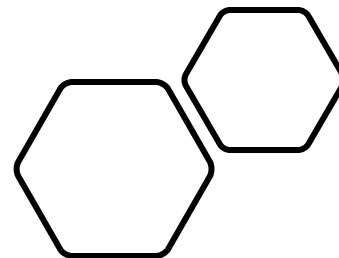


Arquitetura de Software

Prof. Bernardo Copstein



Persistência com JPA




Leituras recomendadas:

- [Java Persistence API \(oracle.com\)](https://javaee.github.io/tutorial/)
- <https://spring.io/guides/gs/accessing-data-jpa/>



JPA – The Java Persistence API



- A API de persistência de Java fornece um modelo de persistência baseado em POJOs que faz mapeamento objeto relacional. A JPA foi desenvolvida para o EJB3.0 como parte da JSR 220, mas seu uso não é limitado a componentes de software EJB. A JPA pode ser usada diretamente em qualquer tipo de aplicação Java incluindo aplicações Java SE. Para mais informações consulte a JSR 220.
- 

As dependências

- Considerando que estamos desenvolvendo um projeto Spring necessário incluir a dependência específica para JPA (ver abaixo) e uma dependência que especifique o SGBD que será usado (inicialmente o H2).

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

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

Anotando as entidades

- Para que o JPA possa entender que uma classe corresponde a uma entidade que deve ser persistida, ela deve ser corretamente anotada como no exemplo ao lado.
- A anotação “@Entity” indica que se trata de uma entidade que deve ser persistida no banco de dados. A anotação “@Id” deve anteceder o campo que será usado como chave primária e a anotação “@GeneratedValue” especifica o tipo de estratégia para a geração dos valores deste campo.
- O construtor “protected” é uma exigência da JPA.
- Para chaves auto geradas o tipo deve ser Long.
- A auto geração da chave primária não é obrigatória.

```
@Entity
public class ItemEstoque {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long nroItem;
    private Long codigoProduto;
    private int quantidade;

    public ItemEstoque(Long codigoProduto, int quantidade) {
        if (quantidade < 0) {
            throw new SistVendasException(
                SistVendasException.Causa.QUANTIDADE_INVALIDA);
        } else {
            this.codigoProduto = codigoProduto;
            this.quantidade = quantidade;
        }
    }

    // Para uso da JPA
    protected ItemEstoque() {}
    // ...
}
```

Gerando o “Repository”

- O Spring é capaz de criar um repositório com as operações de CRUD padrão automaticamente.
- Para tanto precisamos apenas criar uma interface derivada da interface “CrudRepository”
- “CrudRepository” é genérica e permite identificar os tipos da chave e da entidade que deve ser armazenada.
- Os métodos CRUD padrão são gerados automaticamente. É possível especificar métodos adicionais de busca dentro de um certo padrão baseado no nome dos atributos e no padrão da interface “CrudRepository”.
- O Spring se encarrega de gerar uma classe que implementa esta interface e providencia a injeção de dependências.

```
public interface EstoqueH2BD_ITF extends CrudRepository<ItemEstoque, Long> {  
    List<ItemEstoque> findByCodigoProduto(Long codProd);  
    List<ItemEstoque> findAll();  
}
```

Anotando o método “main”

- Para que a JPA funcione corretamente é necessário acrescentar a seguinte anotação na classe que contém o método “main”:

```
@EnableJpaRepositories(basePackages = {"com.bcopstein"})
```



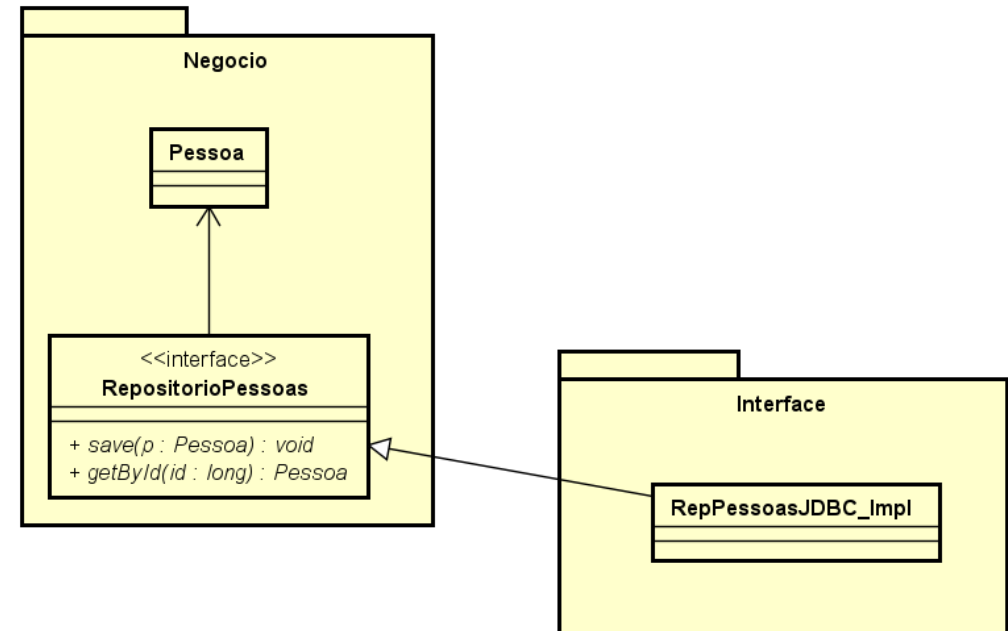
Seu “pacote base”

