
## Desenvolvimento para Servidores-II

# Spring Data JPA

Neste tópico abortaremos o projeto Spring Data JPA para persistência de objetos em um banco de dados relacional

*Prof. Ciro Cirne Trindade*

# Spring Data

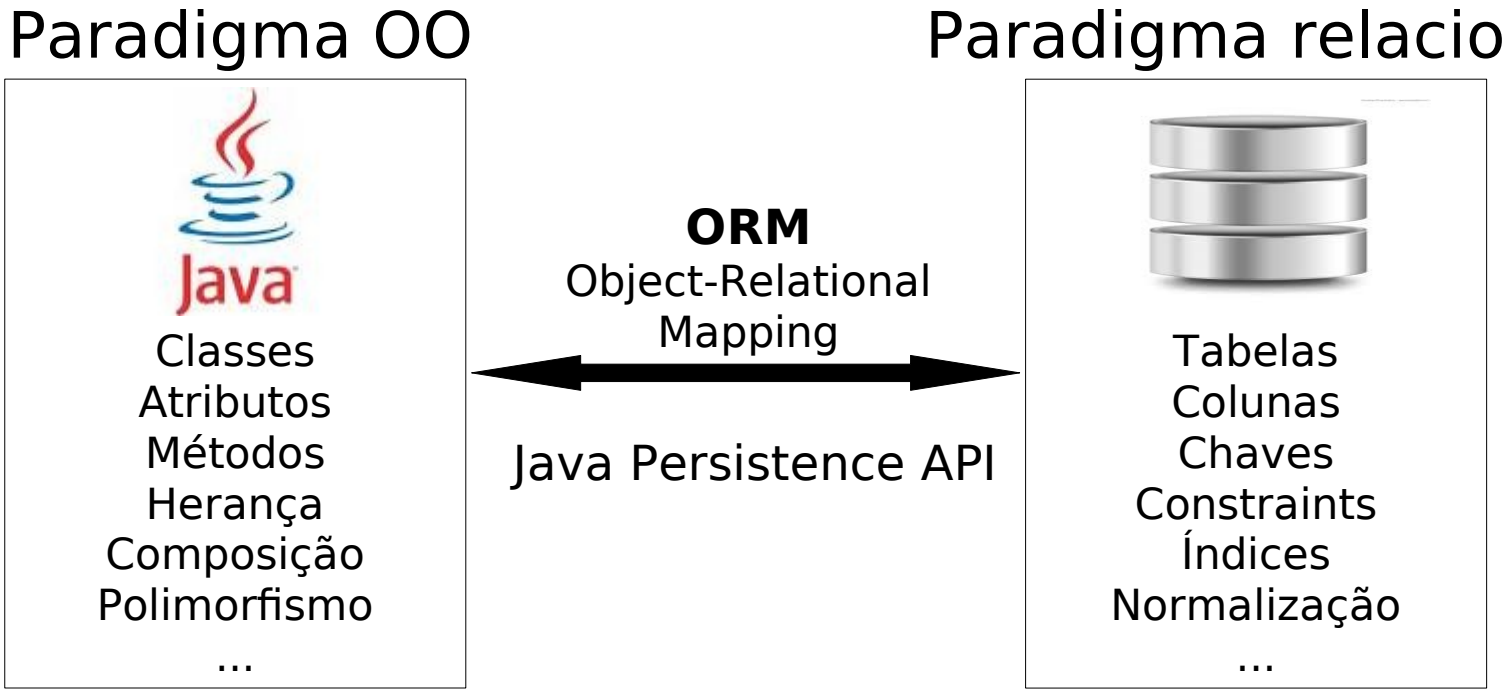
- Spring Data tem como objetivo facilitar o acesso e persistência de dados em bancos de dados relacionais e não-relacionais, serviços de armazenamento baseados em nuvem, entre outros
- É um projeto guarda-chuva que contém vários subprojetos que são específicos para cada banco de dados
  - Spring Data JDBC
  - **Spring Data JPA**
  - Spring Data MongoDB
  - Entre outros

# Spring Data JPA

- Torna fácil a implementação de repositórios baseados em **JPA**
- Ele faz isso nos liberando de ter que implementar as interfaces referentes aos nossos repositórios (ou DAOs)
- Deixando pré-implementadas algumas funcionalidades como, por exemplo, de ordenação das consultas e de paginação de registros

