Programação para Web II
Aula 8

Profa. Silvia Bertagnolli

Anotações

Transient

@Transient

Por definição, todos os atributos da classe Java são mapeados no banco de dados para serem persistidos

Caso algum atributo não deva ser persistido, utilize a anotação **@Transient** na declaração do atributo

```
@Transient

private String atributoTransiente;
```

Enumerated

@Enumerated

@Enumerated que permite associar valores de uma enumeração Java (enum) a um atributo

Este tipo de anotação é bem útil para mapear atributos que têm um conjunto restrito de valores que podem ser especificados em um enum, como é o caso dos dias da semana

@Enumerated

```
public enum DiaDaSemana {
      SEGUNDA, TERÇA, QUARTA, QUINTA, SEXTA
};

public class Consulta{

  @Enumerated(EnumType.STRING)
  private DiaDaSemana diaDaSemana;
  private Integer horaInicio;

  ...
}
```

Usando Enum (String)

@Enumerated(EnumType.STRING)
Grava o valor da constante da enum no banco de dados

Exemplo: MATRICULADO, CANCELADO

Vantagem: dá para trocar a ordem dos enums a qualquer momento e sua aplicação continuará funcionando

Desvantagem: não dá para alterar os nomes das constantes da enum

Usando Enum (Ordinal)

@Enumerated(EnumType.ORDINAL)
Grava a ordem do enum no banco de dados

Exemplo: 1, MATRICULADO; 2, CANCELADO

Vantagem: possibilita renomear as constantes da enum em qualquer momento

Desvantagem: não é possível mudar a ordem das constantes da enum

Temporal

@Temporal

Quando são usadas as classes “java.util.Date” e “java.util.Calendar” deve-se explicitamente utilizar a anotação @Temporal

Essa anotação pode ser usada para Data, Hora ou Data e Hora

@Temporal

```
//Somente Data
@Temporal(TemporalType.DATE)
private Date dataPublicacao;

// Somente hora
@Temporal(TemporalType.TIME)
private Date horaAtual;

// Data e hora
@Temporal(TemporalType.TIMESTAMP)
private java.util.Date dataEHora;
```

Usando datas

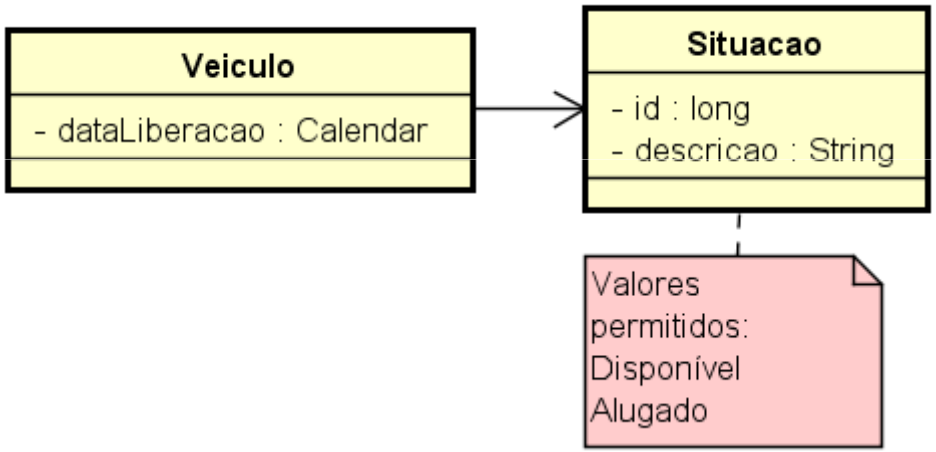
```
@Temporal(TemporalType.DATE)
private Calendar dataIngresso;
```

OU:

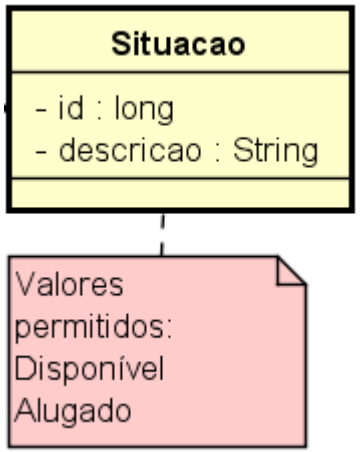
```
@Temporal(TemporalType.DATE)
private java.util.Date dataIngresso;
```

Exemplos

Usando enum e datas



Criando a enum




```
public enum Situacao {
    DISPONIVEL("Disponível"),
    ALUGADO("Alugado");
    private String descricao;
    private Situacao(String descricao) {
        this.descricao = descricao;
    }
    public String getDescricao() {  return descricao; }
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }
}
```

Usando enum e datas

```
@Entity
public class Veiculo implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Enumerated(EnumType.ORDINAL)
    private Situacao estado;

    @Temporal(TemporalType.DATE)
    private Calendar dataLiberacao;
```

Usando enum e datas

```
//get/set; equals(); hashCode
@Override
public String toString() {
  String aux = "";
  if(dataLiberacao != null){
      java.text.SimpleDateFormat sdf =
                              new SimpleDateFormat("dd/MM/yyyy");
      aux = sdf.format(dataLiberacao.getTime());
  }
  return "Veiculo{" + "id=" + id + ", estado=" +
                    estado + ", dataLiberacao=" + aux + '}';
}
```

Usando enum e datas

```
//SALVANDO NO BD

EntityManager em = JPAUtil.getEntityManager();
em.getTransaction().begin();
Veiculo v = new Veiculo();
v.setEstado(Situacao.ALUGADO);
v.setDataLiberacao(Calendar.getInstance());
em.persist(v);
em.getTransaction().commit();
em.close();
```

Usando enum e datas

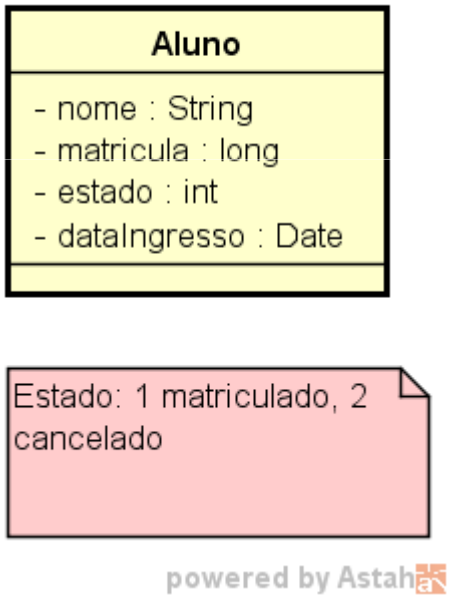
```
//BUSCANDO DO BD

EntityManager em = JPAUtil.getEntityManager();
em.getTransaction().begin();
v = em.find(Veiculo.class, 1L);
Calendar date = v.getDataLiberacao();

System.out.println("Objeto recuperado:" + v.toString());
em.getTransaction().commit();
em.close();
```

EXERCÍCIOS

1) Crie a entidade Aluno abaixo:



EXERCÍCIOS

Observe os seguintes itens:

- Estado deve ser mapeado como uma enumeração
- DataIngresso deve ser mapeado usando TEMPORAL

JPQL

JPQL

A Java Persistence Query Language (JPQL) é a maneira mais conhecida e utilizada de consultar objetos na JPA

O foco da JPQL não está relacionado a tabelas e colunas, mas sim a objetos e seus atributos

O problema é decidir onde armazenar as consultas:

- consultas dinâmicas, normalmente criadas dentro de classes DAO
- consultas estáticas através de *Named Queries*

