Tópicos em Frameworks

Cássio Seffrin

O que é um Framework?

- Abstração de código entre vários projetos.
- Frameworks VS Bibliotecas (classes)
- exemplos de bibliotecas: jquery, lodash, underscore …

Framework...

- Um framework em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle.
  Fonte: wiki

- Todo o fluxo de controle já está lá, existem brechas predefinidas onde devemos preencher com nosso código.

- Um framework é normalmente mais complexo. Ele vai controlar o todo fluxo, seu código será chamado pela estrutura quando apropriado.

- O benefício é que os desenvolvedores não precisam se preocupar com o design, mas apenas sobre a implementação de funções específicas.

- Estes fatores nos fazem pensar muito antes de escolher o framework mais adequado a nosso projeto.

Exemplos

Vert.X
Spring MVC – Model View Controller
Spring (POA)
Spring Data (https://spring.io/projects/spring-data)
GWT – Google Web Toolkit
Grails Web Framework
Struts
JSF – JavaServer Faces
Cake PHP
Zend Framework
Laravel
Hibernate
Doctrine
Flyway - database migration / liquibase
Eureka - Service Discovery Framework
JUnit

Principais Vantagens

- As Principais vantagens para o uso de framework são:

- Utilidade: O objetivo primeiro dos frameworks é auxiliar no desenvolvimento de aplicações e softwares. Para tal, eles têm funcionalidades nativas das mais variadas, que ajudam você a resolver as questões sobre programação do dia-a-dia com muito mais qualidade e eficiência.

- Segurança: Os bons frameworks são projetados de modo a garantir a segurança de quem programa e, principalmente, de quem usa o que foi feito a partir dele. Não se preocupe mais com aquelas intermináveis linhas de código para evitar um SQL Injection, por exemplo; com frameworks, a parte de segurança já “vem de fábrica”.

- Extensibilidade: Os frameworks permitem que você entenda suas funcionalidades nativas. Se aquela biblioteca de envio de e-mails por SMTP não contempla todas as possibilidades que você gostaria, simplesmente entenda suas funcionalidades e as use como se fossem parte do framework (na verdade, elas serão).

- Economia de tempo: O que você demoraria algumas horas ou alguns dias para fazer, você encontra pronto em um framework. Pense no quão são trabalhosas aquelas funções de manipulação de imagens são; usando um framework que tenha isso, você só usa, e pronto.

um pouco de prática

- https://spring.io/tools
- mysql
- git

Spring Boot

O Spring Boot facilita a criação de aplicativos baseados em Spring autônomos e de produção que você pode "executar".

Adotamos uma visão opinativa da plataforma Spring e de bibliotecas de terceiros para que você possa começar com o mínimo de stress. A maioria dos aplicativos Spring Boot precisa de uma configuração de Spring muito pequena.

Características

- Crie aplicativos Spring independentes
- Incorpore Tomcat, Jetty ou Undertow diretamente (não é necessário implantar arquivos WAR)
- Forneça dependências 'iniciais' com opinião para simplificar sua configuração de criação
- Configurar automaticamente bibliotecas Spring e de terceiros sempre que possível
- Fornecer recursos prontos para produção, como métricas, verificações de integridade e configuração externalizada
- Absolutamente nenhuma geração de código e nenhum requisito para configuração XML

Projeto Escola

- https://start.spring.io/

- Spring Data JPA

- Spring Web Starter

- Mysql Driver

- Lombok
  https://projectlombok.org/downloads/lombok.jar

Spring Framework

Spring Framework

Origem: Wikipédia, a enciclopédia livre.

- O Spring é um framework open source para a plataforma Java criado por Rod Johnson e descrito em seu livro "Expert One-on-One: JEE Design e Development". Trata-se de um framework não intrusivo, baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência.

- No Spring o container se encarrega de "instanciar" classes de uma aplicação Java e definir as dependências entre elas através de um arquivo de configuração em formato XML, inferências do framework, o que é chamado de auto-wiring ou ainda anotações nas classes, métodos e propriedades. Dessa forma o Spring permite o baixo acoplamento entre classes de uma aplicação orientada a objetos.

- O Spring possui uma arquitetura baseada em interfaces e POJOs (Plain Old Java Objects), oferecendo aos POJOs características como mecanismos de segurança e controle de transações. Também facilita testes unitários e surge como uma alternativa à complexidade existente no uso de EJBs. Com Spring, pode-se ter um alto desempenho da aplicacao.

