

Módulo II

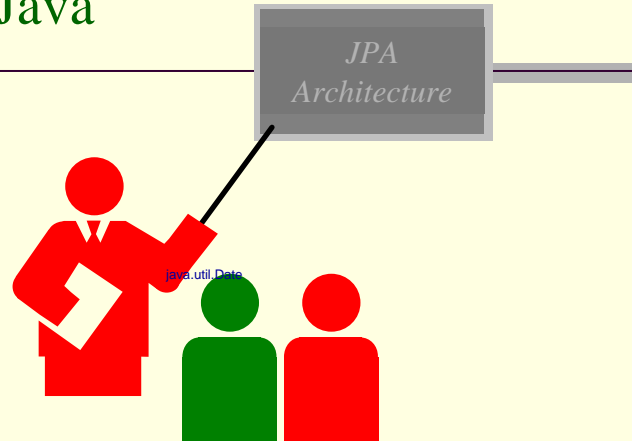
Padrão de Persistência JPA

Prof. Ismael H F Santos

Ementa

- JPA Architecture
- JPA Data Types
- JPA Entity Manager
- JPAQL - JPA Query Language
- JPA Transactions
- JPA ORM (object relational mapping)
- JPA Transitive Persistence
- JPA Hibernate Provider

FPSW-Java

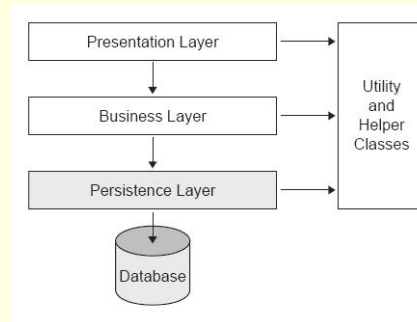


O que é o JPA ?

- **JPA (Java Persistence API)** é a especificação padrão para o gerenciamento de persistência e mapeamento objeto relacional (ORM), surgida na plataforma **Java EE 5.0** (Java Enterprise Edition) na especificação do **EJB 3.0** (Enterprise Java Beans 3.0).
- Introduzida para substituir os **Entity Beans** (que foram descontinuados) e simplificar o desenvolvimento de aplicações **JEE** e **JSE** que utilizem persistência de dados.
- Possui uma completa especificação para realizar mapeamento objeto relacional, utilizando **anotações** da linguagem Java (JSE 5.0 ou superior). Também dá suporte a uma **linguagem de consulta**, semelhante à SQL, permitindo consultas estáticas ou dinâmicas.

O que é o JPA ?

- **JPA**, portanto, é um framework utilizado na camada de persistência para o desenvolvedor ter uma maior produtividade, com impacto principal no modo de controle da persistência dentro de Java.
- O Java Persistence API - **JPA** define uma maneira para mapear Plain Old Java Objects, **POJOs**, para um banco de dados, estes POJOs são chamados de **entidades (Entities)**.



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

5

POJO x JavaBeans

- Um **POJO (Plain Old Java Object)** é a denominação que se dá a uma classe que em geral não deriva de nenhuma interface ou classe, contém atributos privados, um construtor padrão sem argumentos, e métodos públicos get/set para acessar os seus atributos.
- A denominação **POJO** foi criada com a intenção de defender a criação de **designs mais simples** em contraposição aos diferentes frameworks (em especial Enterprise Java Beans, antes de EJB 3.0) que vinham criando soluções cada vez mais complexas dificultando em muito a programação e tornando o reaproveitamento cada vez mais difícil.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

6

POJO x JavaBeans

- O padrão **JavaBeans** foi criado com a intenção de se permitir a criação de objetos de interface reutilizáveis, isto é, componentes (**sw reusable components**) que possam ser manipulados visualmente por IDEs (Eclipse, Netbeans, etc) para a composição de novas aplicações.
- A especificação de um **JavaBeans** estabelece algumas convenções:
 - **construtor padrão sem argumentos**, para permitir a sua instanciação de forma simples pelas aplicações que os utilizem;

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

7

POJO x JavaBeans

- **conjunto de propriedades** implementados através de **atributos privados**, e métodos de acesso públicos **get/set** para acessar os seus atributos. Os métodos de acesso devem seguir a uma convenção padronizada para os seus nomes, **getXxx** e **setXxx** quando os métodos de acesso referentes propriedade **xxxx**;
- deve ser **serializada**, isto é, implementar a interface **Serializable** do pacote **java.io**. Isto permite que as aplicações e frameworks salvem e recuperem o seu estado de forma independente da JVM

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

8

POJO x JavaBeans

■ Declarando um JavaBeans

```
// PersonBean.java
public class PersonBean implements java.io.Serializable{
    private String name;
    private boolean deceased;
    // No-arg constructor (takes no arguments).
    public PersonBean() { }
    // Property "name" (capitalization) readable/writable
    public String getName() { return this.name; }
    public void setName(String name) { this.name = name; }
    // Property "deceased"
    // Different syntax for a boolean field (is vs. get)
    public boolean isDeceased() { return this.deceased; }
    public void setDeceased(boolean deceased) {
        this.deceased = deceased; }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

9

POJO x JavaBeans

■ Usando um JavaBeans

```
// TestPersonBean.java
public class TestPersonBean {
    public static void main(String[] args) {
        PersonBean person = new PersonBean();
        person.setName("Bob");
        person.setDeceased(false);
        System.out.print(person.getName());
        System.out.println(person.isDeceased() ?
            " [deceased]" : " [alive]"); }
}
=> Output: "Bob [alive]"
```

- Como as especificações de JavaBeans são baseadas em convenções e não implicam na criação de interfaces muitas vezes se diz que os JavaBeans são POJOs que seguem um conjunto de convenções específicas.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

10

JPA & ORM

■ Exemplo 1 - JDBC

Listagem 01. JDBCCode.java

```
package jdbc;
import java.sql.*;
public class JDBCCode {
    private static Connection con = null;
    public JDBCCode() { }
    public static Connection open(){
        String user = "root";
        String pass = "123456";
        String url =
            "jdbc:mysql://localhost:3306/BUGDB";
        try{
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url,
                user,pass);
        }catch( Exception e ){
            e.printStackTrace();
        }
        return con;
    }
}
```

```
public static void main( String args[] )
    throws Exception{
    String sql = "SELECT * FROM BUGS";
    con = open();
    try {
        Statement st=con.createStatement();
        ResultSet rs= st.executeQuery(sql);
        while( rs.next() ){
            System.out.println("Titulo: "+
                rs.getString("titulo"));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    } finally{
        con.close();
    }
}
```

```
CREATE TABLE bug (
    id_bug int(11) NOT NULL auto_increment,
    titulo varchar(60) NOT NULL,
    data date default NULL,
    texto text NOT NULL,
    PRIMARY KEY (id_bug)
);
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

11

JPA & ORM

- Em **JPA** uma **Entidade** corresponde a um objeto que pode ser gravado na base de dados a partir de um mecanismo de persistência. A classe que implementa a entidade persistente é um **POJO**, que basicamente contém um **chave primária** (atributo identificador), deve ser uma **classe não final** e usualmente pode ser implementada seguindo as convenções de **JavaBeans** (construtor sem argumentos, com propriedades acessadas através de métodos get/set e serializável)

- A **Java Persistence Specification (JPA)** define mapeamento entre o objeto Java e o banco de dados utilizando **ORM**, de forma que **Entidades** podem ser portadas facilmente entre um fabricante e outro.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

12

JPA & ORM

- **ORM**, mapeamento objeto/relacional é automatizado (e transparente) permitindo a persistência de objetos em aplicações Java para tabelas em um banco de dados relacional. Para descrever o mapeamento utiliza-se metadados que descrevem o mapeamento através de arquivos de configuração XML ou anotações.

Anotações (*Java Annotations*)

- **Anotações** são uma forma especial de **declaração de metadados** que podem ser adicionadas ao código-fonte pelo programador. Provêm informações sobre o comportamento de um programa.
- São aplicáveis à classes, métodos, atributos e outros elementos de um programa, além disso, não tem nenhum efeito sobre a execução do código onde estão inseridas.
- Diferentemente de comentários javadoc, anotações são **reflexivas** no sentido de que elas podem ser embutidas nos arquivos de definição classes (byte-codes) gerados pelo compilador podendo ser “**consultadas**” durante a sua execução pela JVM.