Consulta Dinâmica

```
SELECT * FROM Pessoa WHERE id>10
```

É equivalente em JPQL a:

```
SELECT p FROM Pessoa p WHERE p.id_pessoa>10
```

Operadores

```
>=, <=, >, <

<>

NOT, BETWEEN, LIKE

IN, IS NULL, IS EMPTY

MEMBER [OF]

EXISTS
```

Consulta dinâmica

```
TypedQuery<Pessoa> query = em.createQuery("SELECT obj FROM
Pessoa obj", Pessoa.class);

List<Pessoa> pessoas= query.getResultList();

for (Pessoa p: pessoas) {

	System.out.println("Pessoa:"+p.toString());

}
```

Funções que podem ser aplicadas nas consultas

CONCAT(String , String) – recebe uma lista com suas ou mais Strings e devolve uma só concatenando as duas listas

SUBSTRING (String , int start [, int length]) – recebe um texto e retorna só a parte solicitada

Funções que podem ser aplicadas nas consultas

```
TRIM([[LEADING | TRAILING | BOTH] [char] FROM] String)
```

```
TRIM(' UM TEXTO ') -> 'UM TEXTO'

TRIM(LEADING FROM ' UM TEXTO ') -> 'UM TEXTO '

TRIM(TRAILING FROM ' UM TEXTO ') -> ' UM TEXTO'

TRIM(BOTH FROM ' UM TEXTO ') -> 'UM TEXTO'

TRIM(LEADING 'A' FROM 'ARARA') -> 'RARA'

TRIM(TRAILING 'A' FROM 'ARARA') -> 'ARAR'

TRIM('A' FROM 'ARARA') -> 'RAR'
```

Funções que podem ser aplicadas nas consultas

LOWER (String) – devolve o texto em minúsculo

UPPER(String) – devolve o texto em maiúsculo

LENGTH(String) – devolve o comprimento do texto

LOCATE(String original, String substring [, int start]) - devolve a posição em que a substring e encontrada na original, e opcionalmente ha um terceiro parâmetro indicando em qual posição a busca deve começar

Exemplo 2

```
String sql = "SELECT obj FROM Pessoa obj where
upper(obj.nome) like '%" + pessoa.getNome().toUpperCase()
+"'" ;
```

Funções que podem ser aplicadas nas consultas

ABS(int): devolve o valor absoluto (sem o sinal) de um número

SQRT(int): Devolve a raiz quadrada de um número

MOD(int, int): Retorna o resto da divisão do primeiro número pelo segundo

SIZE(collection): Retorna o tamanho de uma coleção

INDEX(obj): Retorna a posição de um determinado elemento quando ele estiver em uma lista ordenada

Funções que podem ser aplicadas nas consultas

AVG(property): Devolve a media de valores numéricos

MAX(property): Devolve o valor máximo entre valores comparáveis (números, datas, strings)

MIN(property): Devolve o valor mínimo entre valores comparáveis (números,

datas, strings)

SUM(property): Devolve a soma de valores numericos

COUNT(property): Devolve a quantidade de elementos

Consulta Dinâmica

Consulta dinâmica com parâmetros

```
EntityManager em = JPAUtil.getEntityManager();
Pessoa p = new Pessoa();
TypedQuery<Pessoa> query = em.createQuery(
       "SELECT p FROM Pessoa p"
       + " where p.nome = :nome"
       + " and p.id = :id", Pessoa.class);
query.setParameter("nome", "Fulano");
query.setParameter("id", 3L);
List<Pessoa> pessoas = query.getResultList();
for (Pessoa pessoa : pessoas) {
     System.out.println("Pessoa:"+pessoa.toString());
}
em.close();
```

Consulta Estática

Consulta Estática

Uma consulta estática é chamada de N*amed Query*

Os seus parâmetros já estão pré-definidos

Todas as consultas desse tipo são declaradas na entidade através da anotação @NamedQuery

Consulta Estática

Por exemplo, para buscar uma categoria que tem um determinado nome podemos definir a classe como segue:

```
@Entity

@Table(name="categoria")

@NamedQuery(name="Categoria.buscarCategoriaPorNome",

    query="select c from Categoria c where c.nome = :nome")

public class Categoria implements Serializable {

…
```

