

Disciplina Regular 3

# Desenvolvimento Web com Java EE

Graduação em Engenharia de Software - 2020

# Etapa 5 Aula 2

Spring Boot JPA

# Etapa 5 Aula 1 - Competências

- **Competências Trabalhadas Nesta Etapa**
  - **Compreender e utilizar o Spring Boot:**
    - Criar projetos Spring Boot com o Tomcat Embutido.
    - Implementar Controllers do Spring MVC.
    - **Implementar Repositories com JPA.**

# Etapa 5 Aula 1 - Checklist

- **Checklist DR3:**

- Ter revisado os slides e **os exercícios**, principalmente **JPA**.
- Ter implementado exemplos de CRUD com os relacionamentos 1-N e N-M.
  - Construção de telas com JSP + JSTL e EL.
  - Construção de Controllers com validações do Spring MVC.
  - Construção de Repositories do JPA com inserir, alterar, excluir e as consultas.
- Ter experimentado o **AWS Elastic Beanstalk** para implantar os projetos que desenvolveu - WAR e JAR.

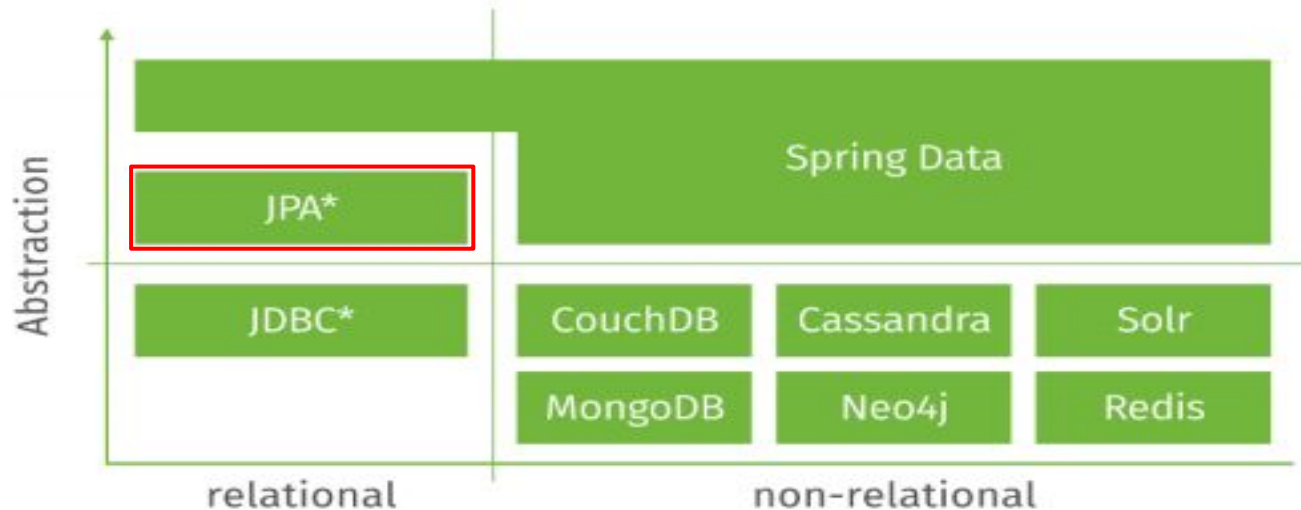
# Spring Data JPA

# Spring Data

- A missão da Spring Data é fornecer um modelo de programação familiar e consistente baseado em Spring para acesso a dados, mantendo as características especiais do armazenamento de dados subjacente.
- Isso facilita o uso de tecnologias de acesso a dados, bancos de dados relacionais e não relacionais, map reduce frameworks e serviços de dados baseados em nuvem.
- Este é um projeto abrangente que contém muitos subprojetos específicos para um determinado banco de dados. Os projetos são desenvolvidos trabalhando em conjunto com muitas das empresas e desenvolvedores que estão por trás dessas tecnologias.