Anotações (*Java Annotations*)

- **Anotações** tem diferentes utilidades, entre elas:
 - Instruções para o compilador – detecção de erros e supressão de warnings;
 - Instruções para uso em tempo de compilação e em tempo de “deployment” – usadas para geração de código, arquivos XML;
 - Instruções para tempo de execução (runtime) – criação de informações que podem ser consultadas durante a execução;

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

15

Anotações (*Java Annotations*)

- **Exemplos:**

```
@Author( name = "Benjamin Franklin", date = "3/27/2003" )
class MyClass() { }                               or
@SuppressWarnings(value = "unchecked")
void myMethod() { }                               or
@SuppressWarnings("unchecked") // caso só exista um elemento "value"
void myMethod() { }                               or
@Override                                       // caso não existam elementos na anotação
void mySuperMethod() { }
```

- As anotações podem possuir nenhum, um ou mais de um elemento em sua definição. Se um tipo de anotação possui elementos, o valor para cada atributo deve ser passo entre parênteses.

```
@interface TesteAnotacao { // definição da anotação
    String nomeUser(); int idade();
}
.....
@TesteAnotacao(nomeUser= "Fulano de tal", idade= 25 ) // uso da anotação
public void metodoTeste{
    System.out.println("o usuario" + nomeUser + "idade: " + idade + ...);
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

16

Anotações (*Java Annotations*)

■ Exemplo anotações para documentação – declaração :

```
import java.lang.annotation.*; // import this to use @Documented
@Documented                    // anotação de documentação
@interface ClassPreamble {
    String author();
    String date();
    int currentRevision() default 1;
    String lastModified() default "N/A";
    String lastModifiedBy() default "N/A";
    String[] reviewers(); // Note use of array
}
```

■ uso da anotação no código fonte:

```
@ClassPreamble {
    author = "John Doe", date = "3/17/2002",
    currentRevision = 6, lastModified = "4/12/2004",
    lastModifiedBy = "Jane Doe",
    reviewers = {"Alice", "Bob", "Cindy"} // Note array notation
}
public class Generation3List extends Generation2List { //class code... }
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

17

Anotações (*Java Annotations*)

■ A anotação anterior irá gerar a seguinte saída do javadoc:

```
public class Generation3List extends Generation2List {
    // Author: John Doe
    // Date: 3/17/2002
    // Current revision: 6
    // Last modified: 4/12/2004
    // By: Jane Doe
    // Reviewers: Alice, Bill, Cindy
    // class code goes here
}
```

■ Exemplo anotações para compilação (J2SE built-in)

- A anotação **@Override** informa ao compilador que o elemento anotado tem o objetivo de sobrescrever o elemento definido na superclasse.

```
@Target(ElementType.METHOD)
public @interface Overrides {}
```

- Usando a anotação

```
class A extends B {
    @Overrides
    void myMethod() { } // marca o método como um método que esta
    ..... // sobrescrevendo um método da superclasse
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

18

Anotações (*Java Annotations*)

■ Exemplo anotações para compilação (cont.)

- Embora esta anotação não seja necessária ela ajuda ao compilador informar ao programador caso exista alguma inconsistência entre a declaração do método na classe derivada e a da superclasse

■ Outros exemplos de anotações (J2SE built-in)

- **@Documented** informa que a anotação será utilizada pelo javadoc ou tools similares.

```
@Documented
@Target(ElementType.ANNOTATION_TYPE)
@public @interface Documented { }
```
- **@Deprecated** informa ao compilador para avisar os usuários que se utilizem da classe, método ou atributo anotado que o uso do objeto não é mais recomendado.

```
@Documented
@Retention(RetentionPolicy.SOURCE)
@public @interface Deprecated { }
```
- **@Inherited** informa que a anotação será herdada por todas as subclasses da classe anotada herdarão a mesma anotação automaticamente.

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
@public @interface Inherited { }
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

19

Anotações (*Java Annotations*)

■ Outros exemplos de anotações (J2SE built-in)

- **@Retention** informa a vida útil da anotação. Podendo ser: SOURCE, CLASS ou RUNTIME. O default é SOURCE.

```
@Documented
@Retention(RetentionPolcy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
@public @interface Retention { RetentionPolicy value(); }
```
 - **@Target** usada para informar o tipo do elemento sobre o qual a anotacao pode ser associada (classe, método, ou campo). Quando não estiver presente significa que a anotacao pode ser aplicada a qualquer elemento do programa.

```
@Documented
@Retention(RetentionPolcy.RUNTIME)
@Target(ElementType.ANNOTATION_TYPE)
@public @interface Target { ElementType[] value(); }
```
- O JSE 5.0 inclui a ferramenta **APT (annotation processing tool)** que pode ser usada para ler um programa Java e tomar ações baseadas nas anotações declaradas.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

20

Anotações (Java Annotations)

- Para disponibilizar uma anotação que pode ser aplicada a classes e métodos para implementar uma política de verificação de segurança em tempo de execução (**runtime**) devemos declará-la da seguinte forma:

```
import java.lang.annotation.*; // import this to use @Retention
@Documented
@Target({METHOD, TYPE})
@Retention(RetentionPolicy.RUNTIME) // available in RUNTIME
public @interface SecurityPermission {
    // Elements that give information for runtime processing
    String[] value();
}
```

- Usando a anotação

```
@SecurityPermission("ALL") // marca a classe sem restrição de acesso
public class AnnotationTest {
    public AnnotationTest() { SecurityChecker.checkPermission(); ... }
    @SecurityPermission("None") // marca o método com restrição de acesso
    void Method1() { SecurityChecker.checkPermission(); ... }
    @SecurityPermission("ALL") // marca o método sem restrição de acesso
    void Method2() { SecurityChecker.checkPermission(); ... }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

21

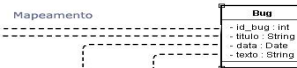
JPA & ORM

Exemplo 2 - JPA

```
Listagem 01. Bug.java
package exemploJPA;
import javax.persistence.*;
@Entity
@Table(name="BUGS")
public class Bug {
    private Integer idBug;
    private String titulo;
    private java.util.Date data;
    private String texto;
    public Bug() {}
    @Id
    @GeneratedValue(strategy=
        GenerationType.SEQUENCE)
    // informa que o id será gerado pelo DB.
    @Column(name="idBug")
    public Integer getIdBug() { return idBug; }
    public void setIdBug(Integer iBug) {
        this.idBug = iBug;
    }
    @Column(name="titulo")
    public String getTitulo() { return titulo; }
    public void setTitulo(String titulo){
        this.titulo = titulo;
    }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br



id_bug	titulo	data	texto
3	Erro Bug	2007-01-13	Quando o context JSF lança este aviso...
5	Value Object	2007-02-15	Quando utilizar VO, colocar também ...
6	heheheh	2007-02-16	kwjkwjkwj
7	Muito show!!!	2007-02-16	heheheheheh 10

```
@Temporal(TemporalType.DATE)
@Column(name="data")
public Date getData(){return data; }
public void setData(Date data) {
    this.data = data; }
@Column(name="texto")
public String getTexto(){ return
texto; }
public void setTexto(String texto) {
    this.texto = texto; }
@Override
public String toString(){
    return "ID: "+this.id_bug; }
```

Listagem 02. Recuperar objeto.

```
public Object findByPk( int pKey ) {
    EntityManager em = getEntityManager();
    return em.find(Bug.class, pKey);
}
```

22

JPA Annotations

- JPA define as anotações: **@Entity**, que torna uma classe persistente. A anotação **@Table** informa o nome da tabela, através do atributo `name`, para a qual a classe será mapeada. Quando o nome da tabela é igual ao nome da classe, está anotação não precisa ser informada, basta a anotação **@Entity**.

```
@Entity
@Table(name = "BUGS")
public class Bug implements Serializable { ... }
```

- A anotação **@Id** define o atributo a que ela se refere como o identificador único para os objetos desta classe, ie, a chave primária da tabela. Já a anotação **@GeneratedValue** faz com que o framework de persistência gere valores para a chave primária na tabela de forma automática e com valores únicos. Estratégias: **AUTO**, **SEQUENCE**, **IDENTITY** e **TABLE**.

```
public class Bug implements Serializable {
    @Id @GeneratedValue(strategy= GenerationType.SEQUENCE)
    private Long idBug; ...
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

23

JPA Annotations

- Mapeamento de atributos primitivos não são necessários. Opcionalmente todo atributo pode ser anotado com **@Column** que tem os seguintes elementos: `name`, `length`, `nullable`, `unique`, etc.

```
public class Bug implements Serializable {
    @Column (name="comNome", length=30, nullable=false,
            unique=true)
    private String nome;
    ...
}
```