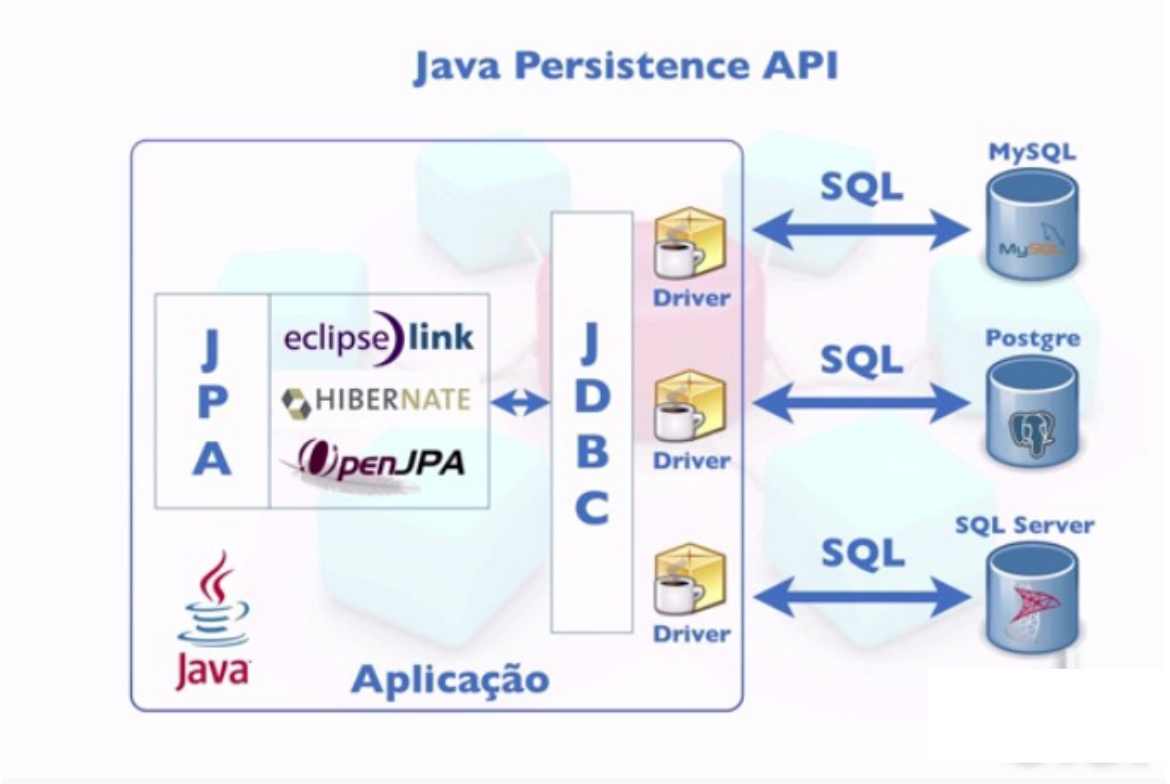
# JPA (1/3)

- JPA provê aos desenvolvedores Java facilidades de um mapeamento objeto/relacional para manipular dados relacionais em aplicações Java



# JPA (2/3)

- Existem diferentes implementação de JPA, todas utilizam JDBC



# JPA (3/3)

- Usando JPA a aplicação nunca manipula o banco de dados diretamente
- JPA usa anotações para marcar as classes que devem ser armazenadas no banco de dados
- Essas classes são chamadas de **entidades**

# Entidades (1/3)

- Tipicamente uma entidade representa uma tabela num banco de dados relacional
- Cada instância de uma entidade representa um registro nesta tabela

# Entidades (2/3)

- Requisitos para que uma classe seja uma entidade:
  - A classe deve ser anotada com `@Entity`
  - Cada classe deve possuir um identificador único, marcado com a anotação `@Id`
  - A classe deve possuir um construtor padrão (sem argumentos)
  - Deve implementar a interface `Serializable`
  - Os atributos da classe não devem ser públicos e só podem ser acessados por métodos da classe

