# *Desenvolvimento de Aplicações para WEB*

## ❖ Persistência

A Persistência de dados é um meio para que um aplicativo persista e recupere informações de um sistema de armazenamento. Sem essa capacidade, os dados só existiriam na **RAM**, e seriam perdidos quando a **RAM** parasse.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="ERPWEB2023PU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:app/erpweb2023JNDI</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.transaction.jta.platform"
       value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

## ❖ JDBC

Possibilita que uma aplicação construída na linguagem consiga acessar um banco de dados configurado **Local** ou **Remotamente.**

## ❖ Hibernante com Persistência

O Hibernante simplifica a persistência de dados em **Java**, fornecendo um mapeamento transparente entre **Objeto**s **Java** e **Tabelas** de **Banco de Fados**, gerenciamento do Ciclo de Vida dos **Objetos**, recursos avançados de consulta e suporte a transações.

Simplificando o processo de armazenar e recuperar **Objetos Java** em um banco de dados relacional, abstraindo a complexidade do acesso ao banco de dados e fornecendo um mecanismo eficiente para a persistência dos dados.

## ❖ Classe Genérica

As Classes Genéricas encapsulam operações que não são específicas de um determinado tipo de dados. Permitem escrever códigos mais seguros e fáceis de ler pelo fato de utilizarem menos conversões de um **Objeto** em outro.

```
17
19
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
```

```java
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    protected abstract EntityManager getEntityManager();

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    public void salvar(T entity) {
        getEntityManager().merge(entity);
    }

    public void remover(T entity) {
        getEntityManager().remove( entity: getEntityManager().merge(entity));
    }

    public T buscar(Object id) {
        return getEntityManager().find(entityClass,  primaryKey: id);
    }

    public List<T> listaTodos() {
        Query q = getEntityManager().createQuery("from "
                + entityClass.getSimpleName());
        return q.getResultList();
    }

    public List<T> listaFiltrando(String filtro, String... atributos) {
        String hql = "from " + entityClass.getSimpleName() + " obj where ";
        for (String atributo : atributos) {
            hql += "lower(obj." + atributo + ") like :filtro OR ";
        }
        hql = hql.substring( beginIndex: 0, hql.length() - 3);
        Query q = getEntityManager().createQuery( qlString: hql);
        q.setParameter( name: "filtro", "%" + filtro.toLowerCase() + "%");
        return q.getResultList();
    }
}
```

Também é responsável por realizar o **CRUD**, Sigla que consiste nas quatro operações básicas de todo banco de dados/aplicação de cadastro, que são: **Create, Read, Update, Delete.**

## ❖ EJB

O **EJB**, ou **Enterprise JavaBeans**, é um componente que roda em um container de servidor de aplicação. É utilizado para facilitar o desenvolvimento de aplicações **Java**, disponibilizando componentes distribuídos, transacionais, seguros e portáteis.

Em resumo ele Instancia o **Objeto.**

**@EJB:** Incrementa as funcionalidades do **Enterprise JavaBeans** na aplicação.

## ❖ Facade

O Facade é um padrão estrutural que atua na redução da complexidade da interface em um sistema com várias classes, bibliotecas ou frameworks, além de mover dependências indesejadas para um só local.

```
13    @Stateless
14    public class EstadoFacade extends AbstractFacade<Estado> {
15
16        @PersistenceContext(unitName = "ERPWEB2023PU")
17        private EntityManager em;
18
19        @Override
          protected EntityManager getEntityManager() {
21            return em;
22        }
23
24        public EstadoFacade() {
25            super( entityClass: Estado.class);
26        }
27    }
```

## ❖ Método Construtor

Os Métodos Construtores são responsáveis por criar o **Objeto** em memória, instanciando a classe que foi definida. Ele deve possuir o mesmo nome da classe, correspondendo às letras maiúsculas e minúsculas.

```
18    @Stateless
19    public class CidadeFacade extends AbstractFacade<Cidade> {
20
21        public CidadeFacade() {
22        super( entityClass: Cidade.class);
23    }
```

## ❖ Converter

O Converter permite a transformação de **Objeto** em **String** e a transformação de **String** em **Objeto**.

**Objeto** para **String** → **(getAsString)**

**String** para **Objeto** → **(getAsObject)**

## ❖ Uso das Classes (Herança)

Para usar a Classe tem é necessário fazer um Filho **(Facade)**, e utilizar o **getEntityManager**, sendo capaz de gerenciar adequadamente os métodos e operações relacionados às entidades Pai e Filhas em um relacionamento de herança, independentemente da estratégia de mapeamento adotada.

Faz se uso de **super** para chamar o construtor da Classe Pai.

**Ex:**

```
public CidadeFacade() {
    super( entityClass: Cidade.class);
}
```