- Em atributos para datas (`Date` ou `Calendar`), deve ser usada a anotação **@Temporal** para definir o tipo de informação desejada, podendo ser um dos tipos **DATE**, **TIME**, **TIMESTAMP**

```
public class Bug implements Serializable {
    @Temporal(TemporalType.DATE)
    private Date data;
    ...
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

24

JPA Annotations

- Os atributos da classe marcados com **@Transient** não serão salvos no BD e os que não possuem anotações associadas são mapeados com anotações padrões que mapeiam os atributos para colunas na tabela com tipos padrões, por exemplo, o atributo nome do tipo String é mapeado para uma coluna nome da tabela do tipo VARCHAR.
- Este tipo de comportamento caracteriza o que chamamos de **Configuração por Exceção (Configuration by Exception)**, isto permite ao mecanismo de persistência aplicar defaults que se aplicam a maioria das aplicações fazendo que o usuário tenha que informar apenas os casos em que o valor default não se aplica.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

25

JPA Annotations

- Como vimos, **anotações** em JPA seguem o **pattern Decorator** e podem ser divididas em 2 tipos básicos:
 - Physical annotations - **@Table**
 - Logical annotations - **@ManyToOne**, etc
- **Configuração por decoração** permite:
 - Refinar o modelo físico do BD
 - **@Table**, **@SecondaryTable**, **@Column**
 - Definir o mapeamento lógico objeto-relacional
 - **@Version**, **@Transient**, **@Embeddable**
 - Definir a estratégia de geração dos identificadores
 - **@GeneratedValue**
 - **@SequenceGenerator** / **@TableGenerator**

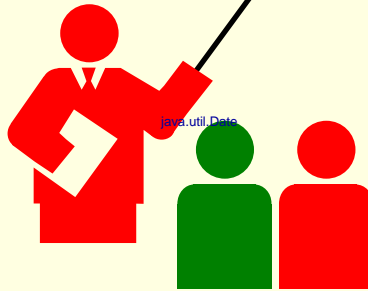
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

26

FPSW-Java

JPA
Data Types



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

27

Mapeamento de Tipos

■ Mapeamento tipos JAVA x SQL

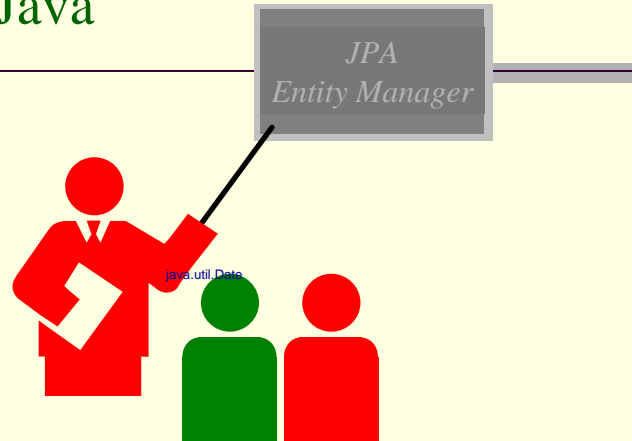
- **big_decimal** - `java.math.BigDecimal` para o tipo de dados SQL NUMERIC ou equivalente.
- **locale, timezone, currency** - `java.util.Locale`, `java.util.TimeZone` e `java.util.Currency` para o tipo de dados SQL VARCHAR.
- **class** - `java.lang.Class` para o tipo de dados SQL VARCHAR. A Classe é mapeada utilizando o seu nome completo.
- **binary** - byte arrays para o tipo binário SQL apropriado.
- **text** - Strings Java strings para o tipo de dados SQL CLOB ou TEXT.
- **serializable** - Classes javas serializáveis para o tipo binário SQL apropriado.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

28

FPSW-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

29

Gerenciador de Entidades – Entity Manager

- Para utilizar o **JPA**, é necessária a escolha de um provedor uma vez que **JPA** é uma especificação para *frameworks* de persistência. Por padrão, a implementação de referência é o *Oracle Toplink Essentials*. Também existem outros provedores JPA no mercado, como o *Hibernate Entity Manager*, *Bea Kodo* e o *ApacheJPA*.
- Com a **API JPA**, a persistência de um objeto é feita invocando o gerenciador de persistência, também conhecido como gerenciador de entidades (*EntityManager*), responsável por controlar o ciclo de vida das instâncias de entidades.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

30

Ciclo de Vida de um Objeto Persistente

- Outro conceito também relacionado a JPA é o de contexto persistente (*PersistenceContext*), que é o espaço de memória onde os objetos persistentes se encontram. Quando se interage com o mecanismo de persistência, é necessário para a aplicação ter conhecimento sobre os estados do **ciclo de vida da persistência**.
- Em aplicações OO, a persistência permite que um objeto continue a existir mesmo após a destruição do processo que o criou. Na verdade, o que continua a existir é seu estado, já que pode ser armazenado em disco e assim, no futuro, ser recriado em um novo objeto. O **ciclo de vida** de uma entidade é descrito por quatro estados a saber: *new*, *managed*, *removed* e *detached*.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

31

Ciclo de Vida de um Objeto Persistente

- *new* estado de todo objeto criado com o operador **new** e que por isso ainda não possui um valor para o seu identificador (primary key) persistente. Objetos neste estado também são conhecidos como objetos **transientes**. Em outras palavras, o objeto ainda não está associado ao *EntityManager*.
- *managed (persistent)* é aquela entidade que tem um valor definido para o seu identificador persistente (primary key) e está associado a um **contexto de persistência**. Assim, o *EntityManager* irá fazer com que a base de dados seja atualizada com novos valores dos atributos persistentes ao fim de cada transação.

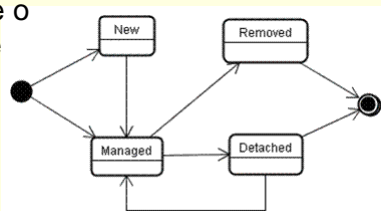
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

32

Ciclo de Vida de um Objeto Persistente

- **removed** são os objetos associados a um contexto de persistência, porém que estão “marcados” para serem excluídos do BD ao final da transação .
- **detached** são os objetos que possuem uma representação correspondente na base de dados, mas no momento não estão associados a um contexto de persistência, ie, a um **EntityManager**. Seja porque o **EM** foi fechado ou porque a entidade foi removida explicitamente do **EM**



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

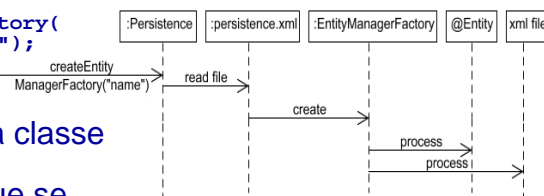
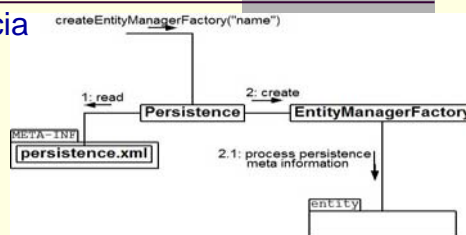
33

EntityManager (EM)

- Para se fazer a persistência em JPA é preciso obter uma instância de um **EntityManager**. Isto é feito utilizando-se a fábrica (factory method) **EntityManager Factory**.

```
EntityManagerFactory fabrica =
Persistence.
createEntityManagerFactory(
"nameOfPersistenceUnit");
```

- Para obtenção da fábrica solicitamos a classe **Persistence** de **javax.persistence** que se encarrega de todo o trabalho.



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

34

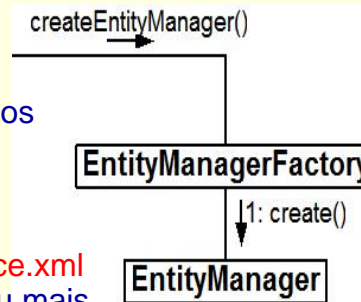
EntityManager (EM)

- Criação do EntityManager:

```
EntityManager em =  
fabrica.createEntityManager();
```

- A partir desse momento podemos executar as operações de persistência.

- O arquivo **META-INF/persistence.xml** é obrigatório e descreve uma ou mais unidades de persistência. Definimos o arquivo JAR que contem o arquivo persistence.xml de um **Persistence Archive**



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

35

EntityManager (EM)

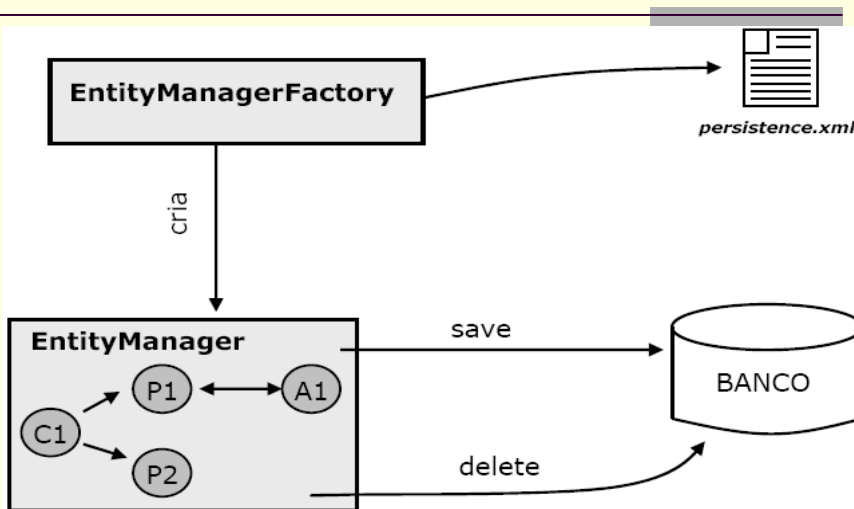
- Uma **unidade de persistência** possui um nome e contém informações para conexão com o BD (diretamente através de um driver **JDBC** em **JSE** ou indiretamente através de uma referência a um data-source registrado no **JNDI** em **JEE**)
- As Meta Informações de Persistência (**Persistence Meta Information**) descrevem a configuração das entidades do BD e das associações entre as Entidades associadas a unidade de persistência. A informação pode ser fornecida através de anotações ou através de arquivos de configuração XML (**META-INF/persistence.xml**).

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

36

Criação de um EntityManager (EM)



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

37

EntityManagerFactory (EMF)

- Implementação `javax.persistence.EntityManagerFactory`
 - Threadsafe
 - Fábrica de EntityManagers
 - É um processo custoso e demorado por isso geralmente se usa um por aplicação
 - Exceto quando se precisa acessar mais de uma fonte de dados diferente dentro da mesma aplicação.
 - Quando o método `Persistence.createEntityManagerFactory("AlunoJPA")` é executado a implementação de JPA invoca o `Class-loader.getResource(META-INF/persistence.xml)`, que obtém as informações sobre todos os `Persistence Providers` definidas no arquivo de definição da persistência e pergunta qual deles disponibiliza a persistence unit `AlunoJPA`, se nenhum for encontrado então ocorre `PersistenceException`

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

38

EntityManager (EM)

- Implementação `javax.persistence.EntityManager`
 - Realiza a troca de informações entre a aplicação e um esquema de armazenamento de dados
 - Começa e finaliza transações
 - Wrapper para conexões JDBC
 - Faz o cache de objetos persistentes
 - Interface semelhante ao objeto `session` do Hibernate
 - Anti-pattern: *EntityManager-per-operation*
 - Uso de auto-commit após execução de cada comando SQL é desaconselhável, so para ad-hoc console interaction
 - Em ambientes cliente/servidor o mais comum é usar o padrão *EntityManager-per-request*, usando uma única transação com o BD para tratar cada requisição do cliente. Este é o modo default em aplicações JEE

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

39

Persistence Context e Persistence Unit

- **Persistence Context (PC)** é o conjunto de instâncias de Entidades no qual cada para cada Entidade persistente no BD existe uma única instancia de Entidade (objeto) em memória, ie, no contexto persistente. **Database Identity == JVM Identity**. O escopo do contexto pode ser:
 - Associado as fronteiras de uma Transação
 - Disponível entre diferentes transações, constituindo uma Unidade de Trabalho estendida (Extended unit of work)
- **OBS: PC não são compartilhados entre diferentes EM !**
- **Persistence Unit (PU)** é o conjunto de tipos de Entidades que são gerenciados por um dado **EntityManager (EM)**. Define o conjunto de todas as classes que são relacionadas ou agrupadas pela aplicação para serem persistidas num mesmo BD.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

40

Container-Managed x Application-Managed Entity Manager

- O ciclo de vida de um EM pode ser gerenciado pelo container JEE (Container-Managed-EM, CM-EM) ou pela aplicação JSE (Application-Managed-EM, AM-EM).
- No CM-EM o container é responsável por abrir e fechar o EM (de forma transparente) e também pela gerência da transação usando a interface JTA (Java Transaction API).
- O PC (PersistentContext) é propagado automaticamente pelo container para todos os componentes que utilizem o EM. No esquema CM-EM a aplicação não interage com o EMF (EntityManagerFactory).
- O EM é obtido pela aplicação através do uso de injeção de dependência (dependency injection) usando a anotação @PersistenceContext(unitName="persistentUnit") ou através de uma consulta ao JNDI.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

41

Container-Managed x Application-Managed Entity Manager

- No AM-EM a aplicação utiliza o EMF para controlar o ciclo de vida do EM e o PC (PersistentContext) que não é propagado automaticamente para os componentes da aplicação. Ao término da aplicação o EMF deve ser fechado.
- A gerência da transação é feita utilizando-se por uma implementação da JTA API ou pela EntityTransaction (resource-local entity manager) disponibilizada pelo EM que mapeia as transações diretamente para transações JDBC.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

42

Persistence Context Scope

- Quando o escopo do **Persistence Context (PC)** estiver associado a uma transação **JTA** (caso mais comum em **JEE**) o ciclo de vida do **PC** também estará associado ao da transação, ie, quando o **EM** é invocado o **PC** também é aberto ou um novo será criado caso ainda não exista.
- Um **Extended Persistence Context (EPC)** deve ser utilizado quando se deseja lidar com múltiplos ciclos de interação com o usuário caracterizando o que chamamos de uma **Unidade de Trabalho** estendida.
- Para o caso **AM-EM (JSE)** somente são suportados **EPC** e **Resource-local EM**. Um **EPC** é criado quando o **EM** é criado e mantido até que o **EM** seja fechado. Logo todas as operações de modificação (**persist**, **merge**, **remove**) executadas fora de uma transação são armazenadas até que o **EPC** seja associado a uma transação o que em geral acontecerá pelo menos ao término da aplicação.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

43

JPA – exemplo 3

Exemplo 3 - JPA

Listagem 01. Aluno.java