Cuidado para não abandonar o ISP e o DIP

- Note que se substituirmos a interface com as operações de acesso a dados que se encontra na camada de domínio pela interface derivada de “CrudRepository” estaremos violando 2 princípios:
 - ISP: porque nossas classes irão depender de mais operações do que realmente necessitam
 - DIP: porque nossas classes irão depender diretamente de uma interface proposta pela tecnologia.
 - Além de quebrar o DIP aumentamos a coesão com a tecnologia tornando quase impossível sua substituição

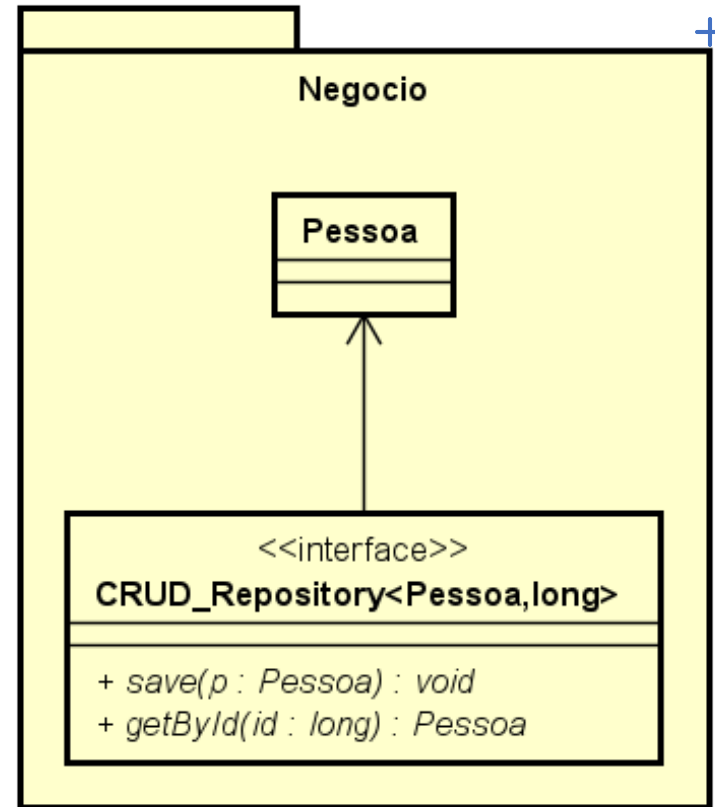
Exemplo: usando JDBC

- Supondo que a camada de negócio define a interface de um repositório.
- A solução correta é que na camada de interfaces adaptadora tenhamos uma classe que implementa esta interface
- Na figura a implementação usa JDBC