- Esse framework oferece diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como módulos voltados para desenvolvimento Web, persistência, acesso remoto
e programação orientada a aspectos.

JPA

- Aplicações corporativas manipulam dados em grande quantidade

- Esses dados são em sua maioria armazenados em banco de dados relacionais

- As aplicações corporativas costumam ser desenvolvidas com linguagens orientadas a objetos

- O modelo relacional e o modelo orientado a objetos diferem no modo de estruturar os dados

- Transformações devem ocorrer toda vez que uma informação trafegar da aplicação para o banco de dados ou vice-versa

JPA

- O JPA é o padrão de mapeamento objeto/entidade relacionamento e interface de gerência de persistência do JEE.

- Implementações do padrão JPA
  - Hibernate
  - EclipseLink
  - OpenJPA
  - …

JPA - ManyToOne

Anotação @JoinColumn

Em um relacionamento um para muitos / muitos para um, o lado proprietário geralmente é definido no lado 'muitos' do relacionamento. Geralmente é o lado que possui a chave estrangeira.

A anotação @JoinColumn define o mapeamento físico real no lado proprietário:

Atributo mappedBy

Depois de definirmos o lado proprietário do relacionamento, o Hibernate já possui todas as informações necessárias para mapear esse relacionamento em nosso banco de dados. Para tornar essa associação bidirecional, tudo o que precisamos fazer é definir o lado da referência. O lado inverso ou de referência simplesmente mapeia para o lado proprietário.

Podemos facilmente usar o atributo mappedBy da anotação @OneToMany para fazer isso. Então, vamos definir nossa entidade Turma:

o valor de mappedBy é o nome do atributo de mapeamento de associação no lado proprietário. Com isso, agora estabelecemos uma associação bidirecional entre nossos alunos e turmas.

JPA - ManyToOne

Aluno

```
    @ManyToOne(optional = false)
    private Turma turma;
```

Turma

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "turma")
    private Collection<Aluno> alunoCollection;
```

```
mysql> desc aluno;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| id       | int(11)      | NO   | PRI | NULL    |       |
| email    | varchar(255) | YES  |     | NULL    |       |
| nome     | varchar(255) | YES  |     | NULL    |       |
| turma_id | int(11)      | NO   | MUL | NULL    |       |
+----------+--------------+------+-----+---------+-------+

mysql> desc turma;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | NO   | PRI | NULL    |       |
| nome  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

OneToOne

- Implementando com uma chave estrangeira na JPA

- Colocamos a anotação @OneToOne no campo da entidade relacionada, Aluno.

- Além disso, precisamos colocar a anotação @JoinColumn para configurar o nome da coluna na tabela de usuários que mapeia para a chave primária na tabela de endereços. Se não fornecermos um nome, o Hibernate seguirá algumas regras para selecionar um padrão.

- Por fim, observe na próxima entidade que não usaremos a anotação @JoinColumn lá. Isso ocorre porque precisamos apenas do lado proprietário do relacionamento de chave estrangeira. Simplificando, quem possui a coluna de chave estrangeira recebe a anotação @JoinColumn.

- A entidade Endereco é mais simples:
  Também precisamos colocar a anotação @OneToOne. Isso porque esse é um relacionamento bidirecional. O lado do endereço do relacionamento é chamado de lado não proprietário.

OneToOne

- Aluno

```java
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "endereco_id", referencedColumnName = "id")
private Endereco endereco;
```

- Endereco

```java
@OneToOne(mappedBy = "endereco")
private Aluno aluno;
```

```
desc aluno;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id          | int(11)      | NO   | PRI | NULL    |       |
| email       | varchar(255) | YES  |     | NULL    |       |
| nome        | varchar(255) | YES  |     | NULL    |       |
| turma_id    | int(11)      | NO   | MUL | NULL    |       |
| endereco_id | int(11)      | YES  | MUL | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
desc endereco;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| id     | int(11)      | NO   | PRI | NULL    |       |
| bairro | varchar(255) | YES  |     | NULL    |       |
| cidade | varchar(255) | YES  |     | NULL    |       |
| rua    | varchar(255) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

JPA - ManyToMany

Turma:

```
@JoinTable(name = "turma_disciplina", joinColumns = {
        @JoinColumn(name = "turma_id", referencedColumnName =
"id") },
                    inverseJoinColumns = {
                @JoinColumn(name = "disciplina_id",
                referencedColumnName = "id") }
                )
    @ManyToMany
    private Collection<Disciplina> disciplinaCollection;
```

Disciplina:

```
@ManyToMany(mappedBy = "disciplinaCollection")
     private Collection<Turma> turmaCollection;
```

JPA - ManyToMany

Resultado:

```
desc turmas_disciplinas;
+---------------+---------+------+-----+---------+-------+
| Field         | Type    | Null | Key | Default | Extra |
+---------------+---------+------+-----+---------+-------+
| turma_id      | int(11) | NO   | MUL | NULL    |       |
| disciplina_id | int(11) | NO   | MUL | NULL    |       |
+---------------+---------+------+-----+---------+-------+
```

```
desc turma;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | NO   | PRI | NULL    |       |
| nome  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

```
desc disciplina;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id              | int(11)      | NO   | PRI | NULL    |       |
| carga_horaria   | int(11)      | YES  |     | NULL    |       |
| nome_disciplina | varchar(255) | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+\
```

JPA - ManyToMany

Como podemos ver, tanto a classe Turma quanto a Disciplina se referem uma à outra, o que significa que a associação entre elas é bidirecional.

Para mapear uma associação muitos-para-muitos, usamos as anotações @ManyToMany, @JoinTable e @JoinColumn. Vamos dar uma olhada neles.

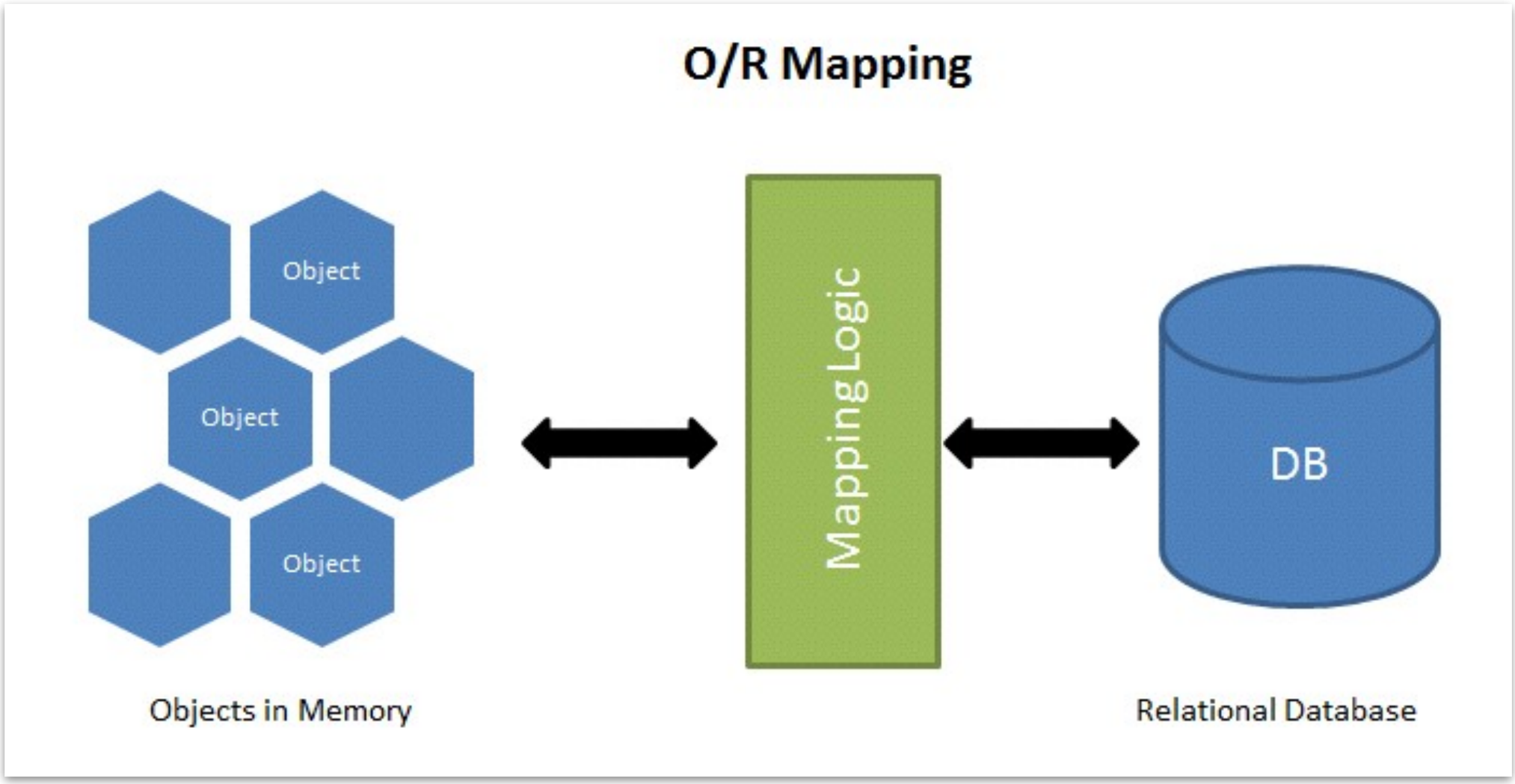
A anotação @ManyToMany é usada nas duas classes para criar o relacionamento muitos-para-muitos entre as entidades.

