

HIBERNATE

Hibernate 3 [Introdução]

- Trata-se de um framework de acesso a banco de dados escrito em Java.
- É um framework de código aberto bastante difundido.
- Permite mapear o modelo de objetos diretamente para o banco de dados (O/R).



Hibernate 3 [Introdução]

- Foi criado para reduzir o tempo que o desenvolvedor consome nas atividades relacionadas à persistência de dados no desenvolvimento de um software orientado a objetos.
- Similar aos frameworks JDO, Toplink...



Hibernate 3 [Introdução]

- Vantagens observadas:
 - Contempla modelo natural de programação OO;
 - Aceleração do desenvolvimento (poupa trabalho manual de conversão de dados relacionais para objetos);
 - Transparência de banco de dados;
 - Performance (cache);
 - Simplicidade;
 - Compatibilidade com os principais bancos de dados de mercado.

Hibernate 3 [Introdução]

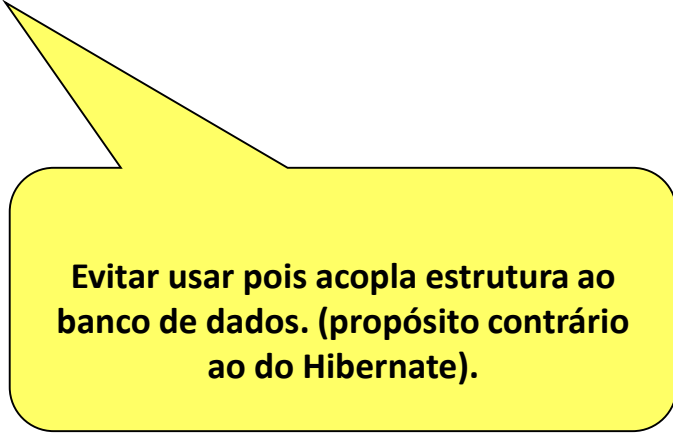
- Desvantagens observadas:
 - Não é a melhor opção para todos os tipos de aplicação. Sistemas que fazem uso extensivo de stored procedures, triggers ou que implementam a maior parte da lógica de negócio no banco de dados (modelo pobre) não vai se beneficiar com o uso do Hibernate. Ele é indicado para sistemas que contam com um modelo rico, onde a maior parte da lógica de negócios fica na própria aplicação dependendo pouco de funções específicas do banco de dados;
 - Consultas muito complexas e extensas exigem grau avançado de conhecimento da API (ajustes finos).

Hibernate 3 [Estrutura]

- Principais interfaces
- **Session** (org.hibernate.Session): Possibilita a comunicação entre a aplicação e a camada de persistência, através de uma conexão JDBC. Com ela é possível criar, remover, atualizar e recuperar objetos persistentes.
- **SessionFactory** (org.hibernate.SessionFactory): Mantém o mapeamento objeto-relacional em memória. Permite a criação de objetos Session, a partir dos quais os objetos são acessados. Também denominado de “fábrica de objetos Session”.
- **Configuration** (org.hibernate.Configuration): É utilizado para realizar as tarefas de inicialização do Hibernate.
- **Transaction** (org.hibernate.Transaction): É utilizada para representar uma unidade indivisível de uma operação de manipulação de dados.
- **Criteria** (org.hibernate.Criteria) e **Query** (org.hibernate.Query): utilizadas para realizar consultas ao banco de dados

Hibernate 3 [Estrutura]

- Oferece 3 maneiras de manipulação de dados:
 - via HQL (linguagem muito parecida com SQL);
 - via API Criteria;
 - via SQL nativo.

A yellow callout box with a black border and a pointer at the top left, containing text.

Evitar usar pois acopla estrutura ao banco de dados. (propósito contrário ao do Hibernate).

HIBERNATE 3 [Entidade]

A classe POJO representa a entidade.

```
package br.fiap.fiap;

public class Pessoa implements java.io.Serializable {

    private int id;
    private String nome;
    private String apelido;
    private String endereco;

    public Pessoa() {
    }

    //getters e setters

}
```


HIBERNATE 3 [Mapeamento]

No arquivo XML ou através das Anotações, além dos atributos, são configurados os relacionamentos e comportamentos.

```
<hibernate-mapping>
  <class name="br.com.fiap.Pessoa" table="pessoa" catalog="introhibernate">
    <id name="id" type="int">
      <column name="ID" />
      <generator class="assigned" />
    </id>
    <property name="nome" type="string">
      <column name="NOME" length="45" />
    </property>
    <property name="apelido" type="string">
      <column name="APELIDO" length="45" />
    </property>
    <property name="endereco" type="string">
      <column name="ENDERECO" length="45" />
    </property>
  </class>
```

HIBERNATE 3 [Configuração]

A classe HibernateUtil.java: responsável por ler o arquivo hibernate.cfg.xml e armazenar seus dados em estruturas na memória.

```
package br.com.fiap;

import org.hibernate.cfg.Configuration;
import org.hibernate.SessionFactory;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new
                Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }
}
```

HIBERNATE 3 [Configuração]