```
package exemplo2JPA;
import javax.persistence.*;

@Entity
@Table(name="aluno")
public class Aluno {
    @Id @GeneratedValue
    private int id;
    private String nome;
    private String matr;

    public Aluno() {}
    public int getId() { return id; }
    public void setId(int id) {this.id = id;}
    public String getNome() { return nome; }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getMatr() { return matr; }
    public void setMatr (String matr) {
        this.matr = matr;
    }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

44

Listagem 02. Principal.java

```
package exemplo2JPA;
import javax.persistence.*;

public class Principal {
    public static void main(String[] args){
        //Cria objeto que gerenciará persistência
        EntityManagerFactory fab = Persistence.
        createEntityManagerFactory("AlunoJPA");
        EntityManager em = fab.createEntityManager();
        try {
            EntityTransaction tx = em.getTransaction();
            tx.begin(); //Inicia transação
            Aluno a1 = new Aluno();
            a1.setNome("Pedro");a1.setMatr("09387");
            em.persist(a1); //Persiste primeiro aluno
            Aluno a2 = new Aluno();
            a2.setNome("Maria");a2.setMatr("02347");
            em.persist(a2); //Persiste segundo aluno
            tx.commit(); //Finaliza transação
        } catch (Exception ex) {
            tx.rollback(); //Rollback em caso de erro
        } finally { em.close(); fab.close(); }
    }
}
```

JPA – exemplo 3 (cont.)

- As configurações de persistência são feitas a em um arquivo XML definindo a unidade de persistência com as seguintes infos:
 - as classes a serem persistidas;
 - informações para conexão com a base de dados
- Observe que na tag **name** é informado o nome da unidade de trabalho "AlunoJPA", usada no momento da criação da fábrica de gerenciadores de persistência, através da chamada **Persistence.createEntityManagerFactory**

Listagem 03. exemplo2JPA.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
version="1.0">
  <persistence-unit name="AlunoJPA"
                    transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>exemplo2JPA.Aluno</class>
    <properties>
      <property name="hibernate.show_sql" value="false">
    <property name="hibernate.connection.driver_class"
              value="org.hsqldb.jdbcDriver"/>
    <property name="hibernate.connection.username" value="sa"/>
    <property name="hibernate.connection.password" value=""/>
    <property name="hibernate.connection.url"
              value="jdbc:hsqldb:hsqldb://localhost"/>
    <property name="hibernate.dialect"
              value="org.hibernate.dialect.HSQLDialect"/>
    <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    <property name="hibernate.max_fetch_depth" value="3"/>
    </properties>
  </persistence-unit>
</persistence>
```

• Abril 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

45

JPA – obtendo um EM

- Um **EMF** deve ser considerado como um objeto imutável que aponta para uma única BD e mapeia um conjunto de entidades definidas

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("AlunoJPA");
EntityManager em = emf.createEntityManager();
```

.....

- Um EMF é tipicamente criado na inicialização da aplicação e fechado quando a aplicação termina.
- Para aplicações **JSE**, a criação de um **EM** mantém o mesmo contexto de persistência (**PC**) durante o ciclo de vida do **EM**, ie, todas as entidades permanecem no estado **managed** entre diferentes transações a menos que seja executado algum dos método **em.clear()** ou **em.close()**.
- Operações de adição, atualização e remoção de entidades devem ser executadas dentro de transações. **(ver adiante !)**

Abril 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

46

JPA – persist

■ Persistindo um objeto

```
public void persist(Object entity);

EntityManager em;
.....
Employee e = new Employee(10);
e.setName("Miller");
em.persist(e); //coloca no estado de managed para o EM
```

- Novas instâncias são consideradas transientes;
- Usa-se o comando **persist** para tornar um objeto persistente, ou ele pode se tornar persistente transitivamente em decorrente de **operações de cascade**;
- Ao persistir um objeto o seu id é gerado;
- Se o objeto tiver associação com outros objetos é preciso salva-los antes (ou usar persistência transitiva)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

47

JPA – persist e find

■ Procurando um objeto pela PK

```
public <T> T find(Class<T> entityClass, Object primaryKey);
EntityManager em;
.....
Employee e = em.find(Employee.class,Integer.valueOf(10));
if ( e != null ) .... // e is now a managed class !!!
```

■ Obtendo um proxy para um objeto pela PK

```
public <T> T getReference(Class<T> entityClass, Object primKey);
EntityManager em;
.....
Employee e = new Employee(10);
e.setName("Miller");
Parent p = em.getReference(Parent.class, parentID);
e.setParent(p);
em.persist(e);
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

48

JPA – find e remove

- Removendo um objeto

```
public void remove(Object entity);
```

```
EntityManager em;  
.....  
Employee e = em.find(Employee.class, Integer.valueOf(10));  
if( e != null ) {  
    em.remove(e); // torna o objeto e managed e depois marca-o  
                  // para ser removido pelo EM  
}
```

- Cuidado para não resultar em violação de integridade

- Exemplo: ao remover um ator, seu cadastro deve também ser removido

```
Ator ator = em.find(Ator.class, new Long(1));  
em.remove(ator.getCadastro()); // caso não exista cascade=  
em.remove(ator);               // REMOVE ou cascade=ALL
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

49

JPA – flush

- Força a atualização dos objetos do EM no BD

```
public void flush();
```

```
EntityManager em;  
.....  
public Employee raiseEmpSalary(int id, float raise) {  
    Employee e = em.find(Employee.class, id);  
    if( e != null ) {  
        e.setSalary(e.getSalary()+ raise);  
    }  
    em.flush(); // equivale em.getTransaction().commit();  
    return e;  
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

50

JPA – persist, flush e refresh

- Refresh serve para atualizar uma entidade e suas coleções. Útil quando triggers do BD são acionados para inicialização de algumas propriedades de uma entidade.