# Spring Data

## Spring Data



# Spring Data

## Spring Data Modules

Core  
modules

Core

JPA

MongoDB

Neo4j

Solr

Gemfire

REST

Redis

KeyValue

Community  
modules

Couchbase

Elasticsearch

Cassandra



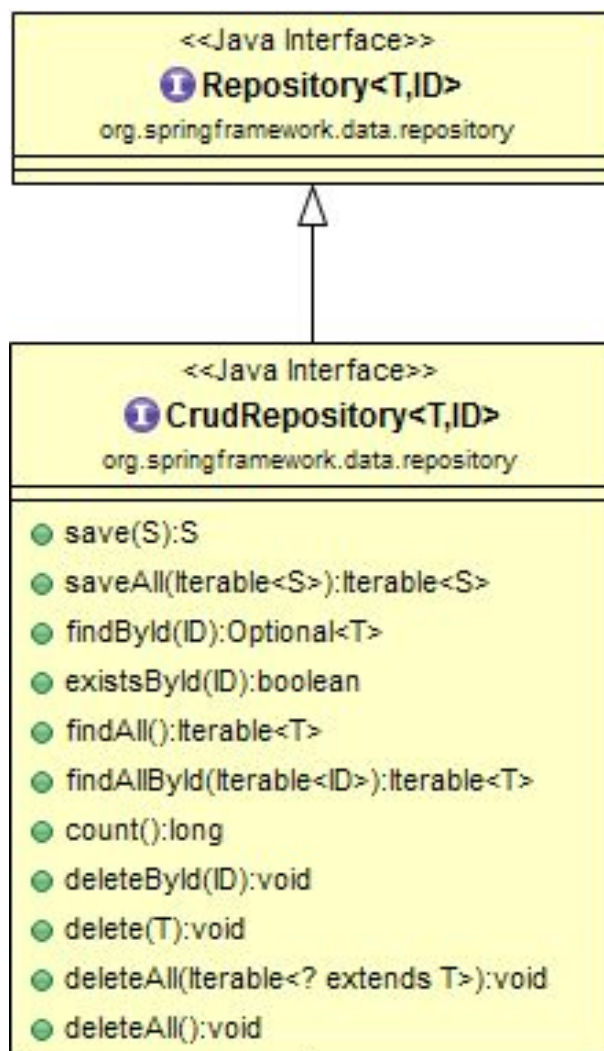
# Spring Data JPA

- **A implementação de uma camada de acesso a dados de um aplicativo é complicada**, com muito **código repetitivo** que precisa ser feito para executar consultas simples, além de realizar paginação e auditoria.
- O Spring Data JPA visa melhorar **significativamente** a implementação das camadas de acesso a dados, reduzindo o esforço ao mínimo necessário.
- O desenvolvedor escreve as interfaces de repositório, incluindo métodos específicos de pesquisa **e o Spring fornecerá a implementação automaticamente.**

# CRUD com Spring Data JPA

# CrudRepository

- A interface central do Spring Data é o Repository.
- Repository atua principalmente como uma interface de marcação para capturar os **tipos com os quais trabalhar** e para ajudar a descobrir interfaces que estendem essa – raiz para outras interfaces.
- CrudRepository é subinterface de Repository e fornece **funcionalidade sofisticada** de CRUD para a classe de entidade que está sendo gerenciada.



# CrudRepository

- O CrudRepository fornece métodos para operação **CRUD genérica**, mas se for necessário adicionar métodos personalizados em nossa interface para ampliar o CrudRepository, temos as seguintes opções:
  - Podemos iniciar os nomes de métodos de consulta com findBy..., readBy..., queryBy..., countBy... e getBy..... Antes do By podemos adicionar expressões como Distinct. Após o By precisamos adicionar os nomes das propriedades da entidade.
  - Para obter dados com base em mais de uma propriedade, podemos concatenar nomes de propriedades usando And e Or ao criar nomes de métodos.

# Exemplo

```
public interface UserRepository extends Repository<User, Long> {  
  
    List<User> findByEmailAddressAndLastname(String emailAddress,  
        String lastname);  
  
}
```

Referência:

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>

# Exemplo

```
public interface ArticleRepository
    extends CrudRepository<Article, Long> {

    List<Article> findByTitle(String title);

    List<Article> findDistinctByCategory(String category);


    List<Article> findByTitleAndCategory(String title, String
category);

    @Query("SELECT a FROM Article a WHERE a.title=:title and
a.category=:category")
    List<Article> fetchArticles(@Param("title") String title,
@Param("category") String category);

}
```