A classe `HibernateUtil.java`: responsável por ler o arquivo `hibernate.cfg.xml` e armazenar seus dados em estruturas na memória.

```
public static SessionFactory getSessionFactory() {  
    return sessionFactory;  
}  
}
```

HIBERNATE 3 [Execução]

A classe helper: usada para manipular os objetos persistentes.

```
package br.com.fiap;

import org.hibernate.*;
import java.util.*;

public class PessoasHelper {
    Session session = null;

    public PessoasHelper(){
        session = HibernateUtil.getSessionFactory().getCurrentSession();
    }
}
```

HIBERNATE 3 [Execução]

A classe helper: usada para manipular os objetos persistentes.

```
public List<Pessoa> getPessoas(){  
    List<Pessoa> lista = new ArrayList<Pessoa>();  
    try{    Transaction tx = session.beginTransaction();  
        Query q = session.createQuery("from Pessoa");  
        lista = q.list();  
    }  
    catch(Exception ex){  
        ex.printStackTrace();  
    }  
    return lista;  
}
```

HIBERNATE 3 [Execução]

A classe helper: usada para manipular os objetos persistentes.

```
public String salvarPessoa(Pessoa p){  
    try{ Transaction tx = session.beginTransaction();  
        session.save(p);  
        tx.commit();  
        return "Dados inseridos";  
    }  
    catch(Exception ex){  
        return ex.getMessage();  
    }  
}
```

HIBERNATE 3 [Execução]

Inserindo um novo registro

```
PessoasHelper helper = new PessoasHelper();  
Pessoa p = new Pessoa();  
p.setId(100);  
p.setNome("Ze");  
p.setApelido("ze");  
p.setEndereco("aclimacao");  
  
out.print(helper.salvarPessoa(p));
```

HIBERNATE 3 [Alguns Métodos]

O Hibernate possui um conjunto de métodos para incluir e atualizar valores no banco de dados. Os métodos se parecem bastante, mas possuem diferenças:

save() – armazena um objeto no banco de dados. Isso significa um novo registro de o identificador não existe, senão ele lança uma exceção.

update() – utilizado para atualizar o objeto através do seu identificador. Se não existir, o método lança uma exceção.

saveOrUpdate() – Este método chama **save()** ou **update()** dependendo da operação. Se o identificador existir, será chamado **update()**; senão será chamado o método **save()**

HIBERNATE 3 [Alguns Métodos]

Carregando um objeto: métodos `load()` e `get()`

O método `load()` fornece um meio de se recuperar uma instância persistente se o identificador é conhecido.

`load()` toma a classe e carrega seu estado em uma nova instância desta classe, em um estado persistente.

O método `load()` lançará uma exceção se não houver um registro válido com a chave informada. Se não houver a certeza de que exista o registro, é recomendável usar o método `get()`, que acessa o banco e retorna null se não houver um registro correspondente.

```
Cat cat = new DomesticCat();  
// carrega o estado de pk em um objeto Cat  
sess.load( cat, new Long(pkId) );  
Set kittens = cat.getKittens();
```

HIBERNATE 3 [Alguns Métodos]

Executando consultas – métodos `list()` e `uniqueResult()`

Se o identificador do objeto não for conhecido, uma consulta (query) será necessária. Sabe-se que o Hibernate utiliza instruções HQL (Hibernate Query Language). Para a criação de consultas geradas programaticamente, é possível utilizar instruções SQL nativas do banco de dados utilizado, com o suporte adicional do Hibernate para a conversão de ResultSets em objetos.

A estrutura HQL e as consultas SQL são representadas por uma instância de Query. Esta interface oferece métodos para ações de consultas. Pode-se sempre obter uma Query usando-se Session atual.

Uma consulta é executada geralmente pela chamada ao método `list()`. Este resultado é atribuído a uma coleção. O método `uniqueResult()` fornece um atalho se o usuário souber que a consulta retorna um único resultado.

```
List mothers = session.createQuery(  
    "select mother from Cat as cat join cat.mother as mother where cat.name  
    = ?")  
    .setString(0, name)  
    .list();
```

HIBERNATE 3 [Alguns Métodos]

Iterando resultados – método iterate()

É possível obter uma melhor performance executando a query usando o método `iterate()`. Isso normalmente acontecerá quando se desejar que as instâncias retornadas pela query já estiverem em sessão ou em cache. Se ainda não estiverem em cache, o método será mais lento que `list()` e realizará múltiplos acessos ao banco de dados para uma simples consulta.

```
// buscando ids
Iterator iter = sess.createQuery("from eg.Qux q order by q.likeliness").iterate();
while ( iter.hasNext() ) {
    Qux qux = (Qux) iter.next(); // busca o objeto
    // algo que não existe na query
    if ( qux.calculateComplicatedAlgorithm() ) {
        // remove a instancia atual
        iter.remove();
        // continuação...
        break;
    }
}
```

Hibernate 3 [Configuração] - Dialeto

- SQL Dialeto
 - Dialeto SQL possibilitam que o Hibernate tire proveito de características próprias do banco de dados;
 - Deve ser configurado utilizando o nome completo de uma subclasse de “net.sf.hibernate.dialect.Dialect”
- Exemplo:
 - `hibernate.dialect=net.sf.hibernate.dialect.MySQLDialect`

Esta configuração é feita dentro do arquivo “**hibernate.cfg.xml**”.