Consulta Estática

```
@Entity
@Table(name="categoria")
@NamedQuery(name="Categoria.buscarCategoriaPorNome",
    query="select c from Categoria c where c.nome = :nome")
public class Categoria implements Serializable {
…
public Categoria buscarCategoriaPorNome(String nome) {
    EntityManager em = JPAUtil.getEntityManager();
    return em.createNamedQuery(
       "Categoria.buscarCategoriaPorNome", Categoria.class)
        .setParameter("nome", nome)
        .getSingleResult();
}
}
```

Consulta Estática

```
public class Teste_JPQL_4 {
  public static void main(String[] args) {
       //...
    Categoria c = new Categoria().buscarCategoriaPorNome("teste");
    System.out.println("Categoria lida:" + c.toString());
  }
}
```

https://www.tutorialspoint.com/jpa/jpa_jpql.htm

EXERCÍCIOS

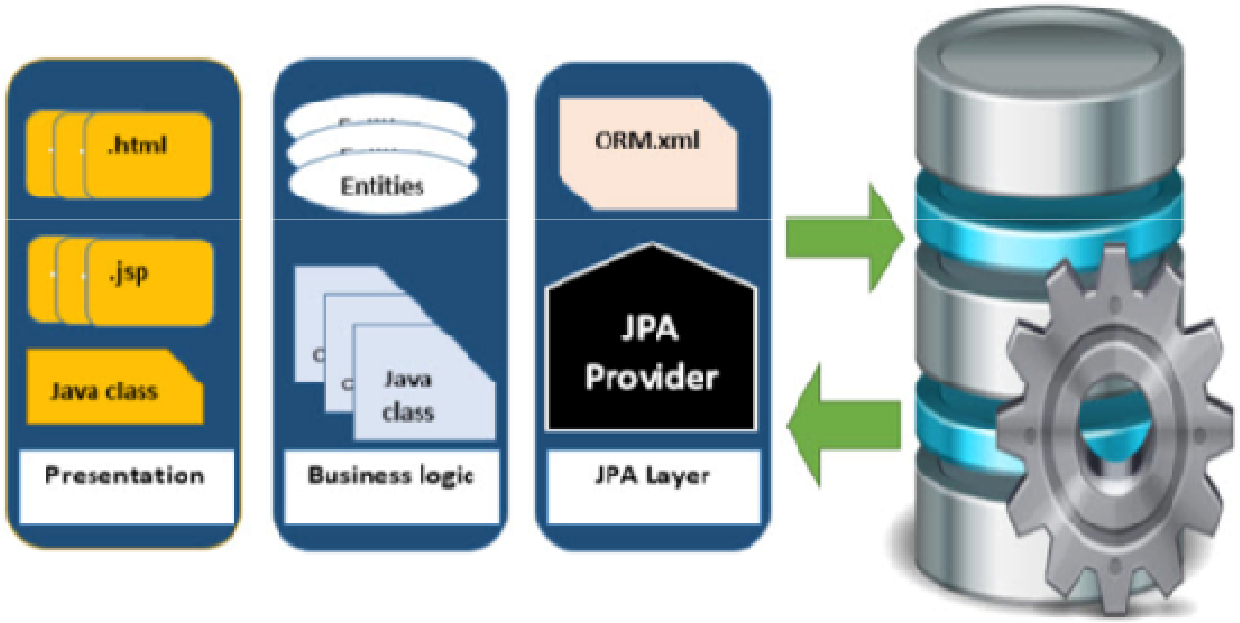
1. Crie a classe de Entidade Mensagem (como POJO) com os atributos id e mensagem

2. Agora, inclua vários objetos no BD usando os códigos dos exemplos da aula de hoje

3. Faça uma consulta dinâmica, que retorna todas as mensagens que possuem dentro da mensagem a palavra Anexo ou anexo ou ANEXO