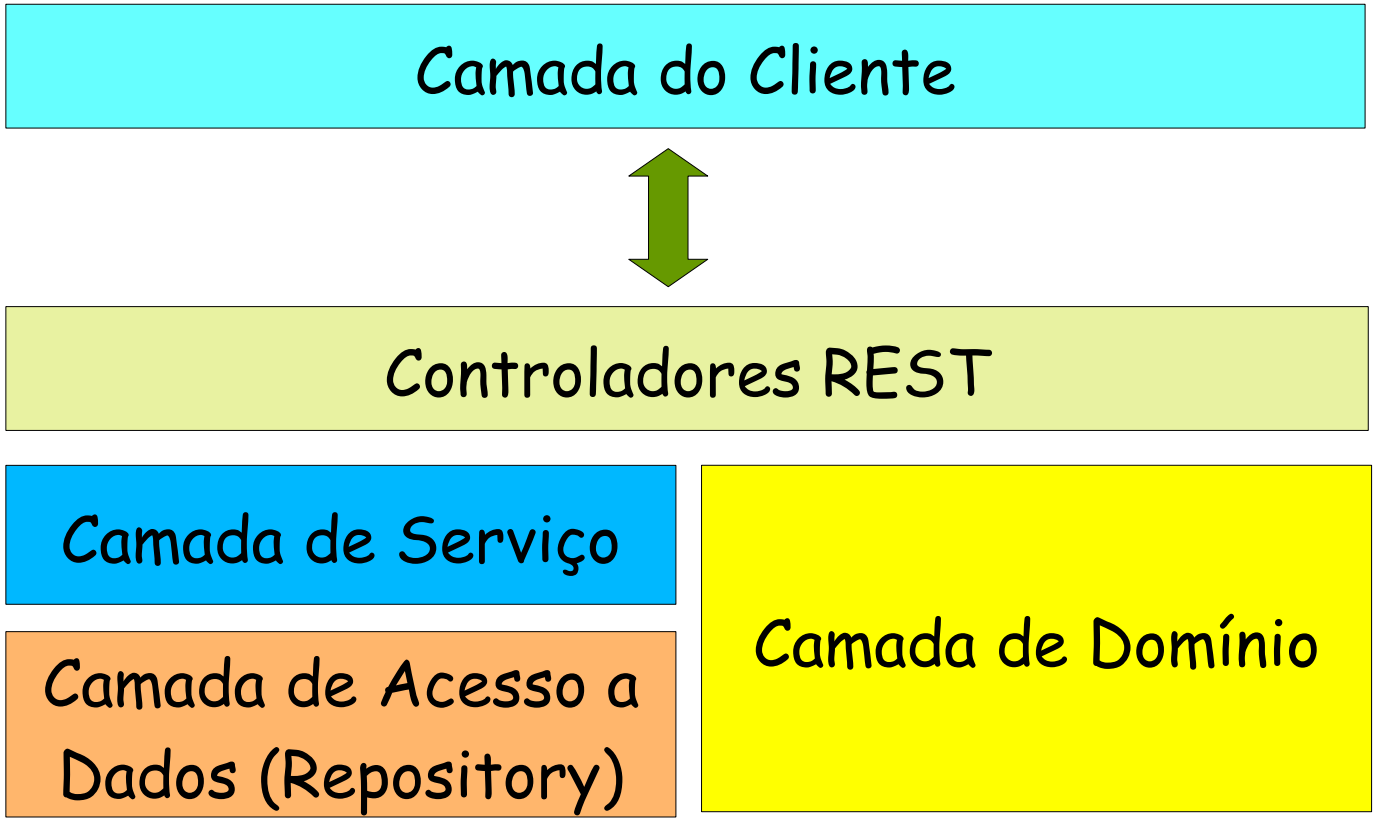
# Entidades (3/3)

- A anotação `@GeneratedValue` é opcional, mas é muito comum usá-la para indicar como a chave primária será gerada
- Com ela indicamos que o banco deve atribuir o valor da chave, e não a aplicação
- Ao inserir um registro no banco de dados, automaticamente será alocada um `ID`
- Como usaremos MySQL, deixaremos a estratégia como `Identity`
  - `@GeneratedValue(strategy= GenerationType.IDENTITY)`

# Exemplo de uma entidade

```
package br.financas.fatec.model;

import java.io.Serializable;
import java.util.Objects;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Conta implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Integer agencia;
    private String numero;
    private String titular;
    private Float saldo;

    public Conta() { }

    // getters e setters
    // hashCode e equals
}
```

# Camadas de uma Aplicação



# Dependências do Spring Data JPA

- Para usar o Spring Data JPA e persistir os dados em um banco de dado MySQL, acrescente as seguintes dependências no arquivo pom.xml:

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

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Spring Data JPA

Driver JDBC do MySQL

# Repositórios

- A principal interface na abstração de repositórios no Spring Data é `Repository`
- Ela espera a classe de domínio (entidade) e o tipo do ID da classe de domínio como argumentos
- A interface `CrudRepository` estende `Repositoy` e provê funcionalidades de CRUD para a entidade que está sendo gerenciada

# CrudRepository

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

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … outras funcionalidades omitidas
}
```

Persiste `entity`

Devolve a entidade cujo ID é igual a `primaryKey`

Devolve todas as entidades

Devolve o número de entidades

Exclui a `entity`

Indica quando uma entidade com ID igual a `primaryKey` existe

# JpaRepository

- Também há abstrações de persistência para tecnologias específicas, tais como `JpaRepository` ou `MongoRepository`
- Estas interfaces estendem `CrudRepository` e expõem as capacidades de uma tecnologia de persistência específica

# Repositórios

- Um repositório Spring Data é uma interface anotada com `@Repository` e que estende alguma interface derivada de `CrudRepository`
- No nosso caso, vamos estender a interface `JpaRepository`

# ContaRepository.java