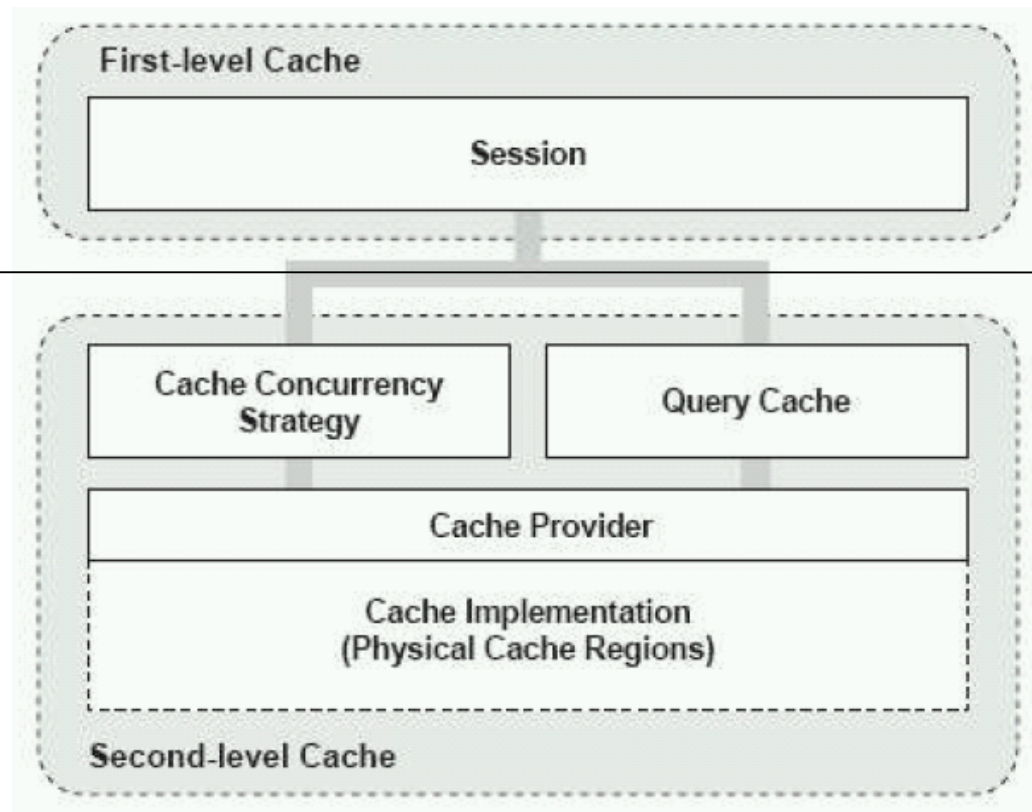


Hibernate 3 [Configuração] - Cache

- Caching
 - Os frameworks de mapeamento objeto relacional são ideais para uso de Caching.
- Usos do cache:
 - Busca pela chave primária
 - Associações com Lazy Loading
 - Queries

Hibernate 3 [Configuração] - Cache

Níveis de cache



Hibernate 3 [Configuração] - Cache

- Cache nível 1;
 - Ativado por default;
 - Sempre que houver chamada de load(), find(), list(), iterate();

Hibernate 3 [Configuração] - Cache

- Provedor de Cache
 - EHCache
 - OpenSymphony OSCache
 - JBossCache
 - ...

Hibernate 3 [Configuração] - Cache

Cache (geral)

```
<hibernate-configuration>
  <session-factory>

    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/estatistica</prope
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password"></property>
    <property name="show_sql">>true</property>

    <property name="cache.provider_class">net.sf.ehcache.hibernate.Provider</prope

    <mapping resource="br/edu/fiap/persistencia/common/RespostaORM.hbm.xml"/>
    <mapping resource="br/edu/fiap/persistencia/common/QuestaoORM.hbm.xml"/>
```

Hibernate 3 [Configuração] - Cache

```
<cache name="model.Noticia"  
    maxElementsInMemory="100"  
    eternal="false"  
    timeToIdleSeconds="300"  
    timeToLiveSeconds="600"  
    overflowToDisk="false"  
/>
```

maxElementsInMemory: número máximo de objetos que podem ficar armazenados em memória.

eternal: se configurado para “true”, significa que o cache nunca vai expirar.

timeToIdleSeconds: tempo em segundos que um objeto pode permanecer inutilizado no cache.

timeToLiveSeconds: tempo em segundos que um objeto pode ficar em cache.

overflowToDisk: se configurado para “true” e caso o limite de objetos em memória seja superior ao definido pela propriedade “maxElementsInMemory”, as informações serão armazenadas em disco.

Hibernate 3 [Dicas]

- Utilizar ferramentas para automatizar geração de códigos;
- Utilizar cache;
- Utilizar HQL em operações muito complexas;
- Configurar adequadamente relacionamentos.