Esta associação tem dois lados, isto é, o lado proprietário e o lado inverso. Em nosso caso, o lado proprietário é Turma, portanto a tabela de junção é especificada no lado proprietário usando a anotação @JoinTable na classe Turma. A @JoinTable é usada para definir a tabela de junção. Nesse caso, é turmas_disciplinas.

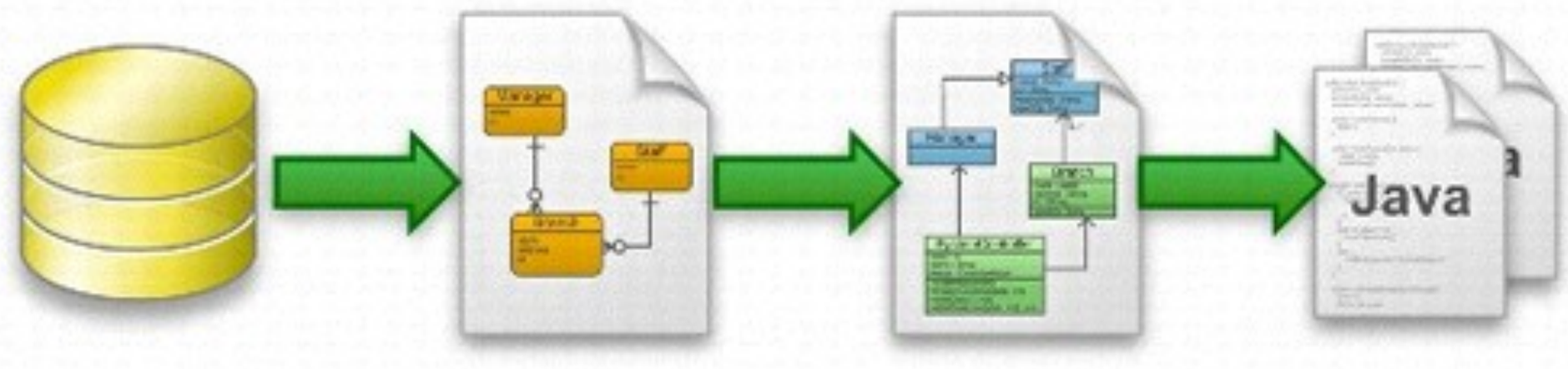
A anotação @JoinColumn é usada para especificar a coluna de junção / vinculação com a tabela principal. Aqui, a coluna de junção é disciplina_id e turma_id é a coluna de junção inversa, pois o Disciplina está no lado inverso do relacionamento.