```
public void refresh(Object entity);
```

```
EntityManager em;  
.....  
Employee e = new Employee(10);  
....  
em.persist(e);  
em.flush(); // força execução de SQL insert e  
            // execução dos triggers associados no BD  
em.refresh(e); // atualiza estado do objeto após  
            // atualizacao no BD
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

51

JPA – flush e merge

- Para evitar a execução de um SQL SELECT (**find**) e um SQL UPDATE (**flush**) para atualizar um objeto pode-se usar o método **merge**

```
public void merge(Object entity);
```

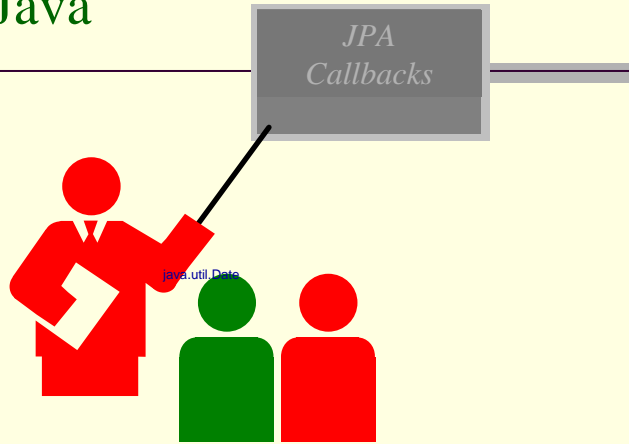
```
em1 = emf.createEntityManager();  
    Employee e = new Employee(10); ....  
    em1.persist(e);  
em1.close();  
....  
e.setSalary(10000);  
....  
em2 = emf.createEntityManager();  
    em2.merge(e); // Sincroniza estado de e no novo EM  
em2.close();
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

52

FPSW-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

53

JPA Callbacks

- JPA disponibiliza listeners para execução de callbacks como `@PostLoad`, `@PrePersist`, `@PostPersist`, `@PreUpdate`, `@PostUpdate`, `@PreRemove` e `@PostRemove`

```
@EntityListeners({CustListener.class})
@Entity(name = "CUSTOMER") //Name of the entity
public class Customer implements Serializable{
    ...
}
public class CustListener {
    @PreUpdate
    public void preUpdate(Customer cust) {
        System.out.println("In pre update"); }

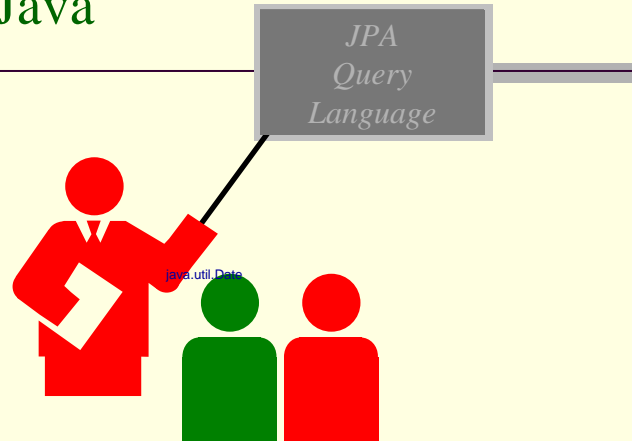
    @PostUpdate
    public void postUpdate(Customer cust) {
        System.out.println("In post update"); }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

54

FPSW-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

55

JPA Queries – JPAQL

- JPA disponibiliza uma query API semelhante a uma query SQL chamada **Java Persistence Query Language (JPQL)**. O objeto Query pode ser obtido a partir do **EntityManager** usando o factory method **createQuery**.
- Uma query pode ser **estática** ou **dinâmica**. Uma query **estática** pode ser definida através de uma **anotação** ou dentro de um **arquivo XML** pela criação de uma query nomeada (**named query**).

```
@NamedQuery( name= "findCourses",
    query="select c FROM Course c WHERE c.name LIKE :courseName")
.....
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("Tutorial");
EntityManager em = emf.createEntityManager();
.....
Query query = em.createNamedQuery("findCourses");
courses = query.setParameter("courseName",
    "Java Programming").getResultList();
.....
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

56

JPA Queries – JPAQL

- Uma query **dinâmica** é criada durante a execução e por isso será um pouco mais custosa porque o **JPA provider** não poderá fazer nenhum tipo de otimização a priori.

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("Tutorial");
EntityManager em = emf.createEntityManager();
.....
@SuppressWarnings("unchecked")
public Collection<Employee> findAllEmployees() {
    Query query = em.createQuery("select e FROM Employee e");
    return query.getResultList(); }
.....
// find all employees
Collection<Employee> listEmps = service.findAllEmployees();
for( Employee e : listEmps )
    System.out.println("Found employee: " + e);
```

- Observe que a query JPQL não se refere a uma tabela **EMPLOYEE** do BD e sim a entidade **Employee**. Portanto esta é uma query selecionando todos os objetos **Employee** sem nenhuma filtragem !

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

57

JPA Queries – JPAQL

- Suporta parâmetros **posicionais** (1,2...) e **nomeados**

```
@SuppressWarnings("unchecked")
List<Comunidade> comunidades =
    em.createQuery("from Comunidade").getResultList();

List<Comunidade> comunidades =
    em.createQuery("from Comunidade c where c.nome = ?")
    .setParameter(1, "Java").getResultList();

List<Comunidade> comunidades =
    em.createQuery("from Comunidade c where c.nome =
    :nome").setParameter("nome", "Java").getResultList();

List<Comunidade> comunidades =
    em.createQuery("from Comunidade c where c.nome = ?1
    and c.descricao like ?2").setParameter(1, "Java")
    .setParameter(2, "%Programadores%").getResultList();
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

58

JPA Queries – JPAQL

- lista de parâmetros nomeados (cont.)

```
List names = new ArrayList();
names.add("JAVA"); names.add("SQL");
em.createQuery("select c from Comunidade c where c.nome
in (:nlist)").setParameter("nlist", names).
getResultList();
```

- getResultList() carrega as instâncias resultado para o Persistente Context da aplicação. Já getSingleResult() pode ser usado quando se espera um valor como retorno.

```
@SuppressWarnings("unchecked")
List<Comunidade> comunidades =
em.createQuery("from Comunidade c where c.nome =
:nome").setParameter("nome", "Java").getSingleResult();
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

59

JPA Queries – JPAQL

- Paginação

- Usado para determinar fronteiras no resultado
- Útil para evitar desperdício de memória.

```
List<Comunidade> comunidades =
em.createQuery("select c from Comunidade c")
.setFirstResult(5).setMaxResults(50)
.getResultList();
```

- JPAQL oferece a possibilidade de execução de projeções suportando left outer join e inner join

```
Iterator it = em.createQuery("select dep, emp from Dependente
dep join dep.empregado emp").getResultList().iterator();
while( it.hasNext() ) {
Object[] tuple = (Object[])it.next();
Dependente dep = tuple[0];
Empregado emp = tuple[1];
}
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

60

JPA Queries – JPAQL

- **fetch join** permite que associações e coleções sejam inicializadas junto com os objetos pai dentro de um único select

```
Iterator it = em.createQuery("select dep, emp from
    Dependente dep join dep.empregado
    emp").getResultList().iterator();
```

- Suporta subquery, order by, group by, having
- Funções de Agregação: avg(...), sum(...), min(...), max(...), count(*), count(...), count(distinct ...)

JPA Queries – JPAQL

■ Exemplo 4 – JPA Query

```
EntityManagerFactory f = Persistence.
    createEntityManagerFactory("Tutorial");
EntityManager e = f.createEntityManager();
.....
Transaction tx = em.getTransaction();
try {
    tx.begin();
    Employee emp = new Employee();
    emp.setName("Miller");
    emp.setSalary(10000);
    e.persist(product);
    tx.commit();
} finally {
    if (tx.isActive()) {
        tx.rollback();
    }
    em.close();
}
.....
tx = em.getTransaction();
try {
```

```
    tx.begin();
    Query q = pm.createQuery("SELECT e FROM
        Employee e WHERE e.salary >= 1000.00");
    List results = q.getResultList();
    Iterator iter = results.iterator();
    while( iter.hasNext() ) {
        Employee emp = (Employee)iter.next();
        ... (use the retrieved object)
    }
    tx.commit();
} finally {
    if (tx.isActive()) {
        tx.rollback();
    }
    em.close();
}

// Find and delete all objects whose last name
// is 'Jones'
Query q = e.createQuery("DELETE FROM Person p
    WHERE p.lastName = 'Jones'");
int numberInstancesDeleted= q.executeUpdate();
```

JPA Queries – Externalizando Queries

- Queries declaradas nas classe via **annotation** para não misturar código de consulta com o da aplicação

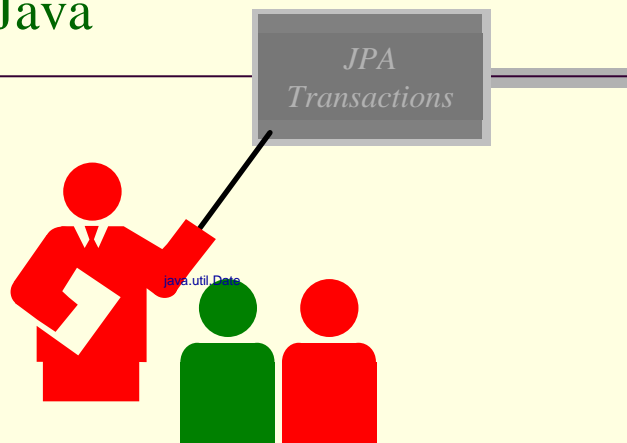
```
@Entity
@Table (name = "COMUNIDADES")
@NamedQueries({
    @NamedQuery(name="Comunidade.porCriador"),
    query="from Comunidade c where c.criador = ?"
})
public class Comunidade implements Serializable {
    ...
}
Ator ator = em.find(Ator.class, new Long(1));
@SuppressWarnings("unchecked")
List<Comunidade> comunidades =
    em.createNamedQuery("Comunidade.porCriador")
        .setParameter(1, ator).getResultList();
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