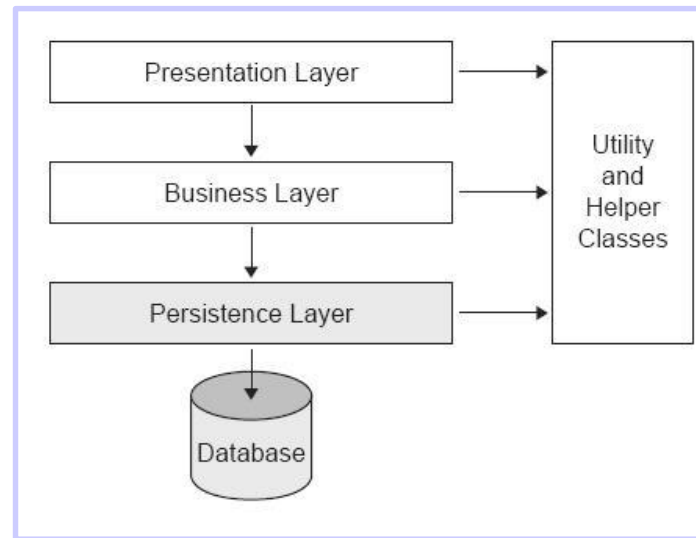
Java Persistence API (JPA)

JPA

- Java Persistence API

- Especificação que descreve como trabalhar com mapeamento objeto relacional. Foi concebida na JSR-220 (EJB 3).
- Hoje pertence a JSR-317 (JPA 2.0).

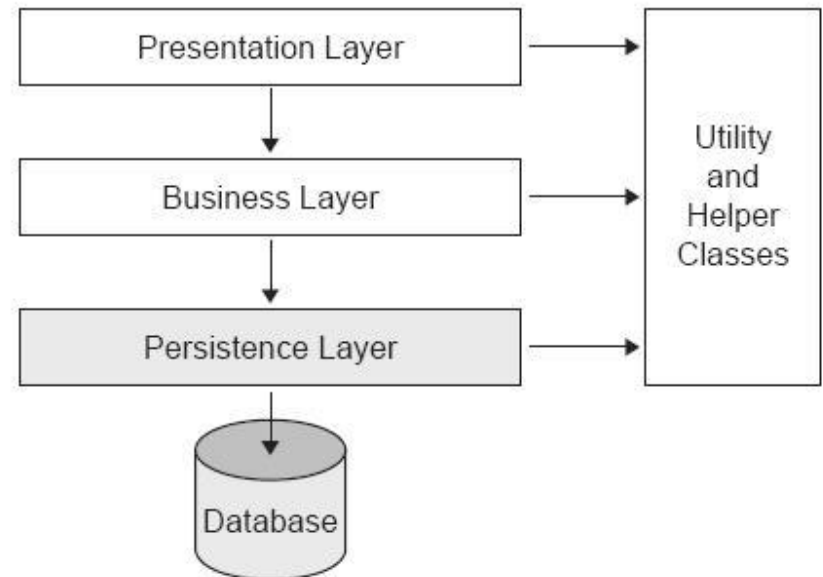
- Na definição de uma nova versão de EJB, no contexto de EJB, percebeu-se que seria melhor fazer uma especificação seguindo o estilo dos frameworks de mercado.



JPA

- Padronização do mecanismo de persistência.
- Definição de metadados (anotações)
- Linguagem de consultas JP-QL

- Hibernate implementa JPA.
- EclipseLink implementa JPA.



JPA [Entity Manager]

