

Laboratório

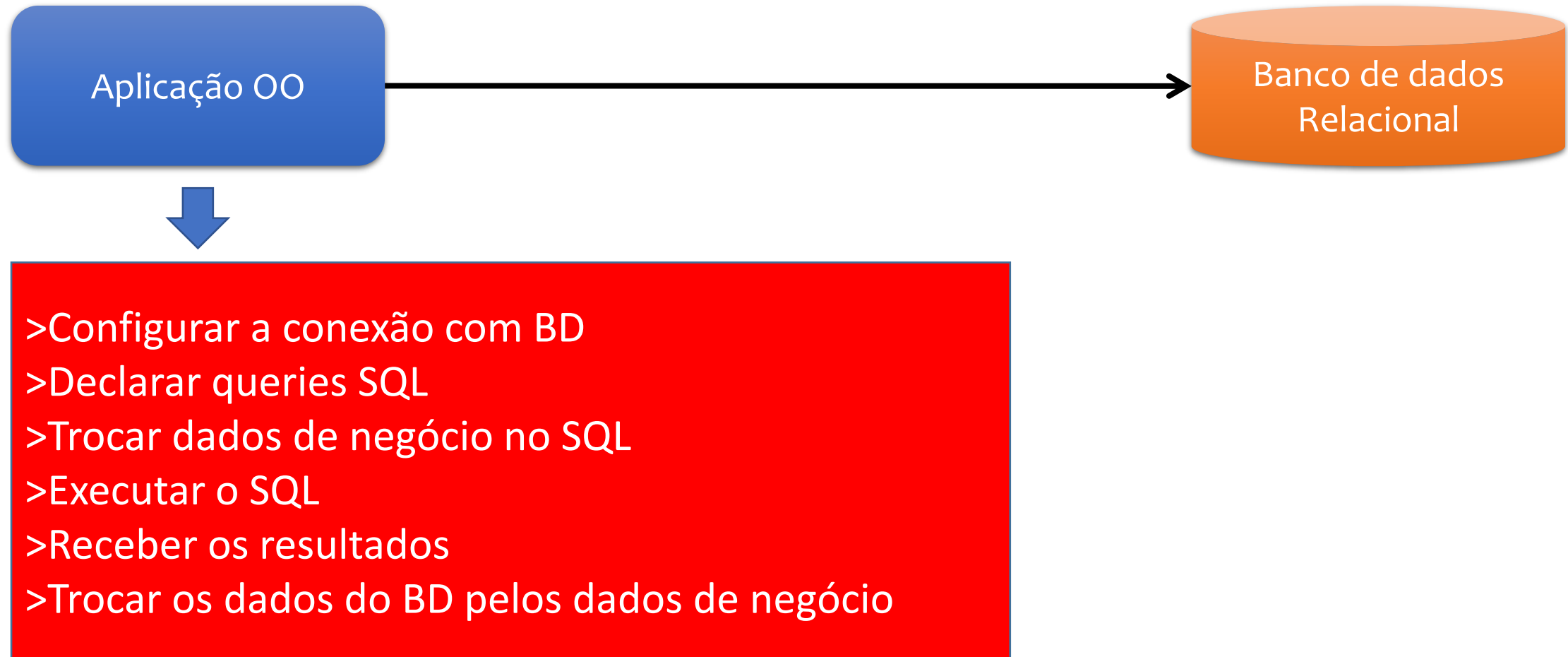
Integração Aplicação OO e BD Relacional

Professor:
Vitor Figueiredo

Disciplina
Sistemas Distribuídos

Versão:
5.0

Introdução



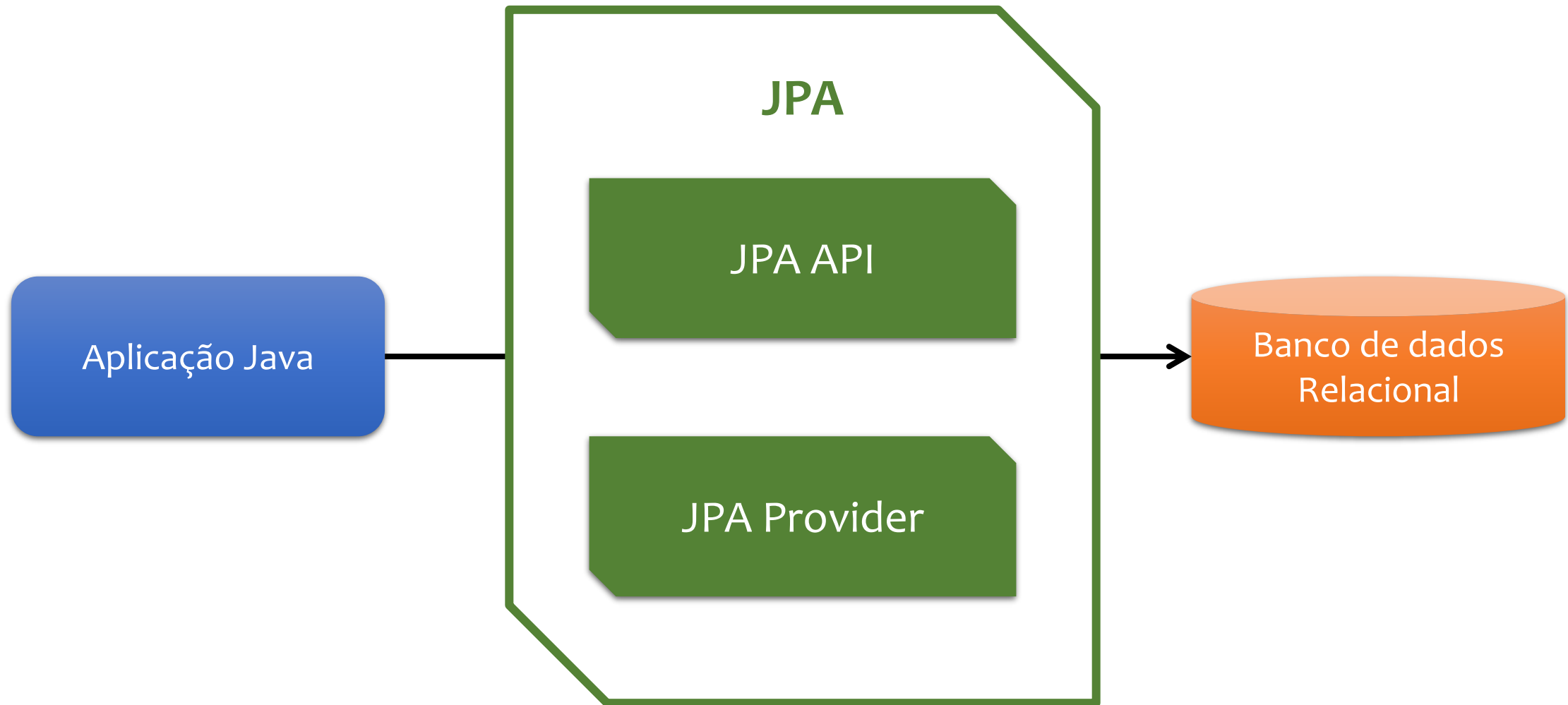


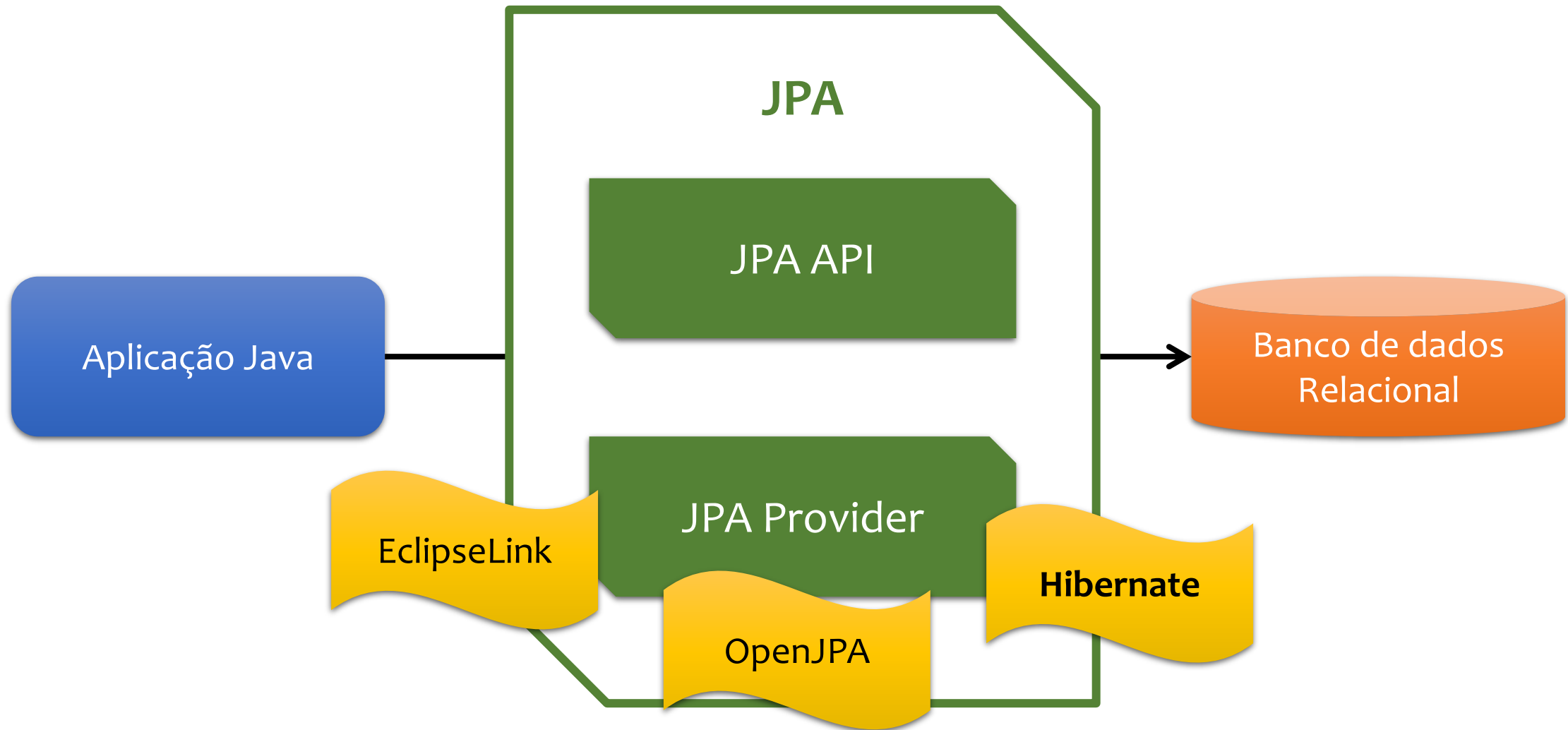
Configuração declarativa da conexão do BD

Recurso de Mapeamento Objeto-Relacional