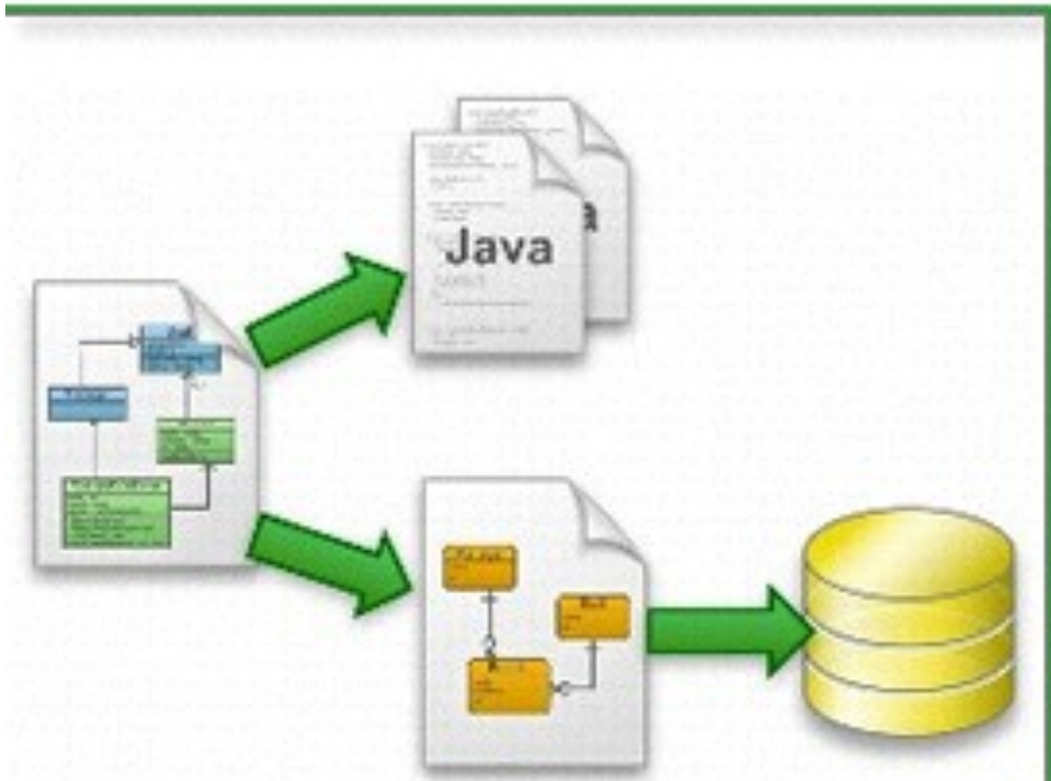
Na classe Disciplina, o atributo mappedBy é usado na anotação @ManyToMany para indicar que a coleção de turmas é mapeada pela coleção de disciplinas (disciplinaCollection) do lado do proprietário.

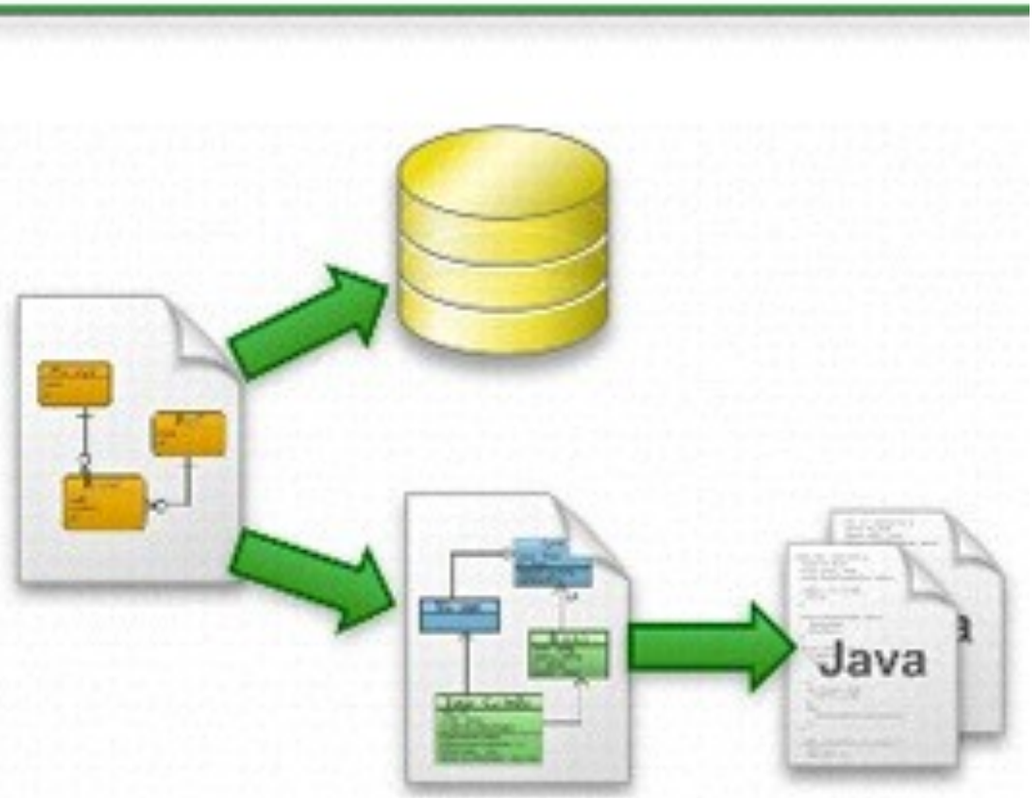
```java
public class FrutaFactory {

    private Connection c;

    public FrutaFactory (Connection c) {
        this.c = c;
    }

    public List<Fruta> getAll () throws SQLException {
        PreparedStatement p = this.c.prepareStatement("SELECT * FROM frutas;");
        ResultSet r = p.executeQuery();
        ArrayList <Fruta>list = new ArrayList <Fruta> ();
        while(r.next())
            list.add(new Fruta(r.getString("nome")));
        return list;
    }
}
```