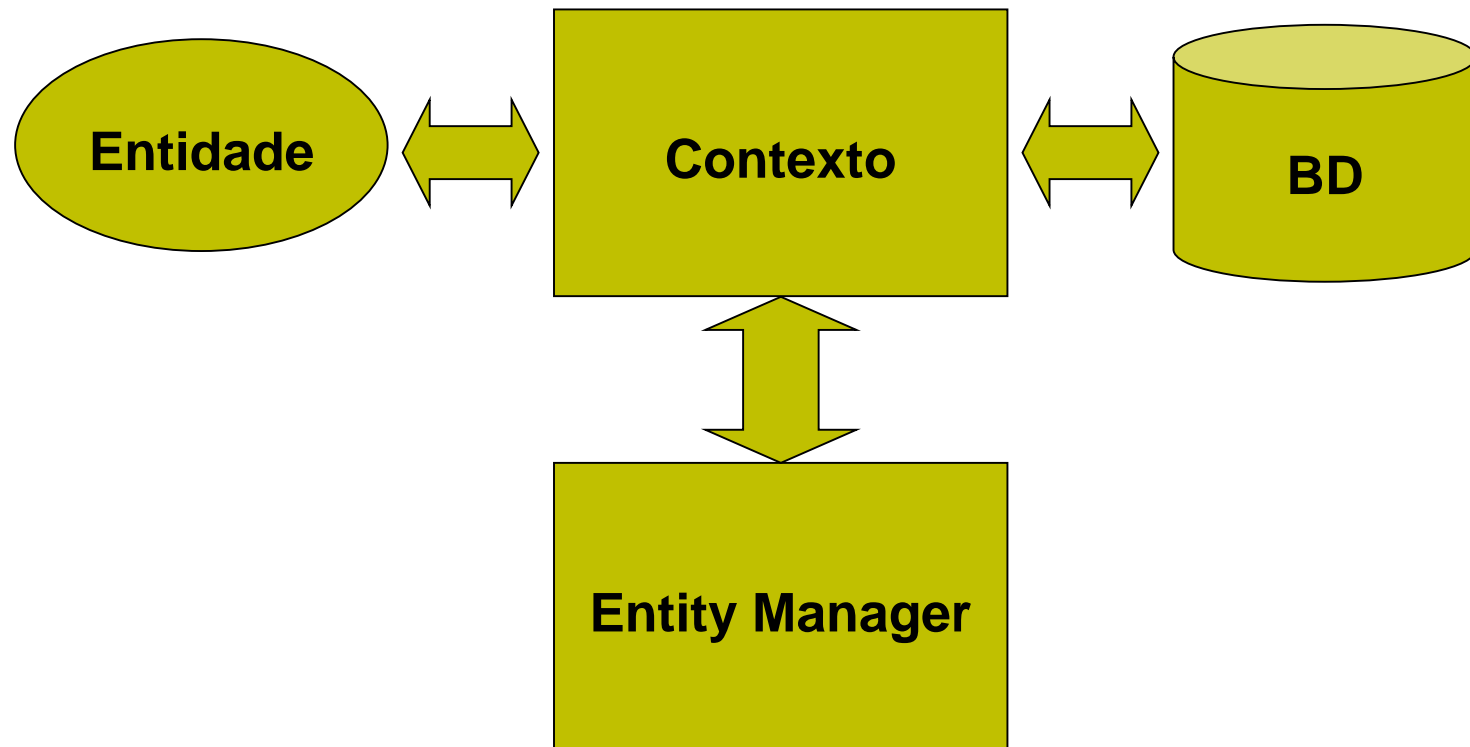
- Unidade central para gerenciamento de entidades na JPA através de uma API padronizada;
- Responsável pela criação, atualização, remoção e consulta às entidades (CRUD);
- Controle de transações;
- Gerenciamento de cache;

JPA [Definições]

- Unidade de Persistência:
 - conjunto fixo de classes mapeadas para o banco de dados (persistence.xml);
- Contexto de Persistência:
 - conjunto de instâncias de entidades gerenciadas de um Entity Manager;
- Entidades Gerenciadas:
 - quando entidades estão associadas a um contexto de persistência;
 - alterações no estado das entidades são sincronizadas com o banco de dados;
 - ao fechar um contexto de persistência, todas suas instâncias de entidades associadas tornam-se não gerenciadas;
- Entidades Não Gerenciadas:
 - entidades não associadas a um contexto de persistência (por exemplo quando são instanciadas);
 - alterações nas entidades não se refletem no banco de dados;

JPA [Contexto de Persistência]

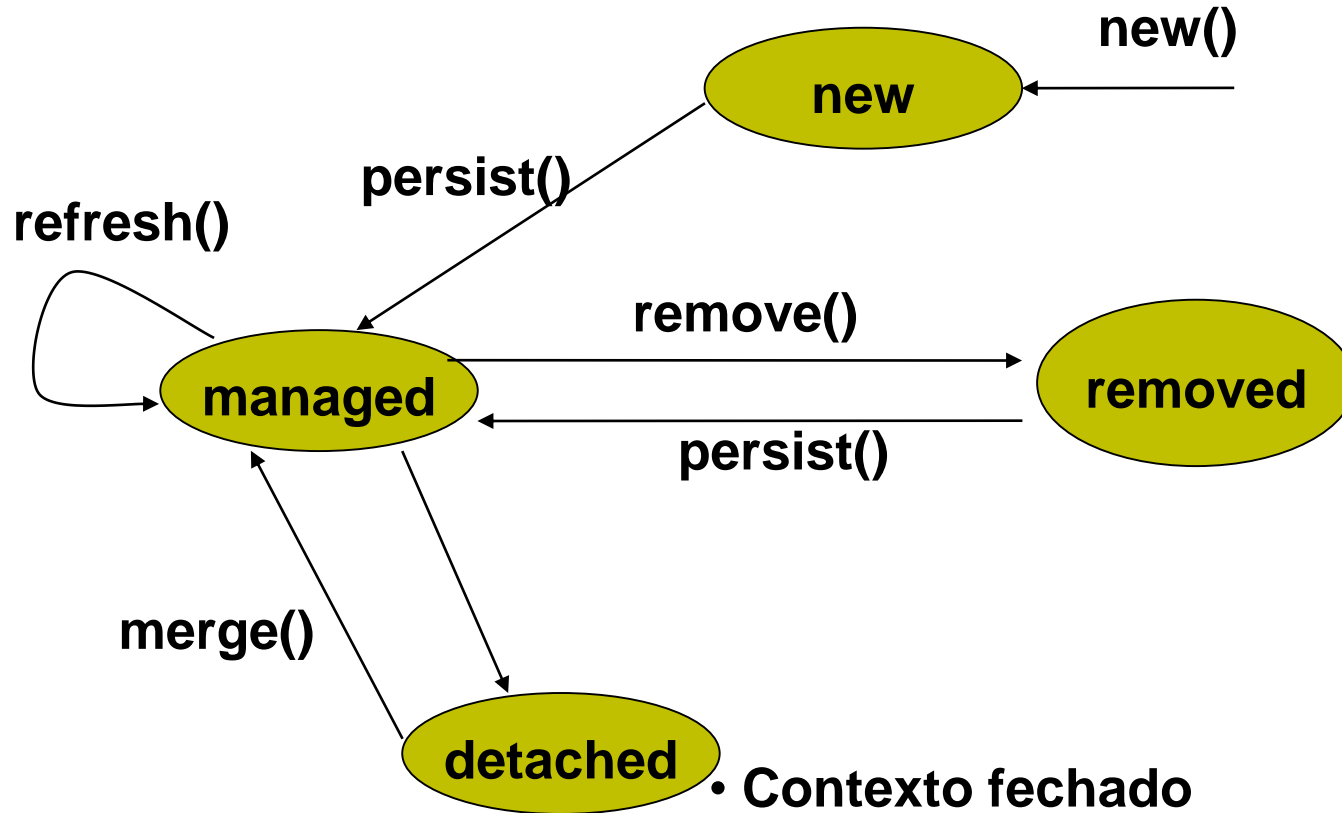
- Ponte entre entidades na memória e banco de dados;
- Gerenciada por um Entity Manager;