Suporte à operações de CRUD

Fornecer uma linguagem de consulta OO







MOR Moderno e Elegante



Ciclo de Vida de Entidades

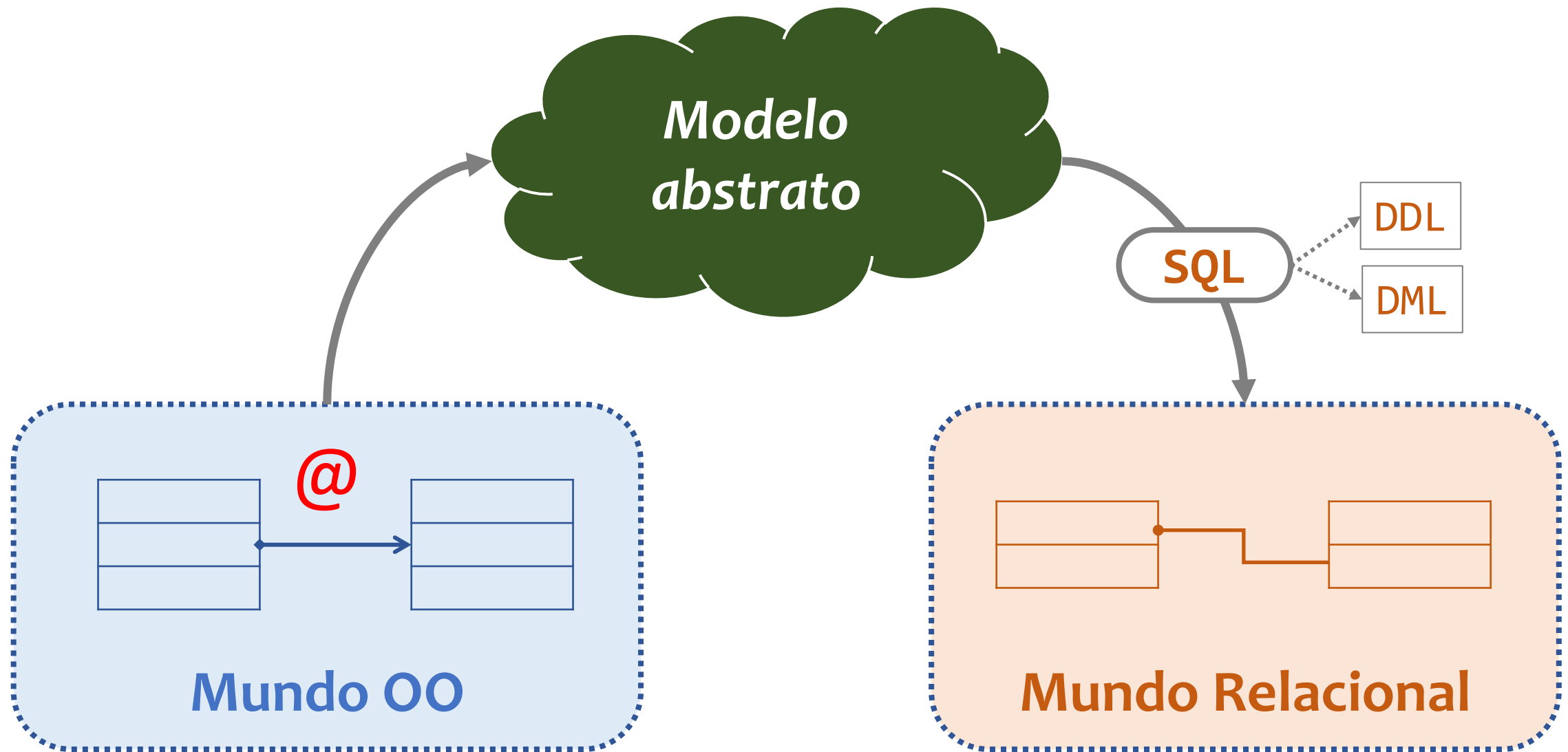


Relatórios e DTO



Melhores Práticas

Mapeamento Objeto-Relacional *Moderno e elegante*



*Bean Validation

@NotNull

@Size(max)

@Positive

...