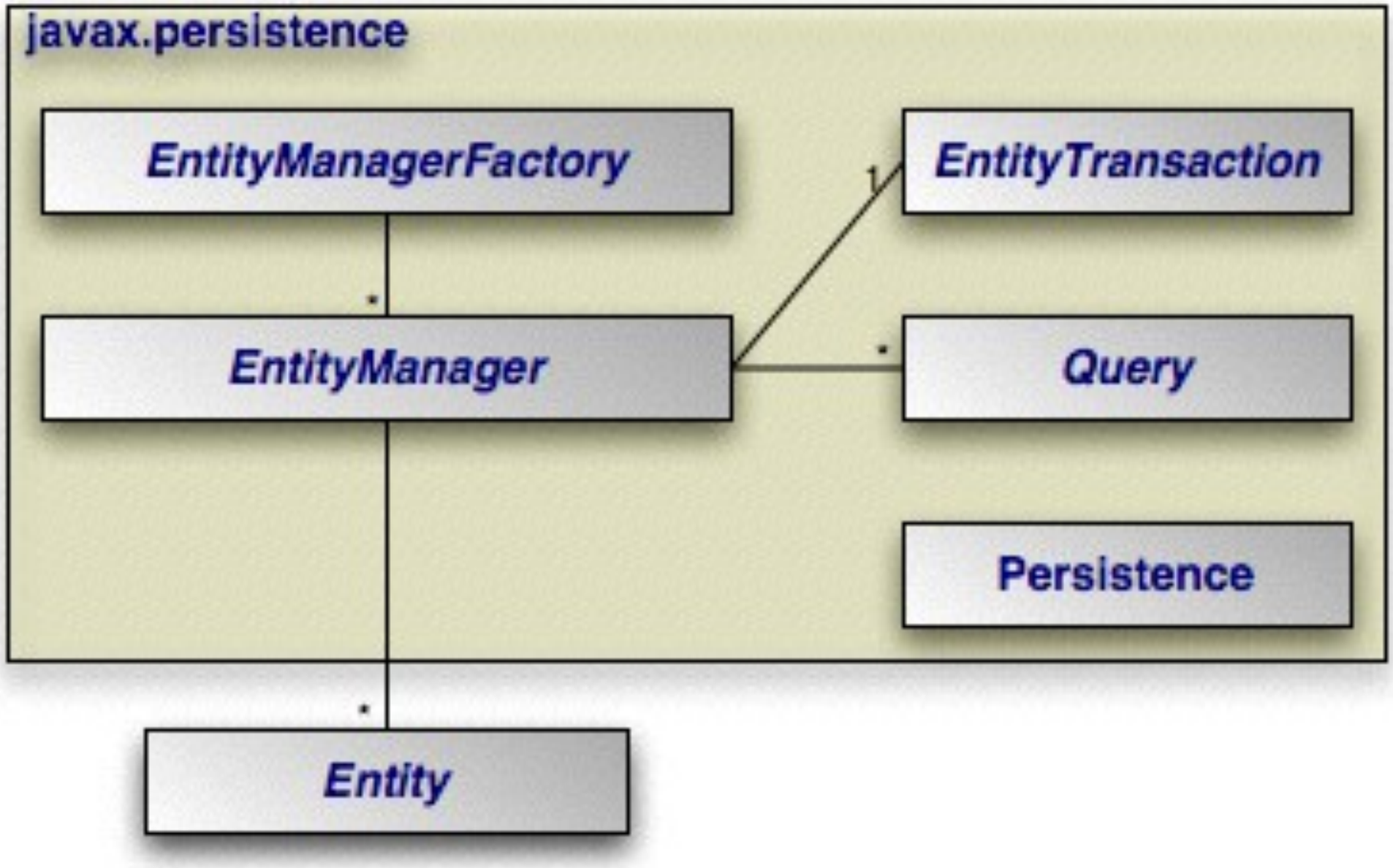
JPA



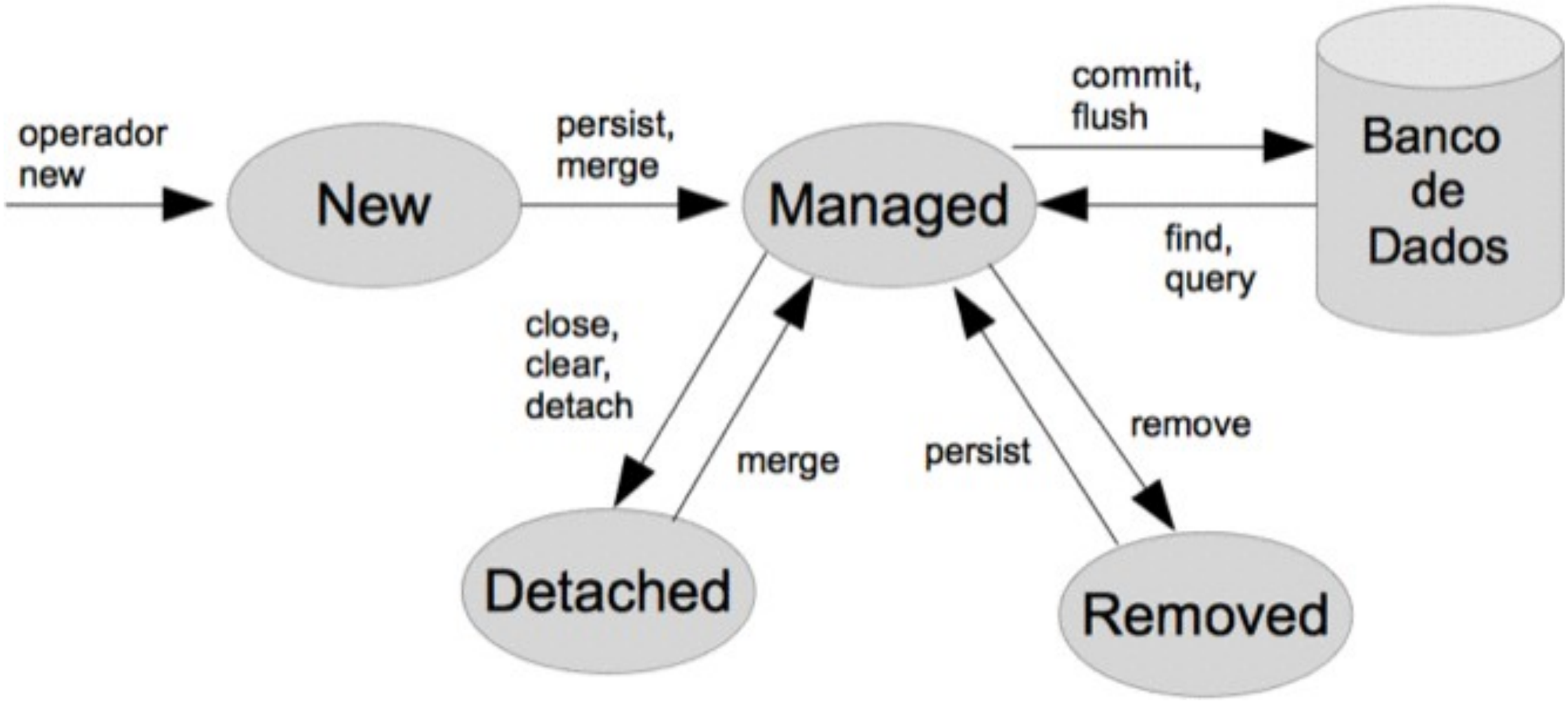




JPA


javax.persistence
EntityManagerFactory
EntityTransaction
1
*
EntityManager
*
Query
Persistence
*
Entity

JPA


operador
new
New
persist,
merge
Managed
commit,
flush
Banco
de
Dados
find,
query
close,
clear,
detach
merge
persist
remove
Detached
Removed

Testes

EclEmma

https://www.youtube.com/watch?v=a_I-87xEBgI