JPA [Estados Entidade]

- new → instância da entidade criada em memória mas não associada a um contexto de persistência e não possui id equivalente no banco de dados;
- managed → tem um id no banco de dados e está associada a um contexto de persistência;
- Detached → tem um id no bando de dados mas não está associada ao contexto de persistência;
- removed → instância da entidade associada a um contexto de persistência mas está programada para ser removida do banco de dados;

JPA [Diagrama de Estados Entidade]



- Contexto fechado
- Commit
- Serialização

JPA [Exemplo de Aplicação]

O arquivo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  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_2_0.xsd">
  <persistence-unit name="ExemploJPA">
    <class>br.com.fiap.Forum</class>
    <class>br.com.fiap.Usuario</class>
    <exclude-unlisted-classes>>false</exclude-unlisted-classes>

    <properties>
      <property name="hibernate.connection.username"
value="root"/>
```

JPA [Exemplo de Aplicação]

```
        <property name="hibernate.connection.password"
value="password"/>
        <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
        <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/forum"/>
        <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.NoCacheProvider"/>
        <property name="hibernate.hbm2ddl.auto"
value="update"/>
        <property name="hibernate.format_sql" value="true"/>
    </properties>
</persistence-unit>
</persistence>
```

JPA [Exemplo de Aplicação]

AS entidades **Forum** e **Usuario**

```
package br.com.fiap;
```

```
import java.io.Serializable;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name="FORUM", schema="forum")
```

```
public class Forum implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

JPA [Exemplo de Aplicação]

```
@Id
@SequenceGenerator(name="FORUM_IDFORUM_GENERATOR" )
@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="FORUM_IDFORUM_GENERATOR")
private int id;
private String assunto, descricao;

@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.LAZY)
private List<Usuario> usuarios;

public Forum() {
    usuarios = new ArrayList<Usuario>();
}
//getters e setters
}
```

JPA [Exemplo de Aplicação]

```
package br.com.fiap;

import java.io.Serializable;
import javax.persistence.*;

import java.util.List;

@Entity
@Table(name="USUARIO",schema="forum")
public class Usuario implements Serializable {
    private static final long serialVersionUID = 1L;
```


JPA [Exemplo de Aplicação]

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;

private String nome, email;

public Usuario() {
}
```

JPA [Exemplo de Aplicação]

Utilização do objeto EntityManager

```
package br.com.fiap;
```

```
import java.util.List;
```

```
import javax.persistence.EntityManager;
```

```
public class ForumJavaPersistenceDAO {
```

```
    private EntityManager em;
```

```
    public ForumJavaPersistenceDAO(EntityManager em) {
```

```
        this.em = em;
```

```
    }
```

JPA [Exemplo de Aplicação]

```
public Forum createForum(String assunto, String descricao){  
    Forum forum = new Forum();  
    forum.setDescricao(descricao);  
    forum.setAssunto(assunto);  
    em.getTransaction().begin();  
    em.persist(forum);  
    em.getTransaction().commit();  
    return forum;  
}
```

JPA [Exemplo de Aplicação]

```
public Usuario createUsuario(String nome,String email){
    Usuario usuario = new Usuario();
    usuario.setNome(nome);
    usuario.setEmail(email);
    em.getTransaction().begin();
    em.persist(usuario);
    em.getTransaction().commit();
    return usuario;
}

public Forum findForum(int id){
    return em.find(Forum.class, id);
}
```

JPA [Exemplo de Aplicação]

```
public Forum changeDescricaoForum(int id, String descricao){  
    Forum forum = this.findForum(id);  
    forum.setDescricao(descricao);  
    return forum;  
}  
  
public void deleteForum(int id){  
    Forum forum = this.findForum(id);  
    em.remove(forum);  
}
```

JPA [Exemplo de Aplicação]