*Estabelecer Padrões:

Nome de atributos

Nome de métodos

Modelagem UML

*JPA:

@Entity

@Id

@GeneratedValue

*Relacionamentos:

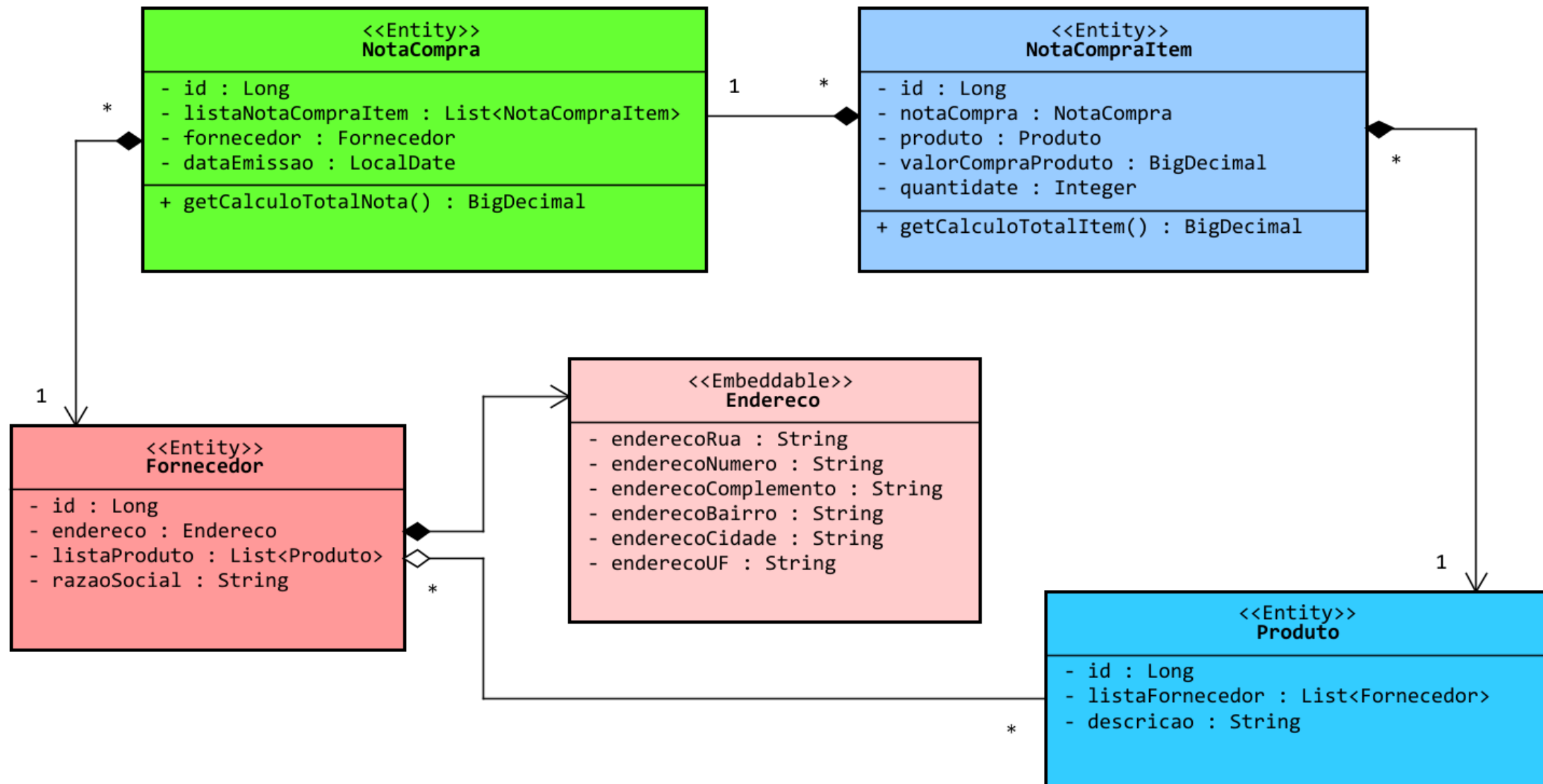
@ManyToOne

@OneToMany

@OneToOne

@ManyToMany

@Embeddable



> Em **start.spring.io**, configurar novo projeto para este LAB:

The screenshot shows the Spring Initializr configuration page. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.6.7' selected. The 'Project Metadata' section has the following values: Group: 'br.inatel.labs', Artifact: 'jpa', Name: 'LabJPA', Description: 'Lab JPA', and Package name: 'br.inatel.labs.jpa'. The 'Packaging' section has 'Jar' selected. The 'Java' section has '11' selected. The 'Dependencies' section has 'Spring Boot DevTools', 'Spring Web', 'Spring Reactive Web', 'Spring Data JPA', 'H2 Database', and 'Validation' selected. A button 'ADD DEPENDENCIES... CTRL + B' is visible in the top right of the dependencies section.

spring initializr

Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M2) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (RC1)
☐ 2.6.8 (SNAPSHOT) ☒ 2.6.7 ☐ 2.5.14 (SNAPSHOT) ☐ 2.5.13

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 18 ☐ 17 ☒ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Reactive Web WEB
Build reactive web applications with Spring WebFlux and Netty.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.


Validation I/O
Bean Validation with Hibernate validator.

>Abrir Eclipse no workspace: **c:\LABS\workspace_labs;**


>Importar projeto do Spring Boot;

>Configurar o acesso ao banco de dados H2:


>Editar **application.properties**:



```
# Datasource
spring.datasource.url=jdbc:h2:file:./h2/mydata
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
```



```
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2
```



```
# JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.open-in-view=false
```

>Criar sub-pacote: **.entity**

>Conforme o UML, criar as classes e declarar os atributos:

***Produto**

***Endereco**

***Fornecedor**

***NotaCompra**

***NotaCompraItem**

(*)Apenas declarar os atributos.

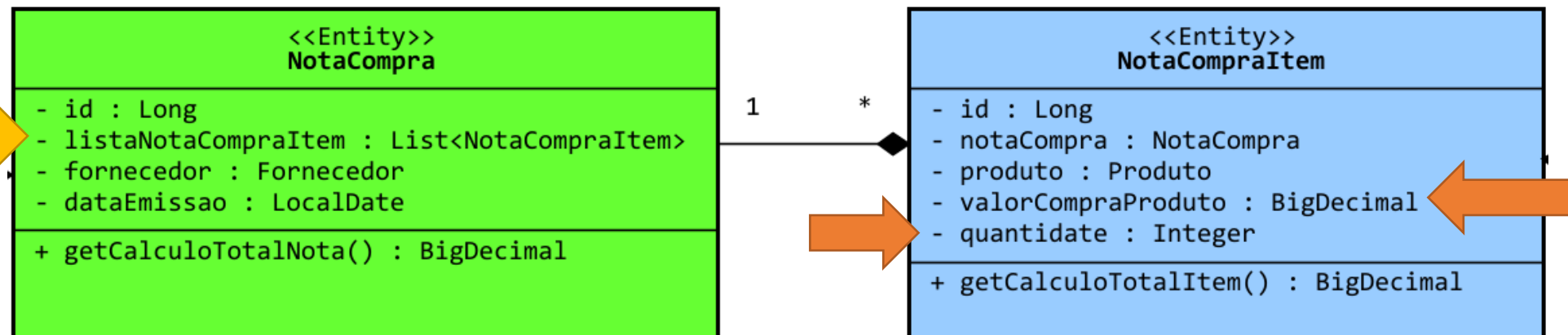
(**)Vamos gerar os construtores ou acessores depois:

*DESAFIO: Implementando os métodos de cálculo:

>Em **NotaCompraItem**, codar o método **getCalculoTotalItem()**

>Em **NotaCompra**, codar o **getCalculoTotalNota()**

(*)Use streams e lambdas



*Solução métodos de cálculo:

>NotaCompraItem:

```
public BigDecimal getCalculoTotal() {  
    return valorCompraProduto.multiply( BigDecimal.valueOf( quantidade ) );  
}
```

*Solução métodos de cálculo:

>NotaCompraItem:

```
public BigDecimal getCalculoTotal() {  
    return valorCompraProduto.multiply( BigDecimal.valueOf( quantidade ) );  
}
```

>NotaCompra:

```
public BigDecimal getCalculoTotalNota() {  
    BigDecimal total = this.listaNotaCompraItem.stream()  
        .map( i -> i.getCalculoTotal() )  
        .reduce( BigDecimal.ZERO, BigDecimal::add );  
  
    return total;  
}
```



Demonstração



Demonstração



Demonstração



Demonstração



Demonstração

**Utilizando os atalhos do Eclipse:*

- 1) Gerar os getters e setters
- 2) Gerar o `equals()` e `hashCode()` usando a chave primária

Opcional, mas é interessante:

- 3) Gerar o `toString()`

(*) Alguns construtores específicos ainda serão criados ao longo dos exercícios

Ciclo de Vida das Entidades

- *Máquina de Estados

- *Lazy Loading

Principal interface do
JPA

Cache L1

EntityManager

Gerência do Ciclo de
Vida de Entidades

Gerência do Contexto de
Persistência

EntityManager

`persist()`

`merge()`

`remove()`

`find()`

`createQuery()`



ProdutoService

- > Criar o sub-pacote **.service**
- > Criar a classe **ProdutoService**:
- > Declarar **EntityManager** como atributo privado e marcar sua injeção
- > Implementar os métodos:

Produto salvar(Produto p)

Produto buscarPeloId(Long id)

List<Produto> listar()

void remover(Produto p)

- (1) Tente fazer usando apenas com as informações deste slide
- (2) Uma vez pronto, consulte no próximo slide a solução

> Solução: classe ProdutoService:

```
@Service
@Transactional
public class ProdutoService {

    @PersistenceContext
    private EntityManager em;

    public Produto salvar(Produto p) {
        p = em.merge( p );
        return p;
    }

    public Produto buscarPeloId(Long id) {
        Produto p = em.find(Produto.class, id);
        return p;
    }

    public List<Produto> listar() {
        List<Produto> listaProduto = em.createQuery("select p from Produto p", Produto.class)
            .getResultList();
        return listaProduto;
    }

    public void remover(Produto p) {
        p = em.merge( p );
        em.remove( p );
    }
}
```

Carga inicial de Produtos

1) Abrir a classe **Produto**:

- a) Gerar construtor que recebe descricao
- b) Gerar construtor padrão

2) Abrir a classe JpaApplicationTests:

- a) Renomear a classe para **DataLoader**
- b) Renomear o único método para load()

Carga inicial de Produtos

3) Em **DataLoader**, codar a carga dos dados:

a) Na classe:

```
@SpringBootTest
class DataLoader {

    @Autowired
    private ProdutoService produtoService;
```

b) No método load():

```
//1.produto
Produto p1 = new Produto("Furadeira");
Produto p2 = new Produto("Lixadeira");
Produto p3 = new Produto("Plaina");
Produto p4 = new Produto("Tupia");
Produto p5 = new Produto("Serra Circular");

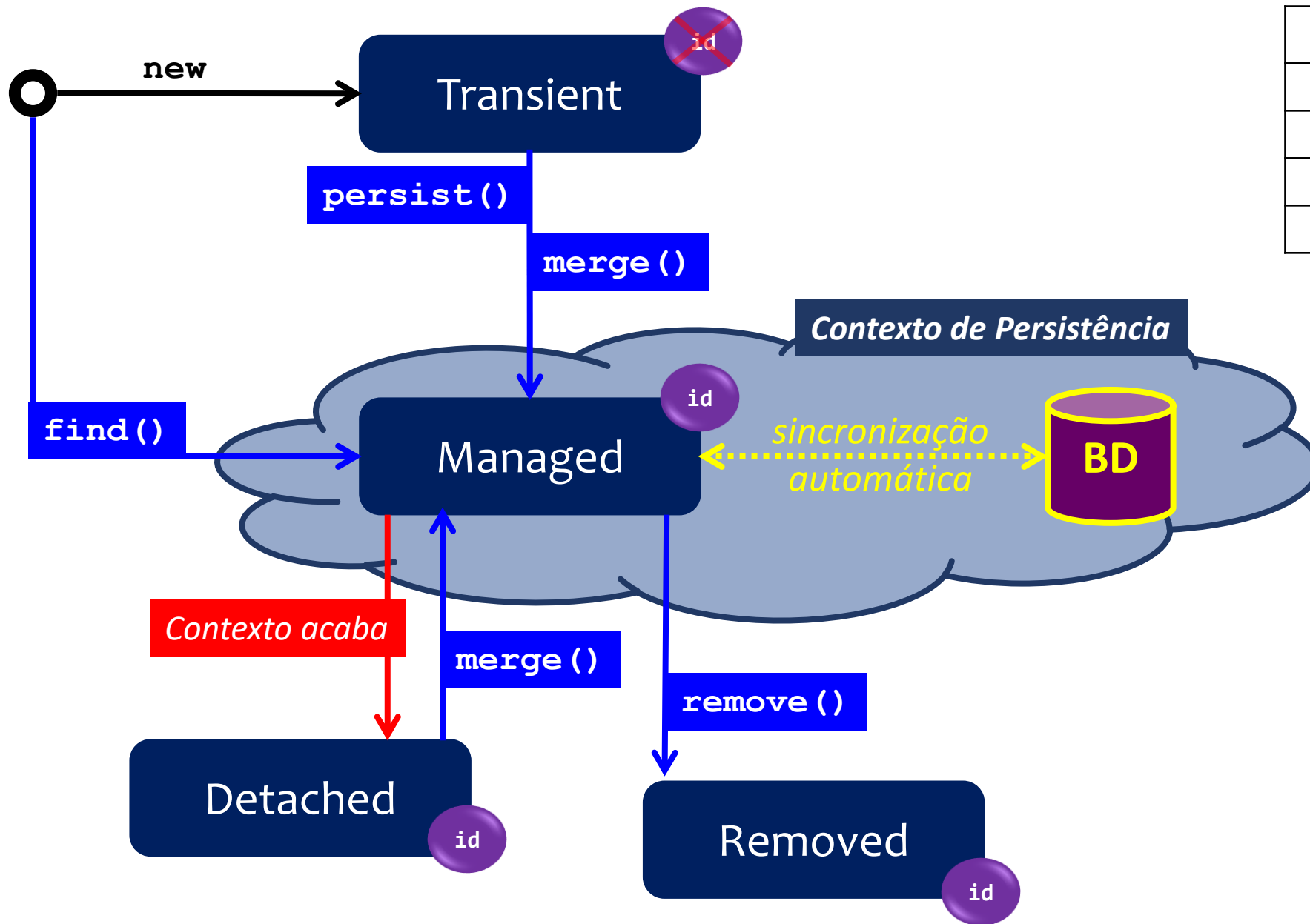
p1 = produtoService.salvar( p1 );
p2 = produtoService.salvar( p2 );
p3 = produtoService.salvar( p3 );
p4 = produtoService.salvar( p4 );
p5 = produtoService.salvar( p5 );

List<Produto> listaProduto = produtoService.listar();
listaProduto.forEach( System.out::println );
```

Atenção aqui.

EntityManager	SQL
<code>persist</code>	INSERT
<code>merge</code>	UPDATE
<code>remove</code>	DELETE
<code>find</code>	SELECT





EntityManager
<code>persist()</code>
<code>merge()</code>
<code>remove()</code>
<code>find()</code>

Quando o **Contexto de Persistência** é criado e destruído?

Quando o *Contexto de Persistência* é criado e destruído?

Em todos os métodos
transacionais da classe de service

```
@Service
@Transactional
public class ProdutoService {

    @PersistenceContext
    private EntityManager em;

    public Produto salvar(Produto p) {
        p = em.merge( p );
        return p;
    }

    public Produto buscarPeloId(Long id) {
        Produto p = em.find(Produto.class, id);
        return p;
    }

    public List<Produto> listar() {
        List<Produto> listaProduto = em.createQuery("select p from Produto p", Produto.class)
            .getResultList();
        return listaProduto;
    }

    public void remover(Produto p) {
        p = em.merge( p );
        em.remove( p );
    }
}
```

FornecedorService

- > No pacote `.service`
- > Criar a classe **FornecedorService**:
- > Declarar **EntityManager** como atributo privado e marcar sua injeção
- > Implementar os métodos:

Fornecedor salvar(Fornecedor f)

Fornecedor buscarPeloId(Long id)

List<Fornecedor> listar()

void remover(Fornecedor f)

> Solução: classe FornecedorService:

```
@Service
@Transactional
public class FornecedorService {

    @PersistenceContext
    private EntityManager em;

    public Fornecedor salvar(Fornecedor f) {
        f = em.merge( f );
        return f;
    }

    public Fornecedor buscarPeloId(Long id) {
        Fornecedor f = em.find(Fornecedor.class, id);
        return f;
    }

    public List<Fornecedor> listar() {
        List<Fornecedor> listaFornecedor = em.createQuery("select f from Fornecedor f", Fornecedor.class)
            .getResultList();
        return listaFornecedor;
    }

    public void remover(Fornecedor f) {
        f = em.merge( f );
        em.remove( f );
    }
}
```

Carga inicial de Fornecedor

1) Abrir Fornecedor:

- a) Declarar construtor que recebe razaoSocial
- b) Declarar construtor padrão

2) Assegurar que o atributo Endereco é instanciado previamente

Carga inicial de Fornecedor

3) Em DataLoader:

a) Na classe:

```
@Autowired
private FornecedorService fornecedorService;
```

b) No método load():

```
//2.fornecedor
Fornecedor f1 = new Fornecedor("Gasômetro Madeiras");
f1.getEndereco().setEnderecoRua("Avenida Pinto Cobra");
f1.getEndereco().setEnderecoNumero("110");
f1.getEndereco().setEnderecoBairro("Vila Mariana");
f1.getEndereco().setEnderecoCidade("Pouso Alegre");
f1.getEndereco().setEnderecoUF("MG");

Fornecedor f2 = new Fornecedor("Loja do Mecânico");
f2.getEndereco().setEnderecoRua("Av Reinaldo Chioca");
f2.getEndereco().setEnderecoNumero("1922");
f2.getEndereco().setEnderecoBairro("Parque Progresso");
f2.getEndereco().setEnderecoCidade("Franca");
f2.getEndereco().setEnderecoUF("SP");

f1 = fornecedorService.salvar( f1 );
f2 = fornecedorService.salvar( f2 );

List<Fornecedor> listaFornecedor = fornecedorService.listar();
listaFornecedor.forEach( System.out::println );
```

NotaCompraService

> No pacote **.service**

> Criar a classe **NotaCompraService**:

> Ela servirá para **NotaCompra** e **NotaCompraItem**

> Implementar os métodos:

NotaCompra buscarNotaCompraPeloId(Long id)

NotaCompra salvar(NotaCompra nota) {

List<NotaCompra> listarNotaCompra()

NotaCompraItem buscarNotaCompraItemPeloId(Long id)

NotaCompraItem salvar(NotaCompraItem item)

List<NotaCompraItem> listarNotaCompraItem()

> Solução: classe NotaCompraService:

```
@Service
@Transactional
public class NotaCompraService {

    @PersistenceContext
    private EntityManager em;

    // nota compra

    public NotaCompra buscarNotaCompraPeloId(Long id) {
        return em.find(NotaCompra.class, id);
    }

    public NotaCompra salvar(NotaCompra nota) {
        return em.merge( nota );
    }

    public List<NotaCompra> listarNotaCompra() {
        return em.createQuery("select n from NotaCompra n", NotaCompra.class)
            .getResultList();
    }

    //nota compra item

    public NotaCompraItem salvar(NotaCompraItem item) {
        return em.merge( item );
    }

    public NotaCompraItem buscarNotaCompraItemPeloId(Long id) {
        return em.find(NotaCompraItem.class, id);
    }

    public List<NotaCompraItem> listarNotaCompraItem() {
        return em.createQuery("select i from NotaCompraItem i", NotaCompraItem.class)
            .getResultList();
    }
}
```

Carga inicial de NotaCompra e itens - 1/3

1) Em NotaCompra:

- a) Declarar construtor que recebe dataEmissao e fornecedor
- b) Declarar construtor padrão

2) Em NotaCompraItem:

- a) Declarar construtor que recebe notaCompra, produto, valorCompradoProduto e quantidade
- a) Declarar construtor padrão

Carga inicial de NotaCompra e itens - 2/3

3) Em **DataLoader**, implementar da carga de notas

a) Na classe:

```
@Autowired  
private NotaCompraService notaCompraService;
```

b) No método load(), carregar notas:

```
//3.Nota Compra  
NotaCompra nc1 = new NotaCompra( LocalDate.of(2021, 1, 15), f1);  
nc1 = notaCompraService.salvar(nc1);  
  
NotaCompra nc2 = new NotaCompra( LocalDate.of(2022, 2, 20), f2);  
nc2 = notaCompraService.salvar(nc2);  
  
notaCompraService.listarNotaCompra().forEach( System.out::println );
```


Carga inicial de NotaCompra e itens - 3/3

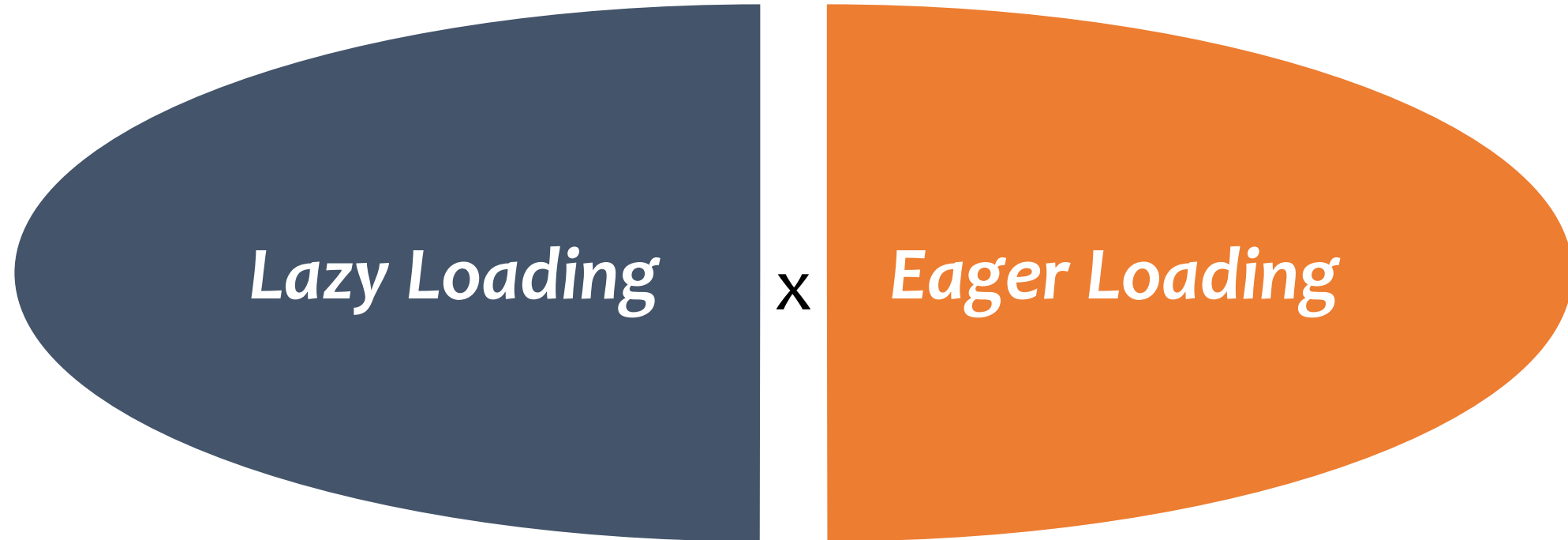
3) Em **DataLoader**, implementar da carga de itens:

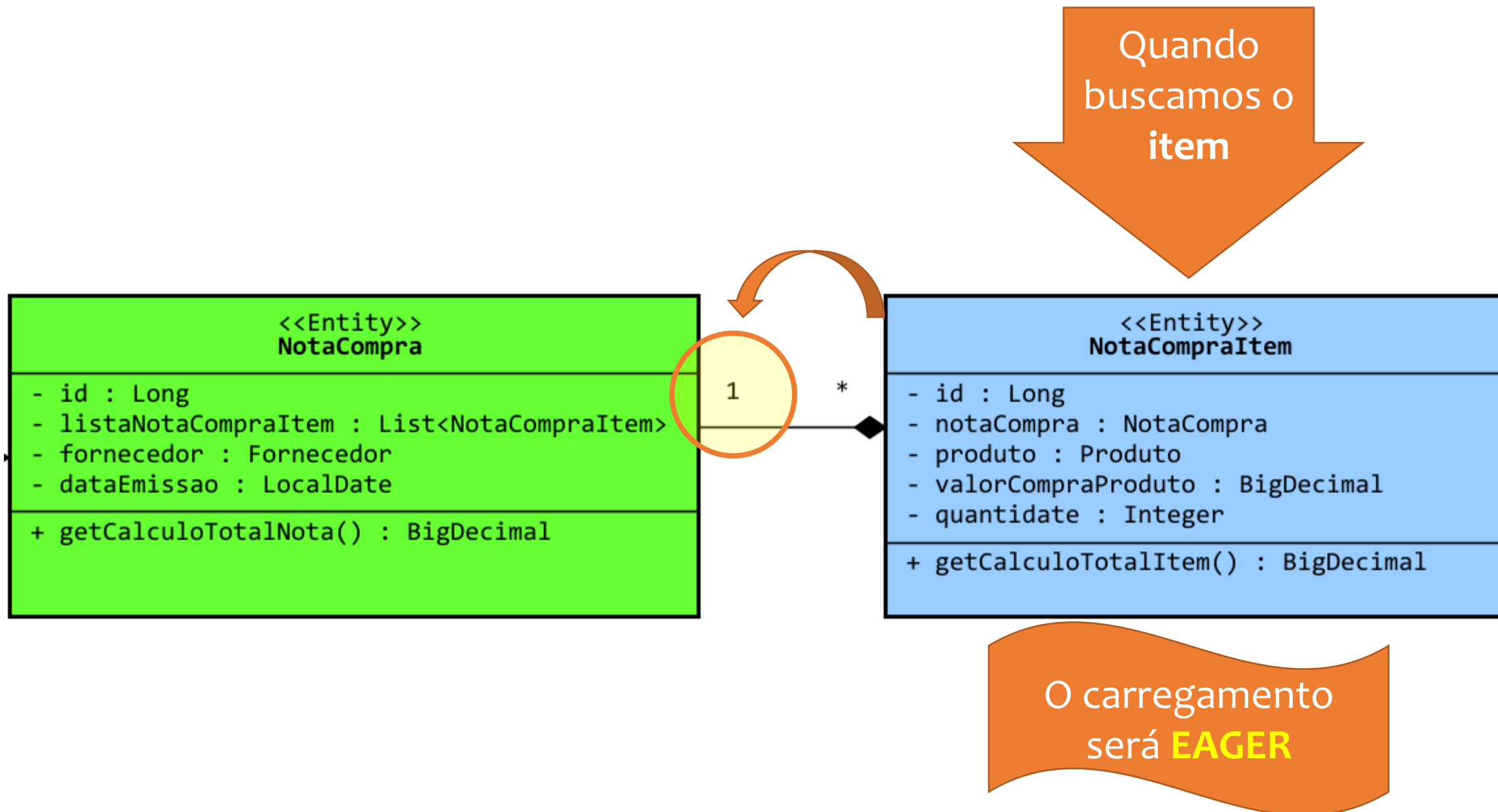
c) No método load(), carregar itens:

```
//4. Nota Compra Item
NotaCompraItem i1_1 = new NotaCompraItem(nc1, p1, new BigDecimal("300.00"), 2);
NotaCompraItem i1_2 = new NotaCompraItem(nc1, p2, new BigDecimal("1000.00"), 1);
NotaCompraItem i1_3 = new NotaCompraItem(nc1, p3, new BigDecimal("500.00"), 3);
i1_1 = notaCompraService.salvar(i1_1);
i1_2 = notaCompraService.salvar(i1_2);
i1_3 = notaCompraService.salvar(i1_3);

NotaCompraItem i2_1 = new NotaCompraItem(nc2, p4, new BigDecimal("400.00"), 7);
NotaCompraItem i2_2 = new NotaCompraItem(nc2, p2, new BigDecimal("1000.00"), 2);
NotaCompraItem i2_3 = new NotaCompraItem(nc2, p5, new BigDecimal("700.00"), 1);
i2_1 = notaCompraService.salvar(i2_1);
i2_2 = notaCompraService.salvar(i2_2);
i2_3 = notaCompraService.salvar(i2_3);

notaCompraService.listarNotaCompraItem().forEach( System.out::println );
```





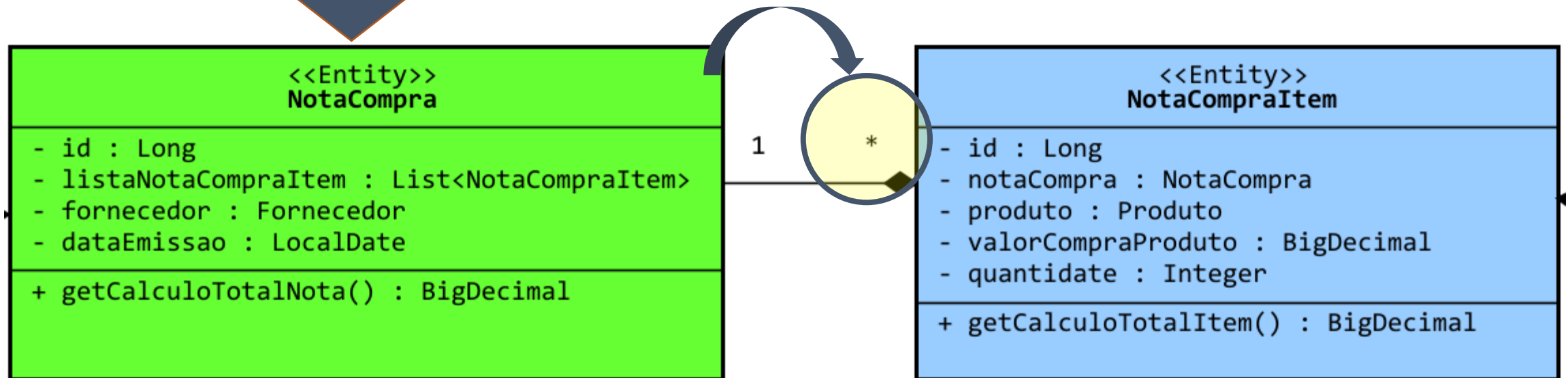


Demonstração
EAGER

EAGER

```
try {  
    NotaCompraItem item = service.buscarNotaCompraItemPeloId( 1L );  
  
    System.out.println( item.getNotaCompra().getDataEmissao() );  
  
    System.out.println("ok ");  
  
} catch (Exception e) {  
    System.out.println( e.getMessage() );  
    e.printStackTrace();  
}
```

Quando
buscamos a
Nota



O carregamento
será **LAZY**

Demonstração

LAZY

LAZY

```
try {  
    NotaCompra nota = servico.buscarNotaCompraPeloId( 1L );  
  
    nota.getListaNotaCompraItem().forEach( System.out::println );  
  
    System.out.println("rodou ok");  
} catch (Exception e) {  
    System.out.println( e.getMessage() );  
    e.printStackTrace();  
}
```


LazyInitializationException

```

Console X
C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (Apr 26, 2022, 8:22:50 PM - 8:23:22 PM)
2022-04-26 20:22:57.555 INFO 16664 --- [main] o.s.b.a.nz.nzConsoleAutoConfiguration : nz console available at /nz
2022-04-26 20:22:58.138 INFO 16664 --- [main] br.inatel.idplabs.jpa.LazyLoadingDemo : Started LazyLoadingDemo in 6.14 seconds (JVM running for 7.476)
org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: br.inatel.idplabs.jpa.entity.NotaCompra.listaNotaCompraItem, could not initialize proxy - no Session
    at org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:614)
    at org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:218)
    at org.hibernate.collection.internal.AbstractPersistentCollection.readSize(AbstractPersistentCollection.java:162)
    at org.hibernate.collection.internal.PersistentBag.size(PersistentBag.java:371)
    at br.inatel.idplabs.jpa.LazyLoadingDemo.test(LazyLoadingDemo.java:28)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:566)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
    at org.junit.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod$2.runInThread(FrameworkMethod.java:153)
    at java.base/java.lang.Thread.run(Thread.java:833)
  