63

FPSW-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

64

JPA – funcionamento flush

■ Em uma transação (Persistence Context)

- De tempos em tempos o **EM** irá executar comandos DML SQL para sincronizar as entidades armazenados em memória com as do BD. Este processo ocorrerá por default (segundo Hibernate Entity Manager), quando:
 - Antes da execução de uma query
 - Após a execução de **EntityTransaction.commit()**
- Se a transação estiver ativa os comandos são executados na seguinte ordem
 - Todas as inserções de entidades, na mesma ordem em que foram persistidas (**em.persist(entity)**).
 - Todas as atualizações de entidades
 - Todas as remoções de coleções de entidades
 - Todas as remoções, atualizações e inserções de elementos de coleções de entidades
 - Todas as inserções de coleções de entidades
 - Todas as remoções de entidades

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

65

JPA Transactions

■ begin – commit/rollback

```
.....
EntityManager em = emf.createEntityManager();
EntityTransaction tx = null;
try {
    tx = em.getTransaction();
    tx.begin();
    em.setFlushMode(FlushMode.COMMIT); // Only flush at commit time
                                        // flush automatically -> default
    Employee e = em.find(Employee.class, id=158); //Usuario recuperado !!!
    e.setSalary(45000);
    em.createQuery("from Empregado as emp left outer join emp.deps dep")
        .getResultList(); //Pode retornar dados desatualizados
    tx.commit(); //Executa o flush
} catch( RuntimeException ex ) { //Excecoes do em não são
    if( tx != null && tx.isActive() ) //recuperaveis
        tx.rollback(); //Desfaz operações em caso de erro
    throw ex;
} finally {
    em.close(); //Libera recursos
}
```

April 05

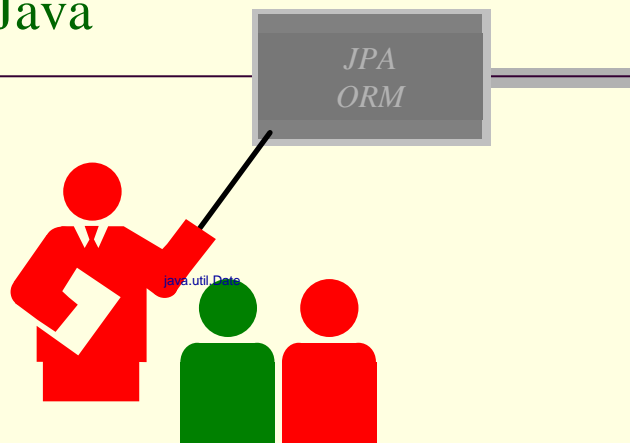
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

66

JPA – funcionamento flush

- Fora de uma transação (**Extended PC**)
 - Todas as operações **read-only** do **EM** podem ser executadas fora de uma transação (**find()**, **getReference()**, **refresh()**).
 - As modificações (**persist()**, **merge()**, **remove()**) podem ser executadas fora de uma transação mas elas serão enfileiradas até que o PC esteja associado a uma transação.
 - Algumas operações não podem ser chamadas fora de uma transação (**flush()**, **lock()** e **queries de update/delete**)

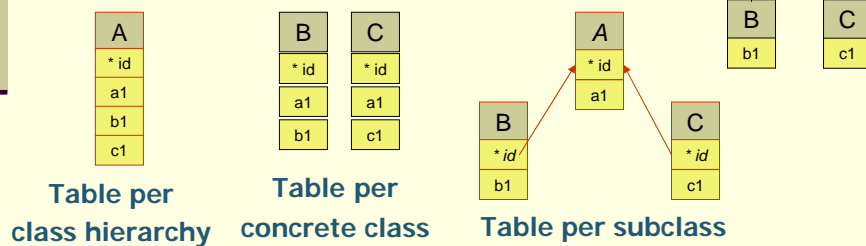
FPSW-Java



JPA ORM - Herança

- JPA oferece 3 estratégias de mapeamento de herança através da anotação `@Inheritance(strategy= InheritanceType.XXX)`. Onde XXX pode ser:

- SINGLE_TABLE** – Single table per class hierarchy;
- JOINED** – Table per subclass
- TABLE_PER_CLASS** – Table per concrete class;



April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br

69

JPA Herança – Single table per class

- Estratégia default, todas as classes de entidade em uma única tabela distinguidas por um atributo discriminador

```

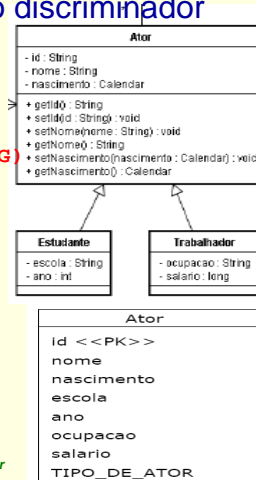
@Entity
@Inheritance(strategy=
    InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TIPO_DE_ATOR",
    discriminatorType=DiscriminatorType.STRING)
public abstract class Ator
    implements Serializable { ... }

@Entity
@DiscriminatorValue(value="T")
public class Trabalhador extends Ator { ... }

@Entity
@DiscriminatorValue(value="E")
public class Estudante extends Ator { ... }
    
```

April 05

Prof. Ismael H. F. Santos - ismael@tegraf.puc-rio.br



JPA Herança – Joined, Table per subclass

- Tabelas para cada classe de entidade. Tabelas são relacionadas através de pk e fk. Suporta facilmente operações polimórficas entre as classes, porem requer operações de **join** para agregar as subclasses, logo não é aconselhável para hierarquias profundas

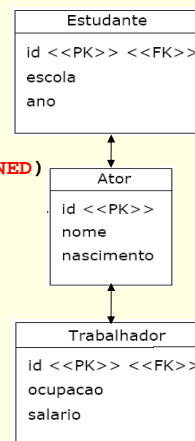
```
@Entity @Inheritance(strategy=InheritanceType.JOINED)
public abstract class Ator
    implements Serializable { ... }
```

```
@Entity @PrimaryKeyJoinColumn
public class Trabalhador extends Ator { ... }
```

```
@Entity @PrimaryKeyJoinColumn
public class Estudante extends Ator { ... }
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br



JPA Herança - Table per concrete class

- Uma Tabela para cada classe concreta. O suporte para esta estratégia foi tornado opcional para os **Persistent providers** devido as dificuldades de implementação de operações polimórficas.

```
@Entity
@Inheritance(strategy=
    InheritanceType.TABLE_PER_CLASS)
public abstract class Ator
    implements Serializable { ...
}
```

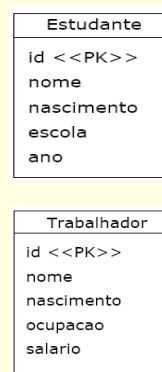
```
@Entity
public class Trabalhador extends Ator { ... }
```

```
@Entity
public class Estudante extends Ator { ... }
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

72



JPA ORM - Herança

■ Exemplo 5 – JPA Herança

Listagem 01. Produto.java

```
package exemplo3JPAHerança;
import javax.persistence.*;
@Entity @Inheritance(strategy=
    InheritanceType.TABLE_PER_CLASS)
public class Produto {
    @Id
    private int id;
    @Basic
    private String nome;
    @Basic
    private String descr;
    @Basic
    private double price;
    public Produto() {}
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getNome() {return nome;}
    public void setNome(String n){ nome = n;}
    public String getDescr() { return descr; }
    public void setDescr(String d){ descr = d; }
    public double getPrice() { return price; }
    public void setPrice(String p) {price = p; }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

73

Listagem 02. Book.java

```
package exemplo3JPAHerança;
import javax.persistence.*;
@Entity
public class Book extends Produto {
    @Basic
    private String autor;
    @Basic
    private String isbn;
    @Basic
    private String editora;
    public Book () {}
    public String getAutor(){return autor;}
    public void setAutor(String a) {
        autor = a; }
    public String getIsbn() {return isbn;}
    public void setIsbn(String i) {
        isbn = i; }
    public String getEditora() {
        return editora; }
    public void setEditora (String e) {
        editora = e; }
}
```

JPA ORM - Herança

■ Listagem 03. META-INF/orm.xml - Separando database schema de ORM

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
<description>JPA tutorial</description>
<package> exemplo3JPAHerança</package>
<entity class=" exemplo3JPAHerança.Produto" name="Produto">
    <table name="JPA_PRODUCTS"/>
    <attributes>
        <id name="id"> <generated-value strategy="TABLE"/> </id>
        <basic name="nome"> <column name="PRODUCT_NAME" length="100"/> </basic>
        <basic name="descr"> <column length="255"/> </basic>
        <basic name="price"> </basic>
    </attributes>
</entity>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

74

JPA ORM - Herança

■ Listagem 04. META-INF/orm.xml - database schema

```
<entity class="exemplo3JPAHerança.Book" name="Book">
  <table name="JPA_BOOKS"/>
  <attributes>
    <basic name="isbn"> <column name="ISBN" length="20"/></basic>
    <basic name="autor"> <column name="AUTOR" length="40"/> </basic>
    <basic name="editora"> <column name="EDITORA" length="40"/> </basic>
  </attributes>
</entity>