```
public Forum addUsuarioToForum(int id, Usuario usuario){
    Forum forum = this.findForum(id);
    forum.getUsuarios().add(usuario);
    return forum;
}

public List<Usuario> listUsuariosFromForum(int id){
    List<Usuario> usuarios = this.findForum(id).getUsuarios();
    return usuarios;
}

public void closeEntityManager() {
    this.em.close();
}
}
```

JPA [Anotações JPA]

As anotações JPA estão localizadas no pacote **javax.persistence** e são apresentadas nas seções que se seguem.

@Entity

Especifica que uma classe é uma entidade.

@Entity

```
public class ClienteEntity {  
    private int id;  
    private String nome;  
    // métodos get e set  
}
```

JPA [Anotações JPA]

@Table

Especifica a tabela associada à entidade no banco de dados. Caso não definido, assume-se que a tabela terá o mesmo nome da classe da entidade.

@Entity

@Table(name="TAB_CLIENTE")

```
public class ClienteEntity {  
    private int id;  
    private String nome;  
    // métodos get e set  
}
```


JPA [Anotações JPA]

@Column

Especifica o campo associada ao atributo da entidade. Caso não definido assume-se que o campo terá o mesmo nome do atributo.

Parâmetros:

Name → nome do campo;

Unique (default false) → não permite duplicidade;

Nullable (default false) → não permite valores nulos;

Insertable (default true) → atributo utilizado em operações de INSERT;

Updatable (default false) → atributo utilizado em operações de UPDATE;

JPA [Anotações JPA]

@Column

@Entity

@Table(name="TAB_CLIENTE")

public class ClienteEntity {

@Column(name="COD_CLIENTE")

private int id;

@Column(name="NOM_CLIENTE", nullable=false)

private String nome;

// métodos get e set

}

JPA [Anotações JPA]

@Id

Atributo que identificará unicamente as instâncias da entidade. Deve-se sempre definir o atributo que representará a chave primária.

@Entity

@Table(name="TAB_CLIENTE")

public class ClienteEntity {

@Id

@Column(name="COD_CLIENTE")

private int id;

@Column(name="NOM_CLIENTE", nullable=false)

private String nome;

// métodos get e set

}

JPA [Anotações JPA]

@GeneratedValue

Especifica a estratégia de geração de valores para atributos que são chave primária.

Parâmetros:

Strategy → indica o tipo de estratégia utilizada;

Generator → nome do gerador de chaves;

Tipos mais comuns:

GeneratorType.SEQUENCE → baseado em sequence;

GeneratorType.IDENTITY → campos identidade;

JPA [Anotações JPA]

@SequenceGenerator

Define um gerador de chave primária baseado em *sequence* de banco de dados. Possui uma associação com o **@GeneratedValue**.

Parâmetros:

Name → nome a ser referenciado pelo **@GeneratedValue**;

sequenceName → nome da *sequence* de banco de dados;

allocationSize (default 50) → increment

JPA [Anotações JPA]

@Entity

**@SequenceGenerator(name="cliente", sequenceName="SEQ_CLIENTE",
allocationSize=1)**

@Table(name="TAB_CLIENTE")

public class ClienteEntity {

@Id

@GeneratedValue(strategy=GeneratorType.SEQUENCE, generator="cliente")

@Column(name="COD_CLIENTE")

private int id;

@Column(name="NOM_CLIENTE", nullable=false)

private String nome;

// métodos get e set

}

JPA [Anotações JPA]

@Transient

Indica que determinado atributo não deve ser persistido.

@Entity

@Table(name="TAB_CLIENTE")

public class ClienteEntity {

@Id

@Column(name="COD_CLIENTE")

private int id;

@Column(name="NOM_CLIENTE", nullable=false)

private String nome;

@Transient

private int chaveAcesso;

}

JPA [Anotações JPA]

@Temporal

Especifica o tipo de dado a ser armazenado em atributos do tipo *Date* e *Calendar*.

Parâmetros:

TemporalType.TIMESTAMP → data e hora;

TemporalType.DATE → somente data;

TemporalType.TIME → somente hora;

JPA [Anotações JPA]

@Entity

@Table(name="TAB_CLIENTE")

public class Cliente {

@Id

@Column(name="COD_CLIENTE")

private int id;

@Column(name="DAT_NASCIMENTO")

@Temporal(value=TemporalType.DATE)

private Date dataNascimento;

// ... Métodos get / set

}

JPA [Anotações JPA]

@PersistenceContext

Utilizar a anotação **@PersistenceContext** para obter, via injeção de dependência, uma instância que implemente a interface **EntityManager**.

