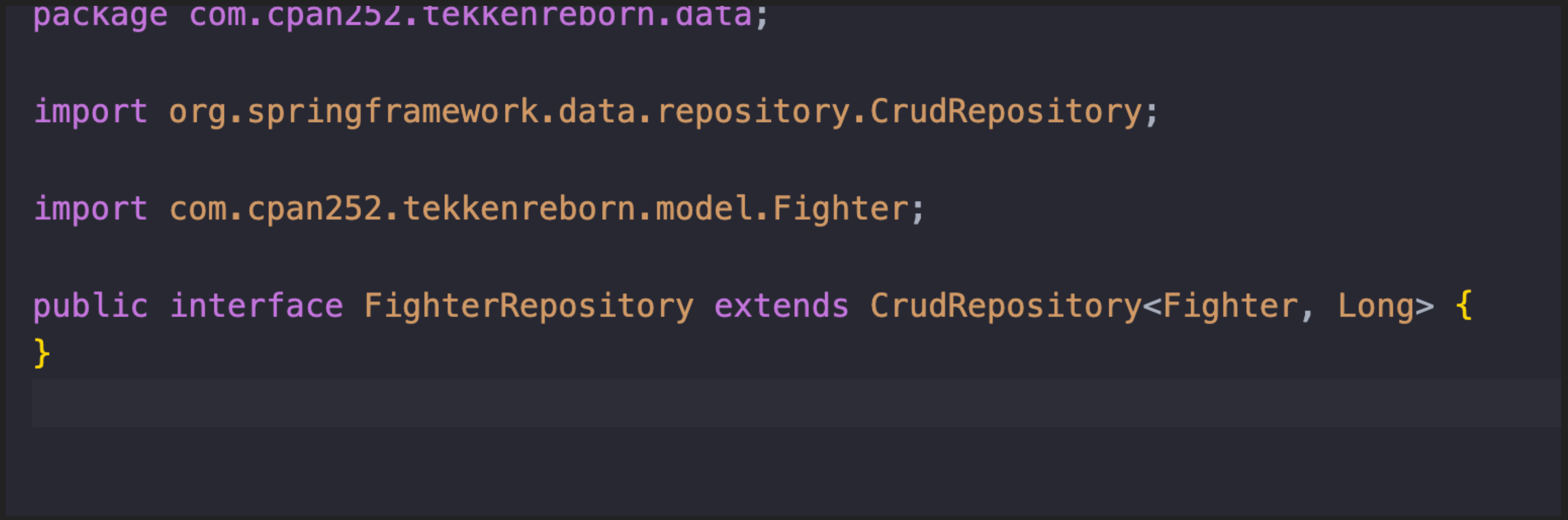
- As of now we used JDBC starter library that provides some improvements on Java SE sql code, but we can do more
- To help us with that we have Spring Data JDBC library
- We can add data keyword to existing one, and we will switch to the new dependency