</entity-mappings>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

75

JPA ORM - Herança

■ Listagem 05. META-INF/exemplo3JPAHerança.xml - ORM

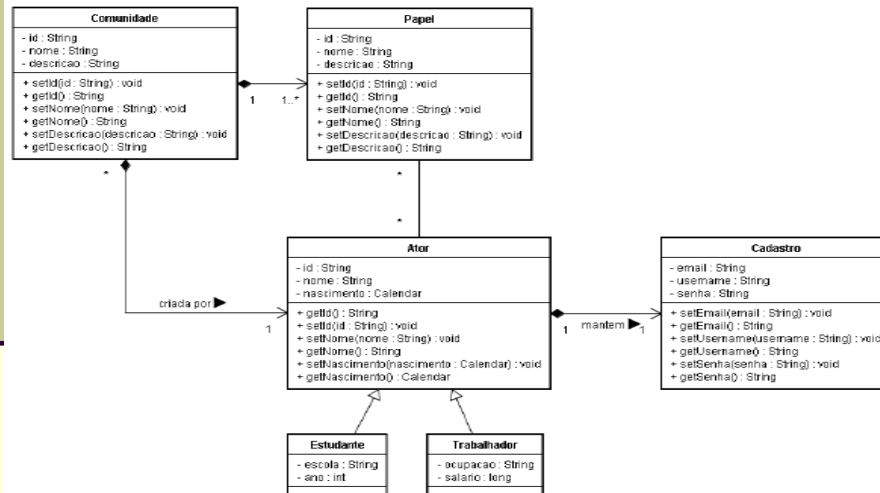
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
  <persistence-unit name="JPAHerança" transaction-type="RESOURCE_LOCAL">
    <mapping-file>exemplo3JPAHerança/orm.xml</mapping-file>
    <class>exemplo3JPAHerança.Produto</class>
    <class>exemplo3JPAHerança.Book</class>
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.show_sql" value="false">
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.username" value="user"/>
      <property name="hibernate.connection.password" value="passwd"/>
      <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    </properties>
  </persistence-unit>
</persistence>
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

76

JPA ORM – Mapeamento de Relacionamentos

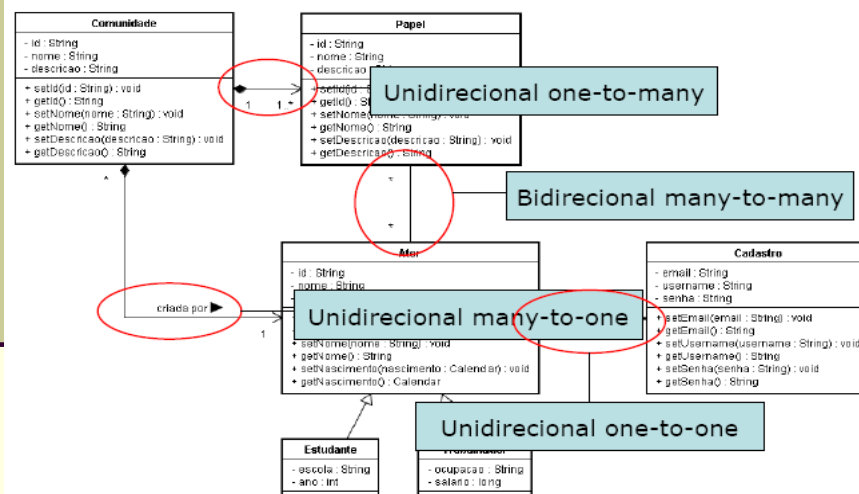


April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

77

JPA ORM – Mapeamento de Relacionamentos



April 05

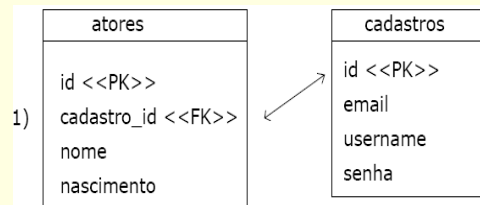
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

77

JPA ORM – Mapeamento de Relacionamentos 1 - 1

- Mapeamento one – to – one
 - Três formas
 - Relação com chave estrangeira (Código 1)

```
public class Ator implements Serializable {  
    @OneToOne(optional=false) // faz inner-join  
    @JoinColumn(name="cadastro_id")  
    private Cadastro cadastro;  
    ...  
} // Código 1
```



April 05

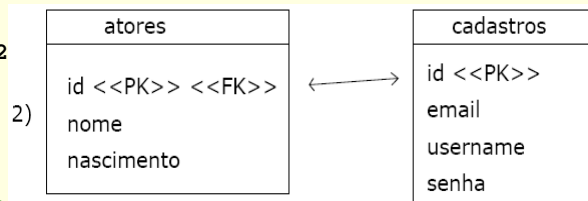
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

79

JPA ORM – Mapeamento de Relacionamentos 1 - 1

- Mapeamento one – to – one
 - Três formas
 - Relação por chave primária (Código 2)

```
public class Ator implements Serializable {  
    @OneToOne  
    @PrimaryKeyJoinColumn  
    private Cadastro cadastro;  
    ...  
} // Código 2
```



April 05

Pr

JPA ORM – Mapeamento de Relacionamentos 1 - 1

- Mapeamento one – to – one
 - Três formas
 - Com tabela associativa (Código 3)

```
public class Ator implements Serializable {
    @OneToOne
    @JoinTable( name="ATOR_CADASTRO",
        joinColumns = @JoinColumn(name = "Ator_ID"),
        inverseJoinColumns= @JoinColumn(name="Cadastro_ID") )
    private Cadastro cadastro;
    ...
} // Código 3
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

81

JPA ORM – Mapeamento 1 - 1

- Exemplo5 – JPA Mappings

Listagem 01. Entity1.java

```
package exemplo5JPAMappings1;
import javax.persistence.*;
@Entity
public class Entity1 {
    @Id
    private int id;
    private Entity2 e2;
    public Entity1 () {}
    public int getId() { return id; }
    public void setId(int id) {
        this.id = id;
    }
    @OneToOne
    public Entity2 getEntity2 () {return e2;}
    public void setEntity2(Entity2 e2) {
        this.e2 = e2;
    }
}
```

Listagem 02. Entity2.java

```
package exemplo5JPAMappings1;
import javax.persistence.*;
@Entity
public class Entity2 {
    @Id
    private int id;
    private Entity1 e1;
    public Entity2 () {}
    public int getId() { return id; }
    public void setId(int id) {
        this.id = id;
    }
    @OneToOne(mappedBy="entity2")
    public Entity1 getEntity1(){return e1;}
    public void setEntity1 (Entity1 e1) {
        this.e1 = e1;
    }
}
```

No relacionamento OneToOne unidirecional, A Entity2 não contém uma referência a Entity1.

April 05

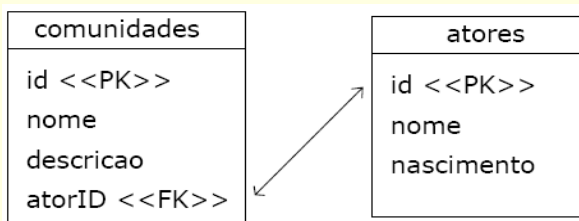
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

82

JPA ORM – Mapeamento de Relacionamentos M-1 e 1-M bidirecional

■ Mapeamento M-1 bidirecional

```
@Entity @Table(name = "COMUNIDADES")
public class Comunidade implements Serializable {
    @ManyToOne
    private Ator criador; // ator criador da comunidade
    public Ator getCriador(){ return criador; }
    public setCriador(Ator a ){ criador = a; }
    private String nome; ....
    private String descricao; ....
}
```



April 05

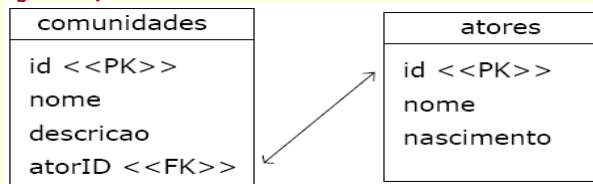
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

83

JPA ORM – Mapeamento de Relacionamentos M-1 e 1-M bidirecional

■ Mapeamento 1-M bidirecional - As interfaces do Java Collection são suportadas: java.util.Set, java.util.Collection, java.util.List, java.util.Map, java.util.SortedSet, java.util.SortedMap

```
@Entity @Table(name = "ATORES")
public class Ator implements Serializable {
    @OneToMany(mappedBy="criador", fetch=FetchType.EAGER)
    private Collection<Comunidade> listaComunidades;
    @Temporal(TemporalType.DATE)
    private Calendar nascimento; ....
    private String nome; ....
}
```



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

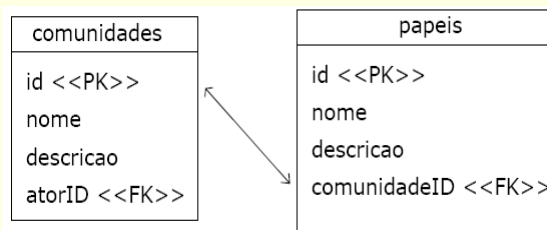
84

JPA ORM – Mapeamento de Relacionamentos M-1 e 1-M unidirecional

■ Mapeamento 1-M unidirecional

- Neste caso não é necessário o mapeamento do lado inverso

```
@Entity @Table(name = "COMUNIDADES")
public class Comunidade implements Serializable {
    @OneToMany(fetch=FetchType.LAZY)
    private Set<Papal> listaPapeis;
    private String nome; ....
    private String descricao; ....
}
```



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

85

JPA ORM – Mapeamentos 1-M e M-1

■ Exemplo5 – JPA Mappings

Listagem 01. Entity1.java