```
package br.fatec.financas.repository;

import org.springframework.data.jpa.
                                repository.JpaRepository;
import org.springframework.stereotype.Repository;
import br.financas.fatec.model.Conta;

@Repository
public interface ContaRepository extends
                        JpaRepository<Conta, Long> {

}
```

Entidade para que destina o repositório

Tipo do atributo marcado com `@Id` na entidade

# Configurando o banco de dados (1/2)

- Vamos criar um perfil de desenvolvimento
  - application-dev.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/
financas_fatec?createDatabaseIfNotExist=true
&useSSL=false
spring.datasource.username=root
spring.datasource.password=root


spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

# Configurando o banco de dados (2/2)

- Definir o perfil atual
  - application.properties

```
spring.profiles.active=dev
```

# ServiceInterface.java

```
package br.fatec.fincanca.service;

import java.util.List;

public interface ServiceInterface<T> {
    T create(T obj);
    T findById(Long id);
    List<T> findAll();
    boolean update(T obj);
    boolean delete(Long id);
}
```

# ContaService.java (1/2)

```
@Service
public class ContaService implements
                        ServiceInterface<Conta> {

    @Autowired
    private ContaRepository repository;

    public ContaService() {}

    @Override
    public Conta create(Conta obj) {
        repository.save(obj);
        return obj;
    }

    @Override
    public Conta findById(Long id) {
        Optional<Conta> obj = repository.findById(id);
        return obj.orElse(null);
    }
```

Anota a classe como sendo da camada de Serviço

Injeta um objeto da interface ContaRepository

# ContaService.java (2/2)

```
    @Override
    public List<Conta> findAll() {
        return repository.findAll();
    }

    @Override
    public boolean update(Conta obj) {
        if (repository.existsById(obj.getId())) {
            repository.save(obj);
            return true;
        }
        return false;
    }

    @Override
    public boolean delete(Long id) {
        if (repository.existsById(id)) {
            repository.deleteById(id);
            return true;
        }
        return false;
    }
}
```

# ControllerInterface.java

```
package br.fatec.financas.controller;

import java.util.List;
import org.springframework.http.ResponseEntity;

public interface ResourceInterface<T> {
   ResponseEntity<List<T>> getAll();
   ResponseEntity<?> get(Long id);
   ResponseEntity<T> post(T obj);
   ResponseEntity<?> put(T obj);
   ResponseEntity<?> delete(Long id);
}
```

# ContaController.java (1/3)

```
@RestController
@RequestMapping("/contas")
public class ContaController implements
                                    ResourceInterface<Conta> {
   @Autowired
   private ContaService service;

   @Override
   @GetMapping
   public ResponseEntity<List<Conta>> getAll() {
      return ResponseEntity.ok(service.findAll());
   }

   @Override
   @GetMapping(value = "/{id}")
   public ResponseEntity<?> get(@PathVariable("id") Long id){
      Conta obj = service.findById(id);
      if (obj != null) {
         return ResponseEntity.ok(obj);
      }
      return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
   }
```

# ContaController.java (2/3)

```
@Override
@PostMapping
public ResponseEntity<Conta> post(@RequestBody Conta obj)
{
   service.create(obj);
   URI location=ServletUriComponentsBuilder.fromCurrentRequest()
         .path("/{id}").buildAndExpand(obj.getId()).toUri();
   return ResponseEntity.created(location).body(conta);
}

@Override
@PutMapping
public ResponseEntity<?> put(@RequestBody Conta obj)
{
   if (service.update(obj)) {
      return ResponseEntity.ok(obj);
   }
   return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
}
```

# ContaController.java (3/3)

```
    @Override
    @DeleteMapping(value = "/{id}")
    public ResponseEntity<?> delete(@PathVariable("id") Long id) {
       if (service.delete(id)) {
          return ResponseEntity.ok().build();
       }
       return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    }

}
```

# Outras Anotações do JPA (1/2)

- O JPA por *default* cria as tabelas a partir dos nomes das classes de entidade e as colunas a partir dos atributos
- Para gerar nomes diferentes de tabelas é possível usar a anotação **`@Table`**
- Por exemplo:
  - `@Table(name="tb_conta")`

# Outras Anotações do JPA (2/2)

- De forma semelhante, para gerar nomes diferentes e definir outros atributos de colunas é possível usar a anotação **`@Column`**
- Por exemplo:
  - `@Column(name="nm_titular", length = 100)`
  - `@Column(name="nr_numero", nullable = false)`

Por *default* atributos do tipo String geram campos colunas do tipo `VARCHAR(255)`

Não aceita nulo (`NULL`) como valor da coluna

# Referências


- Spring Data JPA. Disponível em: https://spring.io/projects/spring-data-jpa
- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf, 2014.