# Exemplo

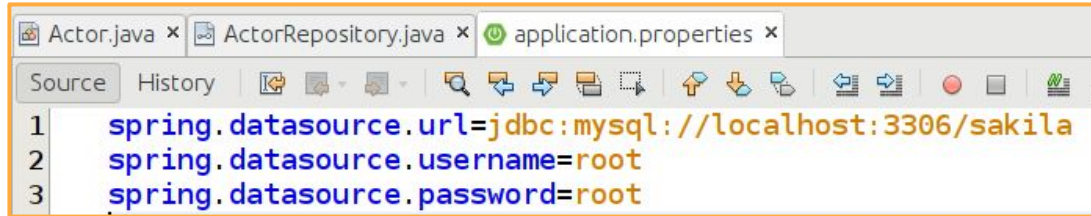
```
Actor.java x
Source History
1 package br.edu.infnet.actors.domain;
2
3 import ...15 lines
18
19 @Entity
20 @Table(catalog = "sakila", schema = "")
21 @NamedQueries({
22     @NamedQuery(name = "Actor.findAll", query = "SELECT a FROM Actor a"),
23     @NamedQuery(name = "Actor.findByActorId", query = "SELECT a FROM Actor a WHERE a.actor_id = :actor_id"),
24     @NamedQuery(name = "Actor.findByFirstName", query = "SELECT a FROM Actor a WHERE a.first_name = :first_name"),
25     @NamedQuery(name = "Actor.findByLastName", query = "SELECT a FROM Actor a WHERE a.last_name = :last_name"),
26     @NamedQuery(name = "Actor.findByLastUpdate", query = "SELECT a FROM Actor a WHERE a.last_update = :last_update")
27 public class Actor implements Serializable {
28
29     private static final long serialVersionUID = 1L;
30     @Id
31     @GeneratedValue(strategy = GenerationType.IDENTITY)
32     @Basic(optional = false)
33     @Column(name = "actor_id", nullable = false)
34     private Short actorId;
```



```
1 package br.edu.infnet.actors.infrastructure;
2
3 import ...2 lines
4
5
6 public interface ActorRepository extends CrudRepository<Actor, Short> {
7 }
```