```
package exemplo5JPAMappings1M;
import javax.persistence.*;
@Entity
public class Entity1 {
    @Id
    private int id;
    public Entity1 () {}
    public int getId() { return id; }
    public void setId(int id) {
        this.id = id;
    }
    @ManyToOne
    private Entity2 e2;
    public Entity2 getEntity2(){return e2;}
    public void setEntity2(Entity2 e2) {
        this.e2 = e2;
    }
}
```

A Entity1 contém uma referência a uma instância única da Entity2.
A Entity2 contém uma referência a uma coleção de objetos da Entity1.

April 05

Listagem 02. Entity2.java

```
package exemplo5JPAMappings1M;
import javax.persistence.*;
@Entity
public class Entity2 {
    @Id
    private int id;
    public Entity2 () {}
    public int getId() { return id; }
    public void setId(int id) {
        this.id = id;
    }
    @OneToMany(mappedBy="entity2")
    private Collection<Entity1> lE1;
    public Collection<Entity1> getEntity1(){
        return lE1;
    }
    public void setEntity1(Collection<Entity1> e1){
        this.e1 = lE1;
    }
}
```

No relacionamento ManyToOne e OneToMany unidirecional, a Entity2 não contém uma referência a Entity1.

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

86

JPA ORM – Mapeamento de Relacionamentos M - M

■ Mapeamento many – to – many -> Ator x Papel

```
@Entity @Table(name = "ATORES")
public class Ator implements Serializable {
    @ManyToMany
    @JoinTable( name = "ATOR_PAPEL",
        joinColumns = {@JoinColumn(name = "ATOR_ID")},
        inverseJoinColumns={@JoinColumn(name= "PAPEL_ID")} )
    private Set<Papel> listaPapeis;
}
```

April 05

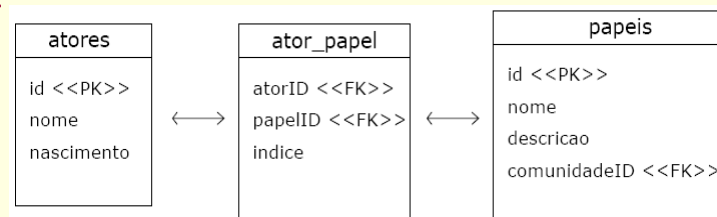
Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

87

JPA ORM – Mapeamento de Relacionamentos List M - M

■ Mapeamento List many – to – many

```
@Entity
@Table(name = "PAPEIS")
public class Papel implements Serializable {
    @ManyToMany( mappedBy="listaPapeis" )
    @IndexColumn(name="indice" ) // para manter a ordem de
    private List<Ator> atores; // criação de List
}
```



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

88

JPA ORM – Mapeamento M – M Bi

■ Exemplo5 – JPA Mapping

Listagem 01. Entity1.java

```
package exemplo5JPAMappingsMM;
import javax.persistence.*;
@Entity
public class Entity1 {
    @Id
    private int id;
    public Entity1 () {}
    public int getId() { return id; }
    public void setId(int id) {
        this.id = id;
    }
    @ManyToMany
    private Collection<Entity2> lE2;
    public Collection<Entity2> getEntity2()
    { return lE2; }
    public void setEntity2(Entity2 e2) {
        this.lE2 = e2;
    }
}
```

A Entity1 contém uma referência a uma coleção de objetos da Entity2.

A Entity2 contém uma referência a uma coleção de objetos da Entity1.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

Listagem 02. Entity2.java

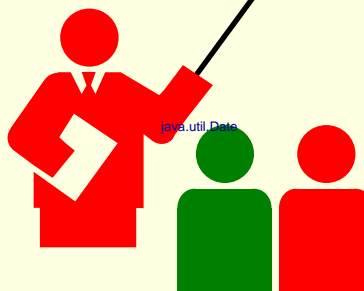
```
package exemplo5JPAMappingsMM;
import javax.persistence.*;
@Entity
public class Entity2 {
    @Id
    private int id;
    public Entity2 () {}
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    @ManyToMany(mappedBy="entity2")
    private Collection<Entity1> lE1;
    public Collection<Entity1> getEntity1(){
        return lE1;
    }
    public void setEntity1(Collection<Entity1>
    e1) {
        this.lE1 = e1;
    }
}
```

A Entity2 contém a annotation mappedBy, utilizada para referenciar a Entity1, dona do relacionamento. A Entity1 e Entity2 contém uma foreign key para a tabela (n para n) que contém o relacionamento entre Entity1 e Entity2.

89

FPSW-Java

JPA
Transitive
Persistence



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

90

JPA – Persistência Transitiva

- As operações (**persist**, **remove**, **merge**) realizadas com o banco podem resultar em **violações de integridade**
- O Hibernate possui mecanismos para tratar relações do tipo pai/filho automaticamente
 - Ex.: Ao remover um ator do banco não faz sentido manter o cadastro do ator
- Entidades tem o seu próprio ciclo de vida, suportam referencias compartilhadas (o que implica que removendo uma entidade de uma coleção não significa que a mesma pode ser deletada). Por default, não existe nenhuma relação de cascata de estados entre uma entidade e suas outras entidades associadas.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

91

JPA – Persistência Transitiva

- Para cada operação básica do EM (**persist**, **remove**, **merge**, **refresh**) existe um correspondente **cascade style**, respectivamente **CascadeType.PERSIST**, **REMOVE**, **MERGE**, **REFRESH** e **CascadeType.ALL**.
- **Recomendações**
 - Normalmente não faz sentido habilitar o cascade em associações **@ManyToOne** ou **@ManyToMany**;
 - Para o caso em que o ciclo de vida do filho estiver limitado ao do objeto pai, habilitar no pai **CascadeType.ALL**;
 - Fora estas situações, em geral, não será necessário a utilização de cascades

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

92

JPA Hibernate Provider – Persistência Transitiva

```
public class Ator implements Serializable {
    ...
    @OneToOne(cascade={CascadeType.ALL})
    @JoinColumn(name="cadastro_id", unique=true)
    private Cadastro cadastro;
    ...
}

public class Comunidade implements Serializable {
    ...
    @ManyToOne(cascade=CascadeType.PERSIST)
    @JoinColumn(name="Ator_ID", nullable=false)
    private Ator criador;
    ...
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

93

JPA Hibernate Provider – Carga Preguiçosa – Lazy Load

- Carga dos objetos é feita apenas quando necessário
 - Evita sobrecarregar a memória com objetos não necessários
 - Hibernate usa proxies das coleções Java
 - Pode ser difícil de se trabalhar (LazyInitializationException)

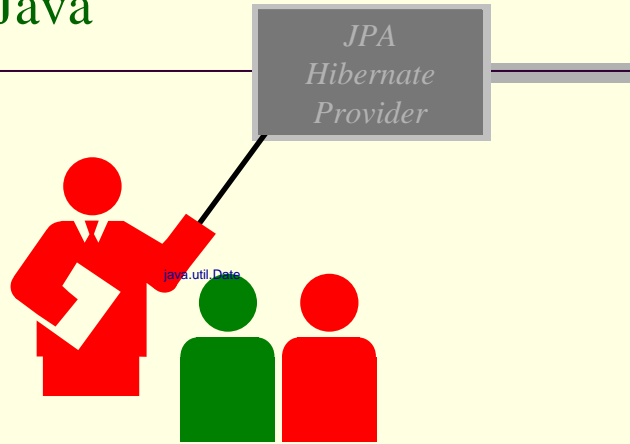
```
public class Comunidade implements Serializable {
    ...
    @ManyToOne(cascade=CascadeType.PERSIST,
               fetch=FetchType.LAZY)
    @JoinColumn(name="Ator_ID", nullable=false)
    private Ator criador;
    ...
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

94

FPSW-Java



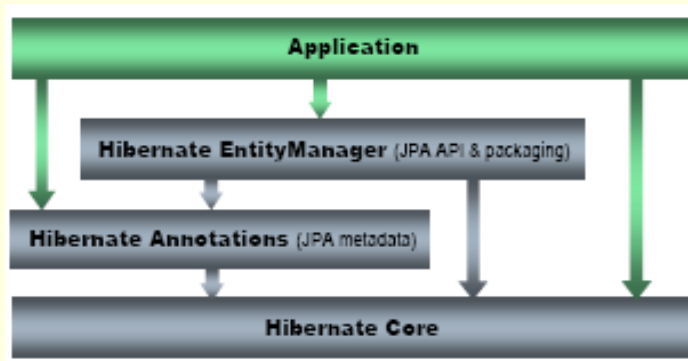
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

95

JPA Hibernate Provider

- A implementacao de JPA pelo Hibernate é feita segundo a seguinte arquitetura



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

96

JPA Hibernate Provider

- É necessário copiar as seguintes bibliotecas do Hibernate para o classpath de sua aplicação
- Baixar o arquivo com as bibliotecas do link abaixo:
http://.../~ismael/Cursos/Cidade_FPSW/.../JPA-Hibernate-libs.rar
 - From Hibernate EntityManager:
 - hibernate-entitymanager.jar
 - lib/ejb3-persistence.jar
 - lib/hibernate-annotations.jar
 - lib/hibernate-commons-annotations.jar
 - lib/jboss-archive-browsing.jar

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

97

JPA Hibernate Provider

- From Hibernate Core
 - hibernate3.jar, lib/antlr-2.7.6.jar
 - lib/asm-attrs.jar,
 - lib/asm.jar
 - lib/c3p0-0.9.1.jar
 - lib/cglib-2.1.3.jar
 - lib/commons-collections-2.1.1.jar
 - lib/commons-logging-1.0.4.jar
 - lib/concurrent-1.3.2.jar
 - lib/dom4j-1.6.1.jar
 - lib/ehcache-1.2.3.jar
 - lib/javassist.jar
 - lib/log4j-1.2.11.jar

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

98