```



```

Console X
C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (Apr 26, 2022, 8:22:50 PM - 8:23:22 PM)
2022-04-26 20:22:57.555 INFO 16664 --- [main] o.s.b.a.nz.nzConsoleAutoConfiguration : nz console available at /nz
2022-04-26 20:22:58.138 INFO 16664 --- [main] br.inatel.idplabs.jpa.LazyLoadingDemo : Started LazyLoadingDemo in 6.14 seconds (JVM running for 7.476)
org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: br.inatel.idplabs.jpa.entity.NotaCompra.listaNotaCompraItem, could not initialize proxy - no Session
    at org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:614)
    at org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:218)
    at org.hibernate.collection.internal.AbstractPersistentCollection.readSize(AbstractPersistentCollection.java:162)
    at org.hibernate.collection.internal.PersistentBag.size(PersistentBag.java:371)
    at br.inatel.idplabs.jpa.LazyLoadingDemo.test(LazyLoadingDemo.java:28)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:566)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
    at org.junit.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod$2.runInThread(FrameworkMethod.java:153)
    at java.base/java.lang.Thread.run(Thread.java:833)
  
```

```

Database available at jdbc:nz:rie:./nz/mydata
4 seconds (JVM running for 7.476)
a.listaNotaCompraItem, could not initialize proxy - no Session
    at org.hibernate.collection.internal.AbstractPersistentCollection.throwLazyInitializationException(AbstractPersistentCollection.java:614)
    at org.hibernate.collection.internal.AbstractPersistentCollection.withTemporarySessionIfNeeded(AbstractPersistentCollection.java:218)
    at org.hibernate.collection.internal.AbstractPersistentCollection.readSize(AbstractPersistentCollection.java:162)
    at org.hibernate.collection.internal.PersistentBag.size(PersistentBag.java:371)
    at br.inatel.idplabs.jpa.LazyLoadingDemo.test(LazyLoadingDemo.java:28)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:566)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
    at org.junit.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod$2.runInThread(FrameworkMethod.java:153)
    at java.base/java.lang.Thread.run(Thread.java:833)
  
```

“no Session”
significa
“não há contexto de
persistência ativo”
ou
“entidade está no estado
Detached”



@OneToMany (fetch=FetchType.EAGER)



Design Pattern OpenEntityManagerInView



Planejando as consultas -> criar **métodos de serviço** especializados


```
@Service
@Transactional
public class NotaCompraService {

    @PersistenceContext
    private EntityManager em;

    // nota compra

    public NotaCompra buscarNotaCompraPeloId(Long id) {
        return em.find(NotaCompra.class, id);
    }

    public NotaCompra buscarNotaCompraPeloIdComListaItem(Long id) {
        NotaCompra nota = em.find(NotaCompra.class, id);
        nota.getListaNotaCompraItem().size(); // força o carregamento estando em Managed
        return nota;
    }
}
```



Contexto de persistência é **ativado** a cada invocação de método **transacional**