```
public class ClienteBean ... {  
    @PersistenceContext  
    EntityManager manager = null;  
}
```

JPA [Anotações JPA]

@OneToMany

Utilizar a anotação **@OneToMany** no atributo que representa a associação.

Utilizar o atributo **cascade** para indicar:

CascadeType.ALL → todas as operações na entidade pai serão refletidas na(s) filho(s);

CascadeType.MERGE → somente operação de merge será refletida;

CascadeType.PERSIST → somente operação de persist será refletida;

CascadeType.REFRESH → somente operação refresh será refletida;

CascadeType.REMOVE → somente operação remove será refletida;

Podem-se combinar vários tipos:

@OneToMany(cascade={CascadeType.MERGE, CascadeType.REMOVE})

JPA [Anotações JPA]

Utilizar a anotação **@JoinColumn** em conjunto com **@OneToMany** para indicar o nome da coluna que representa a chave estrangeira na tabela filho. Pode ser utilizada em uma associação unidirecional.

```
public class NFEntity {  
    ...  
    @OneToMany(cascade=CascadeType.ALL)  
    @JoinColumn(name="COD_NF")  
    private Collection<NFItemEntity> itens;  
    ...  
}  
public class NFItemEntity {  
    ...  
}
```

JPA [Anotações JPA]

@ManyToOne

A anotação **@ManyToOne** pode ser utilizada para indicar que uma associação é bidirecional. Nas associações bidirecionais existe uma referência tanto do pai para o filho quanto do filho para o pai. Na entidade pai utilizar o atributo **mappedBy** para indicar o nome do atributo no filho que representa a ligação com o pai.

Dispensa o uso de **@JoinColumn** na entidade pai, uma vez que a mesma é definida na entidade filho.

JPA [Anotações JPA]

```
public class NFEntity {  
    ...  
    @OneToMany(cascade=CascadeType.ALL,  
    private Collection<NFItemEntity> itens;  
    ...  
}  
public class NFItemEntity {  
    ...  
    @ManyToOne  
    @JoinColumn(name="COD_NF")  
    private NFEntity nf;  
    ...  
}
```

mappedBy="nf")

JPA [Anotações JPA]

@ManyToMany

Utilizada para representar associações M:N. Possui o mesmo atributo **cascade** da anotação **@OneToMany**.

Utilizar a anotação **@JoinTable** associada para referenciar a tabela associativa e os campos de chave estrangeira. Parâmetros:

name → nome da tabela associativa;

joinColumns → colunas de chave estrangeira que referenciam a entidade diretamente;

inverseJoinColumns → colunas de chave estrangeira que referenciam a entidade no outro lado da relação;

Utilizar a anotação **@JoinColumn** para definir as referências diretas e inversas.

Parâmetro:

name → nome da coluna de chave estrangeira;

JPA [Anotações JPA]

```
public class CursoEntity {  
    ...  
    @ManyToMany  
    @JoinTable(name="TAB_ALUNO_CURSO",  
        joinColumns=@JoinColumn(name="COD_CURSO"),  
        inverseJoinColumns=@JoinColumn(name="COD_ALUNO"))  
    private Collection<AlunoEntity> alunos;  
    ...  
}
```


JPA [Anotações JPA]

@Embeddable (Chave Composta)

Chaves compostas podem ser representadas através de uma classe com a anotação **@Embeddable**. A classe de chave primária deve ser utilizada como se fosse um **@Id**, só que com a anotação **@EmbeddedId**;
Esta classe deve ser serializable. Deve implementar os métodos **equals(Object)** e **hashCode()**.

JPA [Anotações JPA]

@Embeddable

```
public class MatriculaID implements Serializable {  
    @ManyToOne  
    @JoinColumn(name="COD_ALUNO")  
    private Aluno aluno;  
    @ManyToOne  
    @JoinColumn(name="COD_CURSO")  
    private Curso curso;  
    public MatriculaID(){ }  
    public MatriculaID(Aluno aluno, Curso curso){  
        this.aluno = aluno;  
        this.curso = curso;  
    }  
}
```

JPA [Anotações JPA]

```
}  
// get e set  
@Override  
public boolean equals(Object arg0) {  
    return super.equals(arg0);  
}  
@Override  
public int hashCode() {  
    return super.hashCode();  
}  
}
```

JPA [Anotações JPA]

@Entity

@Table(name="TAB_ALUNO_CURSO")

public class Matricula {

@EmbeddedId

private MatriculaID id;

@column(name="DAT_MATRICULA")

private Date data;

// get e set

}

Bibliografia Básica

- BAUER, Christian; KING, Gavin. Hibernate in Action. USA: Manning Publications, 2004.
- BAUER, Christian; KING, Gavin. Java Persistence with Hibernate. USA: Manning Publications, 2006.
- BEGIN, Clinton; GOODIN, Brandon; MEADORS, Larry. Ibatis in Action. USA: Manning Publications, 2006.
- FISHER, Maydene; ELLIS, Jon; BRUCE, Jonathan. JDBC API Tutorial and Reference. USA: Addison-Wesley Professional, 2003.
- KEITH, Mike; SCHINCARIOL, Merrick. Pro EJB 3 – Java Persistence API. Apress, 2006.
- REESE, George. Database Programming With JDBC and Java. USA: O'Reilly, 2000.
- SPEEGLE, Gregory D. JDBC: Practical Guide for Java Programmers (The Practical Guides). USA: Paperback, 2001.
- WILLIAMSON; ALAN Moran. Java Database Programming: Servlets & JDBC. São Paulo: Pearson, 2000.