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

SET UP REPOSITORY INTERFACES

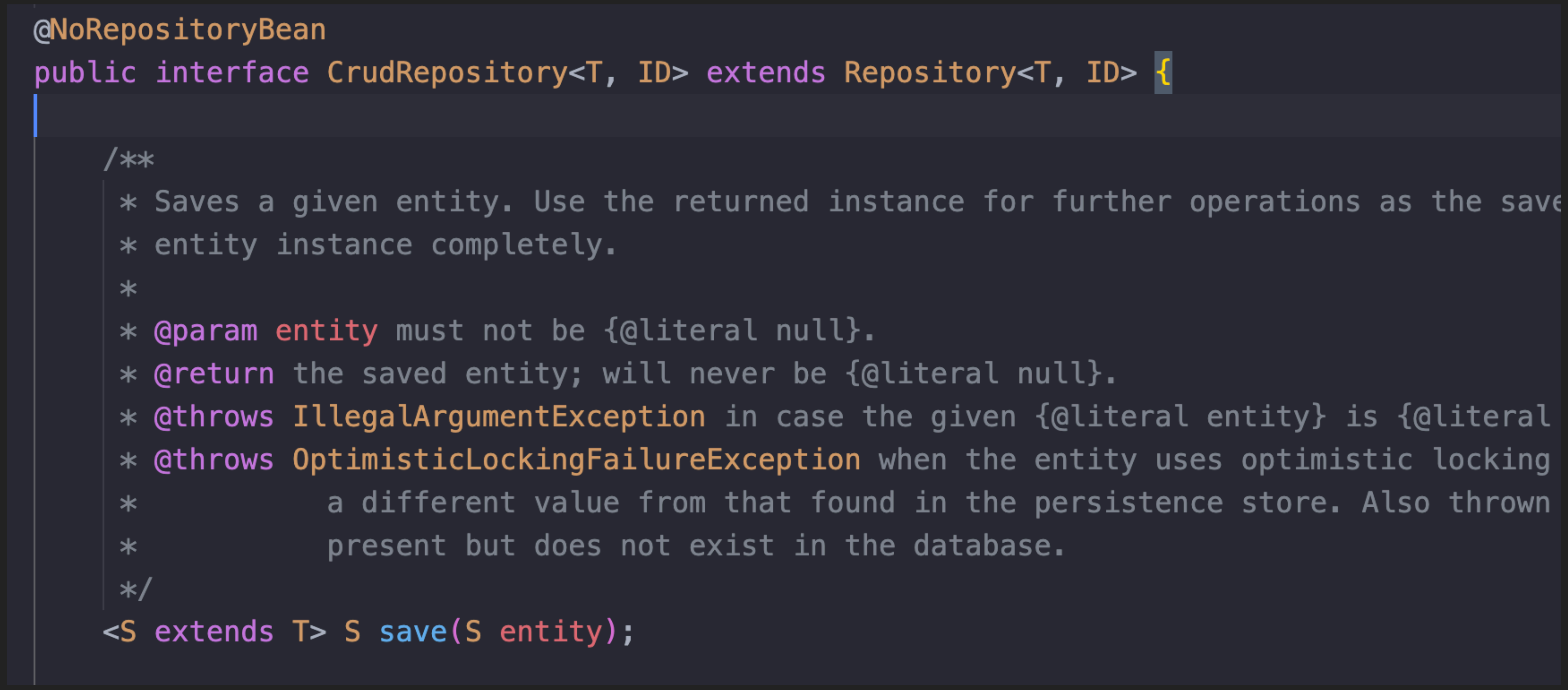
- As we switched to the new dependency we can improve our code a little bit.
- We can use CrudRepository interface to remove some boilerplate code and provide even more functionality



```
package com.cpan252.tekkenreborn.data;

import org.springframework.data.repository.CrudRepository;

import com.cpan252.tekkenreborn.model.Fighter;

public interface FighterRepository extends CrudRepository<Fighter, Long> {
}
```

USING SPRING DATA JDBC

- As you see from the code it became leaner, but where are the methods?
- If we inspect CrudRepository interface, we will see some definitions provided, this is a useful helper from data dependency.



```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {

    /**
     * Saves a given entity. Use the returned instance for further operations as the save
     * entity instance completely.
     *
     * @param entity must not be {@literal null}.
     * @return the saved entity; will never be {@literal null}.
     * @throws IllegalArgumentException in case the given {@literal entity} is {@literal
     * @throws OptimisticLockingFailureException when the entity uses optimistic locking
     *          a different value from that found in the persistence store. Also thrown
     *          present but does not exist in the database.
     */
    <S extends T> S save(S entity);
```

SPRING DATA JDBC

- As of now, we can remove the JDBC Implementation class, as all these methods that we have implemented are generated by CrudRepository as generic CRUD operations, so we are covered on this front
- When we delete JdbcFighterRepository class, you can see that compiler doesn't complain in Controller, where we perform saving of the Fighter, that's the magic of CrudRepository

ANNOTATING DOMAINS WITH PERSISTENCE ANNOTATIONS

- The only thing we have to do now is annotate domain class (Fighter) so Spring Data JDBC knows how to persist it
- Class is too big to paste it here, but we need to provide two annotations:
  - @Table - on the class level to represent that this will be a table
  - @Id - on the Id property, to provide some metadata that this property is an id in the database

TESTING THE APP WORKS

- So let’s test the application correctness and validate that fighter is actually saved to the database and inspect it in H2 console
- Let’s go!

QUESTIONS?

On the lab we are going to learn how to populate data in the table without providing sql files and also develop list view of fighters