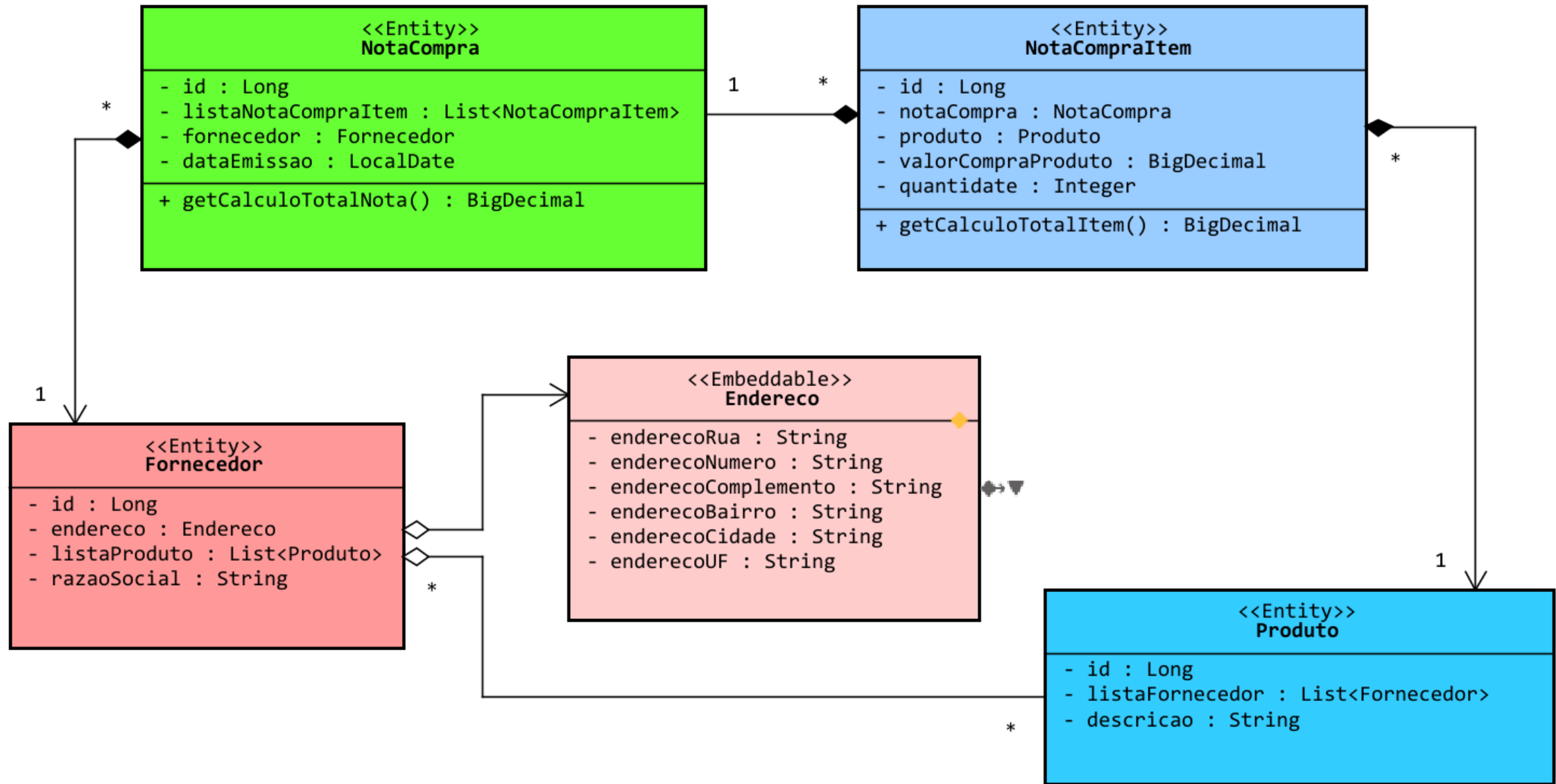
Demonstração

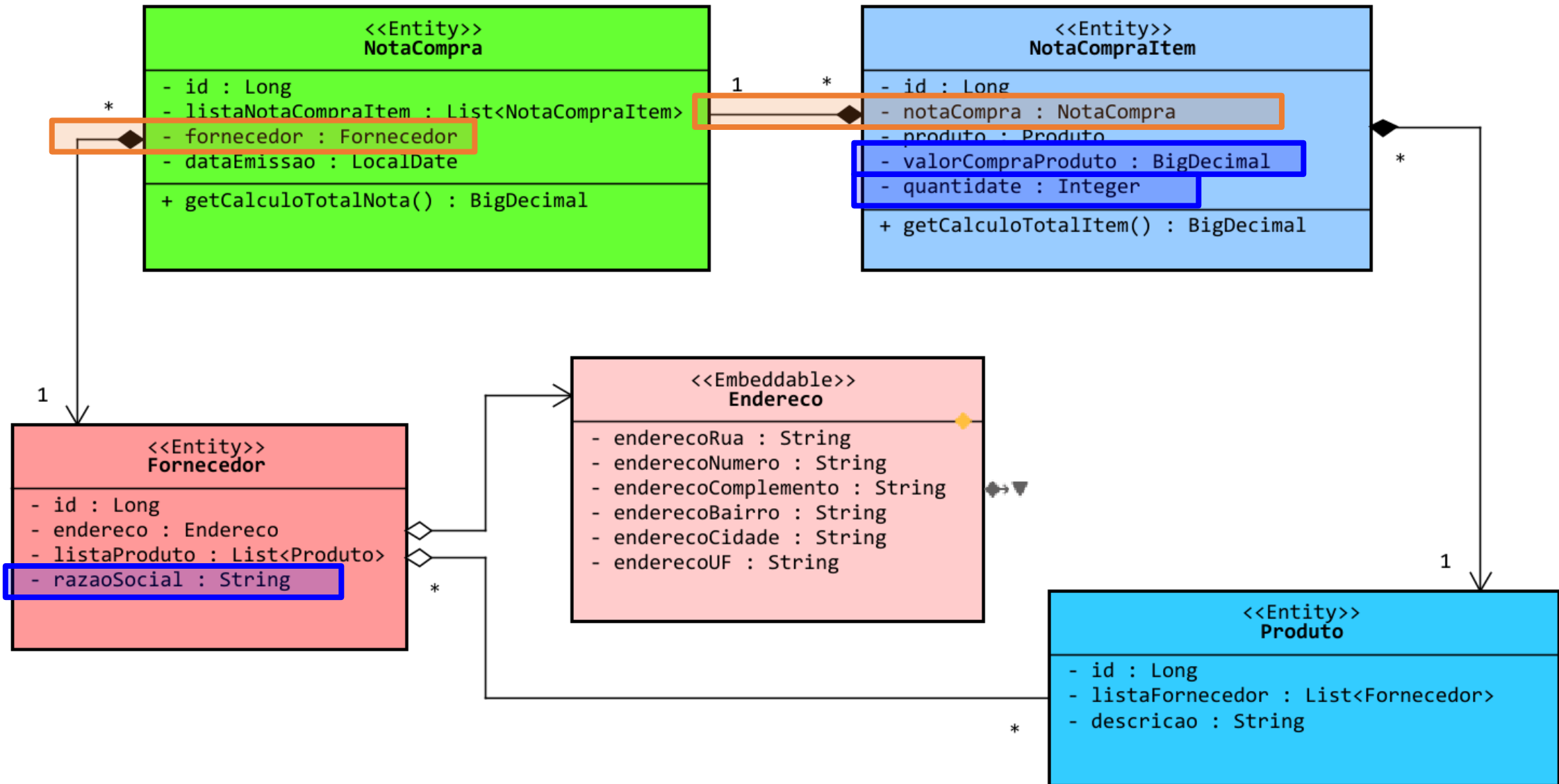
Planejando as consultas

Relatórios e DTO (Data Transfer Object)

Relatório de Total Comprado de Produtos por Fornecedor

Fornecedor Razão Social	Total comprado
Casa do Mecânico	15.080,50
Gasômetro Madeiras	10.250,00
Palácio das Ferramentas	2.300,00





```
select f.razaoSocial  
      , sum(i.quantidade * i.valorCompraProduto)  
from NotaCompraItem i  
     join i.notaCompra n  
     join n.fornecedor f  
group by f.razaoSocial
```

Como recuperar o resultado da query?

***Design Pattern
DTO***

- 1) Criar o **DTO** com os campos da query
 - a) Mesmos atributos da query
 - b) Construtor recebendo estes atributos
 - c) Getters já são suficientes

```
public class TotalCompradoPorFornecedorDTO {  
  
    private String fornecedorRazaoSocial;  
  
    private BigDecimal totalComprado;  
  
    public TotalCompradoPorFornecedorDTO(String fornecedorRazaoSocial, BigDecimal totalComprado) {  
        this.fornecedorRazaoSocial = fornecedorRazaoSocial;  
        this.totalComprado = totalComprado;  
    }  
  
    //getters são suficientes...
```

(*)é possível também declarar os atributos como final

2) Declarar o método de pesquisa:

a) Deve retornar uma lista do DTO

b) Declarar a query numa variável String

c) Usar `EntityManager.createQuery(...)`

```
public List<TotalCompradoPorFornecedorDTO> pesquisarRelatorioTotalCompradoPorFornecedor() {  
    String query  
        = " select f.razaoSocial "  
        + "      , sum( i.quantidade * i.valorCompraProduto )"   
        + " from NotaCompraItem  i "  
        + "      join i.notaCompra  n "  
        + "      join n.fornecedor  f "  
        + " group by f.razaoSocial ";  
  
    return em.createQuery(query, TotalCompradoPorFornecedorDTO.class)  
        .getResultList();  
}
```

3) Embutir o construtor do DTO na query:

a) Usar o caminho full-qualified do DTO

b) Cuidado com a posição dos atributos com o construtor do DTO

```
public List<TotalCompradoPorFornecedorDTO> pesquisarRelatorioTotalCompradoPorFornecedor() {  
    String query  
        = " select new br.inatel.idplabs.jpa.entity.dto.TotalCompradoPorFornecedorDTO "  
        + "      ( f.razaoSocial "  
        + "        , sum( i.quantidade * i.valorCompraProduto ) "  
        + "      ) "  
        + " from NotaCompraItem  i "  
        + "      join i.notaCompra  n "  
        + "      join n.fornecedor  f "  
        + " group by f.razaoSocial ";  
  
    return em.createQuery( query, TotalCompradoPorFornecedorDTO.class )  
        .getResultList();  
}
```

- > Criar o sub-pacote **.dto**;
- > Criar a classe **TotalCompradoPorFornecedorDTO**
 - > Declarar os atributos
 - > Declarar o constructor
 - > Declarar os getters

```
public class TotalCompradoPorFornecedorDTO {  
  
    private String fornecedorRazaoSocial;  
  
    private BigDecimal totalComprado;  
  
    public TotalCompradoPorFornecedorDTO(String fornecedorRazaoSocial, BigDecimal totalComprado) {  
        this.fornecedorRazaoSocial = fornecedorRazaoSocial;  
        this.totalComprado = totalComprado;  
    }  
  
    //getters são suficientes...
```

>No pacote **service**, criar a classe **RelatorioService**

```
@Service
public class RelatorioService {

    @PersistenceContext
    private EntityManager em;
```


>Na classe **RelatorioService**

>Implementar o método de pesquisar:

```
public List<TotalCompradoPorFornecedorDTO> pesquisarRelatorioTotalCompradoPorFornecedor() {  
    String query  
        = " select new br.inatel.idplabs.jpa.entity.dto.TotalCompradoPorFornecedorDTO "  
        + "      ( f.razaoSocial "  
        + "      , sum( i.quantidade * i.valorCompraProduto )"   
        + "      ) "  
        + " from NotaCompraItem  i "  
        + "      join i.notaCompra  n "  
        + "      join n.fornecedor  f "  
        + " group by f.razaoSocial ";  
  
    return em.createQuery( query, TotalCompradoPorFornecedorDTO.class)  
        .getResultList();  
}
```

- > Criar a classe de teste **RelatorioServiceTest**
- > Escrever um método de teste invocando o método de pesquisa:

```
@SpringBootTest
public class RelatorioServiceTest {

    @Autowired
    private RelatorioService service;

    @Test
    void test() {

        List<TotalCompradoPorFornecedorDTO> listaDTO = service.pesquisarRelatorioTotalCompradoPorFornecedor();

        listaDTO.forEach( System.out::println );

    }
}
```

JPA / Hibernate / Spring Data

Melhores práticas

MOR

*Sempre reescreva os métodos equals e hashCode nas entitys, usando o atributo anotado com @Id

Hibernate

*De verdade, **NUNCA** use `@OneToMany(fetch=FetchType.EAGER)`

*Ainda que pareça simples, isolado ou irrelevante

*No `application.properties`, sempre desabilitar o open-in-view:

`spring.jpa.open-in-view=false`

Spring Data e DTO

- * Usamos **@Query** para declarar query personalizadas no Spring Data
- * Se a query tiver algum erro de sintaxe, será lançado **erro de deployment**
- * A mesma pesquisa usando repository do Spring Data:

```
@Repository
public interface RelatorioRepository extends JpaRepository<NotaCompraItem, Long> {

    @Query( " select new br.inatel.idplabs.jpa.entity.dto.TotalCompradoPorFornecedorDTO "
        + "      ( f.razaoSocial "
        + "      , sum( i.quantidade * i.valorCompraProduto )"
        + "      ) "
        + " from NotaCompraItem i "
        + "   join i.notaCompra n "
        + "   join n.fornecedor f "
        + " group by f.razaoSocial "
        + " ) "
    public List<TotalCompradoPorFornecedorDTO> pesquisarTotalCompradoPorFornecedor();
```