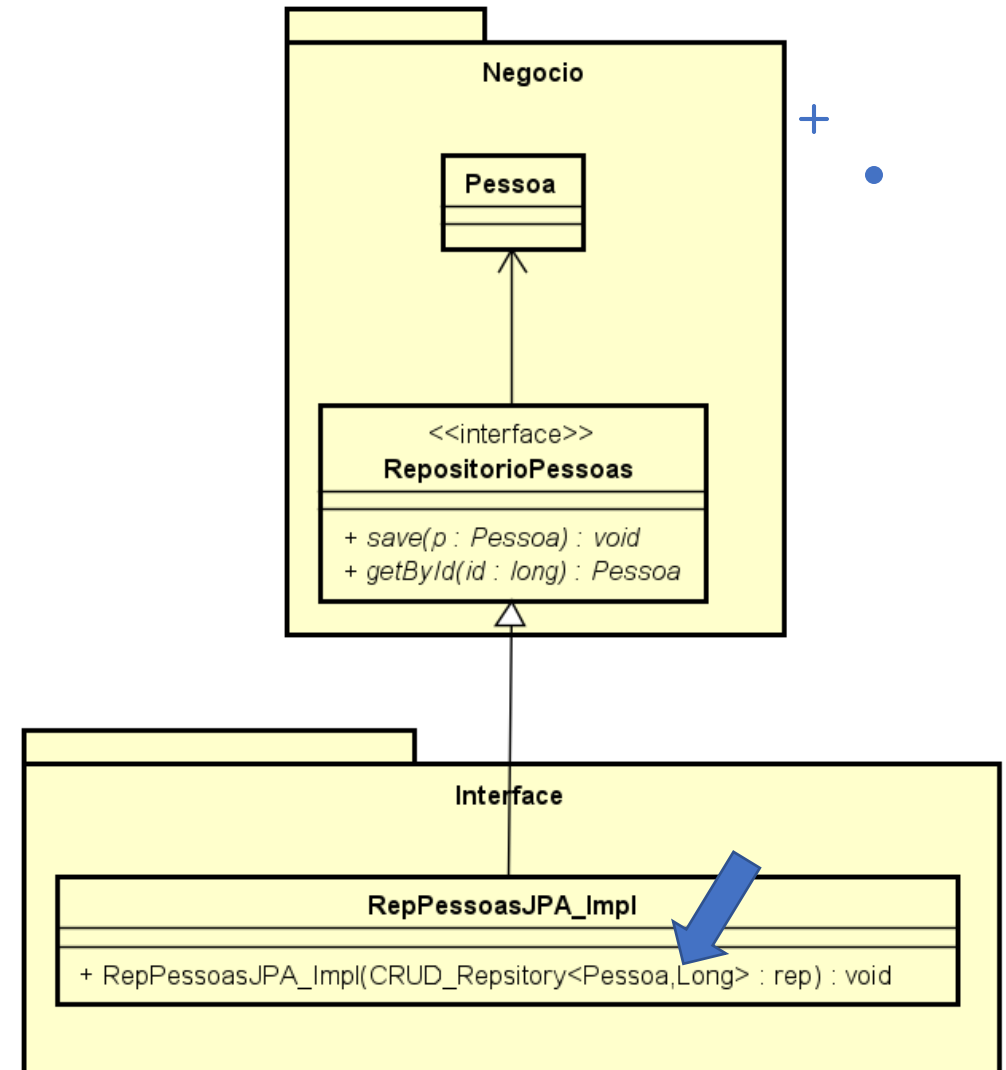
O erro comum

- Quando se usa JPA, um erro comum é utilizar a interface fornecida pela tecnologia diretamente na camada de domínio como na figura ao lado
- Desta forma violamos ISP (porque CRUD_Repository define mais operações que o necessário) e DIP (porque trazemos a tecnologia direto para dentro do domínio)



Solução

- Mantenha as classes que implementam os “Repository” na camada de adaptadores de interface
- Os “Repository” devem receber a interface definida anteriormente por parâmetro do construtor (ver ao lado)
- O Spring irá se encarregar de gerar uma implementação para o CRUD e fazer a injeção na camada de interface.



Configurações (1)

- É padrão que a pasta “resources” seja utilizada para arquivos e scripts de configuração
- No caso iremos utilizar este recurso para inicializar o banco de dados, visto que estamos usando um BD em memória
- Um arquivo chamado “application.properties” pode ser incluído na pasta “resources”. Este pode conter uma série de parâmetros de configuração. Como exemplo use os parâmetros ao lado.
- Estes parâmetros indicam que na subpasta “resources/db” existe um arquivo chamado “data.sql” que contém um script SQL que deve ser executado quando o servidor começar a executar.

```
spring.sql.init.data-locations=classpath*:db/data.sql
#spring.datasource.data=classpath*:db/data.sql
spring.jpa.hibernate.ddl-auto=create
logging.level.org.hibernate.SQL=DEBUG
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

Configurações (2)

- Exemplo de script SQL (arquivo data.sql)

```
INSERT INTO produto (codigo,descricao,preco) VALUES (10, 'Geladeira', 2500.0);
INSERT INTO produto (codigo,descricao,preco) VALUES (20, 'Fogao', 1200.0);
INSERT INTO produto (codigo,descricao,preco) VALUES (30, 'Lava louca', 4300.0);
INSERT INTO produto (codigo,descricao,preco) VALUES (40, 'Lava roupa', 3350.0);
INSERT INTO produto (codigo,descricao,preco) VALUES (50, 'Aspirador de po', 780.0);

INSERT INTO item_estoque(nro_item,codigo_produto,quantidade) VALUES (1, 10, 10);
INSERT INTO item_estoque(nro_item,codigo_produto,quantidade) VALUES (2, 20, 0);
INSERT INTO item_estoque(nro_item,codigo_produto,quantidade) VALUES (3, 30, 7);
INSERT INTO item_estoque(nro_item,codigo_produto,quantidade) VALUES (4, 40, 11);
INSERT INTO item_estoque(nro_item,codigo_produto,quantidade) VALUES (5, 50, 22);
```

- Observe a regra de formação dos nomes das tabelas a partir dos atributos das classes
- Note que mesmo para as chaves que são auto geradas é necessário informar o valor da chave primária. O modo automático só funciona quando o objeto é inserido a partir do JPA.

Configurações (3)

- Observe este conjunto de configurações adicionais que permite configurar o acesso ao console do banco de dados.
- Com esta configuração acesse a console em: localhost:8080/h2-console
- Conecte com o seguinte BD: jdbc:h2:mem:testdb
- Use “sa” como usuário e deixe a senha em branco

```
# Enabling H2 Console
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.data.jpa.repositories.bootstrap-mode=default
```

Veja as listas de exercícios

Exercícios de fixação