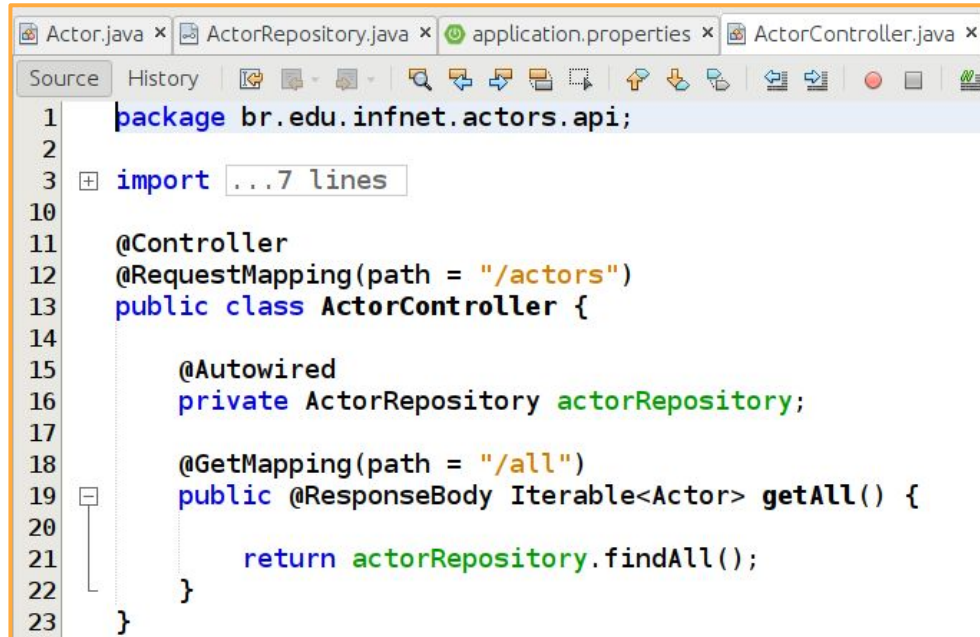


# Exemplo



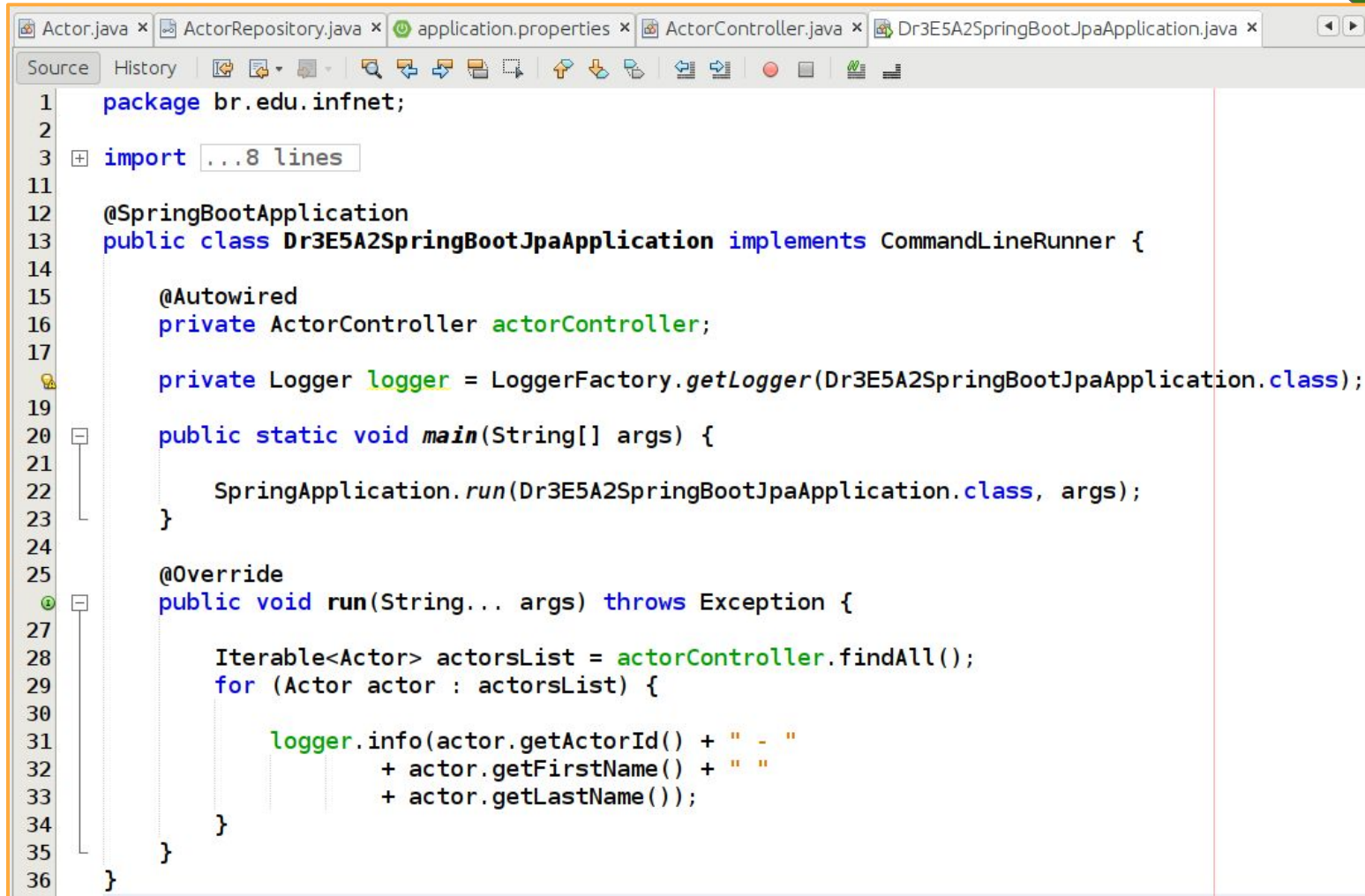
A screenshot of an IDE window titled 'application.properties'. The window shows three lines of configuration for a Spring datasource. The first line sets the URL to 'jdbc:mysql://localhost:3306/sakila'. The second line sets the username to 'root'. The third line sets the password to 'root'.

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/sakila
2 spring.datasource.username=root
3 spring.datasource.password=root
```



A screenshot of an IDE window titled 'ActorController.java'. The window shows the package declaration, imports, and the implementation of the ActorController class. The class is annotated with @Controller and @RequestMapping(path = "/actors"). It has a private field actorRepository of type ActorRepository, annotated with @Autowired. It also has a method getAll() annotated with @GetMapping(path = "/all") and @ResponseBody, which returns actorRepository.findAll().

```
1 package br.edu.infnet.actors.api;
2
3 import ...7 lines
4
5
6
7
8
9
10
11 @Controller
12 @RequestMapping(path = "/actors")
13 public class ActorController {
14
15     @Autowired
16     private ActorRepository actorRepository;
17
18     @GetMapping(path = "/all")
19     public @ResponseBody Iterable<Actor> getAll() {
20
21         return actorRepository.findAll();
22     }
23 }
```



```
Actor.java x ActorRepository.java x application.properties x ActorController.java x Dr3E5A2SpringBootJpaApplication.java x
Source History
1 package br.edu.infnet;
2
3 import ...8 lines
11
12 @SpringBootApplication
13 public class Dr3E5A2SpringBootJpaApplication implements CommandLineRunner {
14
15     @Autowired
16     private ActorController actorController;
17
18     private Logger logger = LoggerFactory.getLogger(Dr3E5A2SpringBootJpaApplication.class);
19
20     public static void main(String[] args) {
21
22         SpringApplication.run(Dr3E5A2SpringBootJpaApplication.class, args);
23     }
24
25     @Override
26     public void run(String... args) throws Exception {
27
28         Iterable<Actor> actorsList = actorController.findAll();
29         for (Actor actor : actorsList) {
30
31             logger.info(actor.getActorId() + " - "
32                         + actor.getFirstName() + " "
33                         + actor.getLastName());
34         }
35     }
36 }
```

# Exercício

- Crie um projeto para implementar o CRUD de contatos, aproveitando a tabela e os dados do exercício de JDBC.