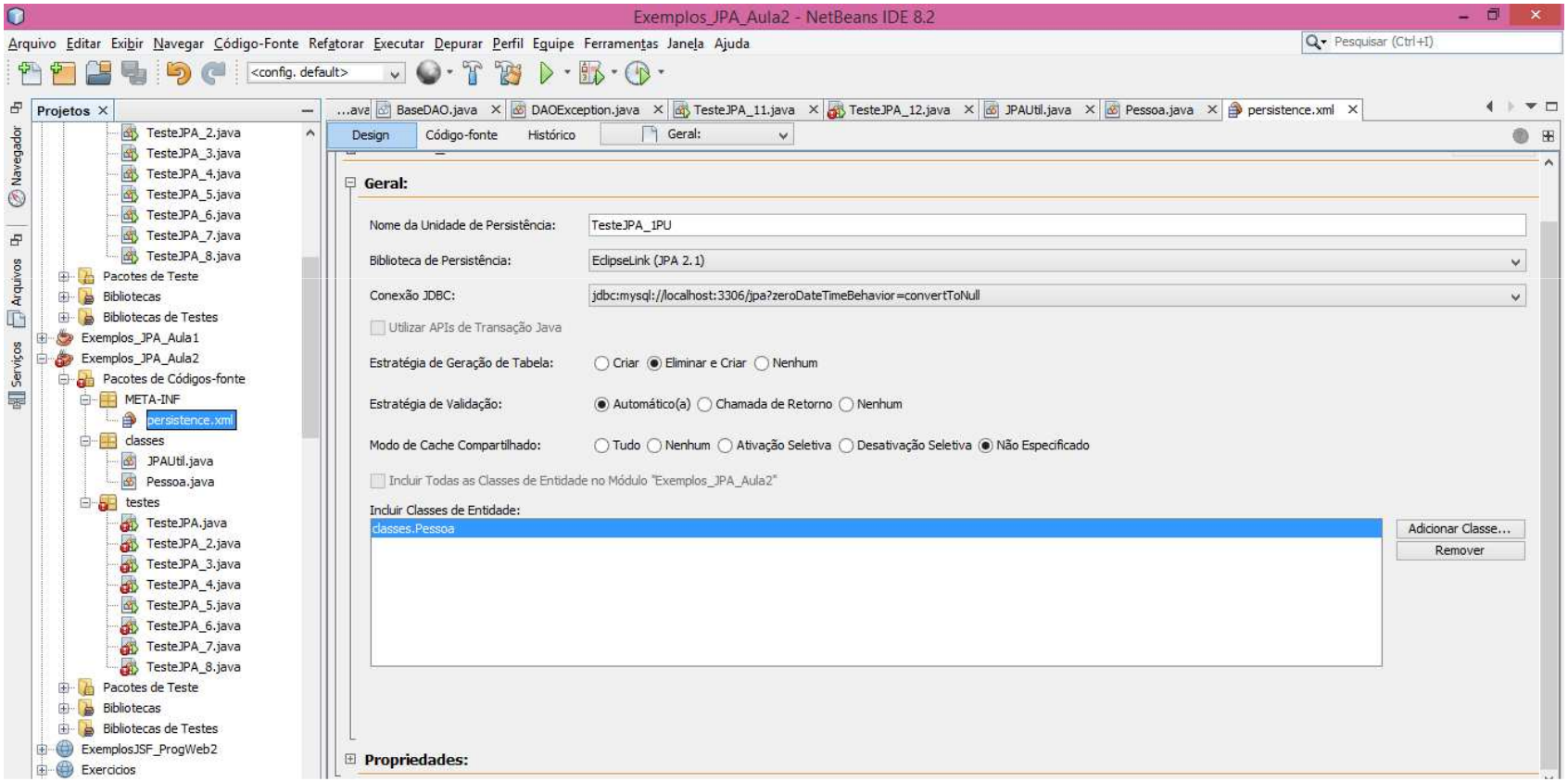
JPA
x
Camadas

MVC com JPA



Fonte = http://www.w3ii.com/pt/jpa/jpa_introduction.html

Passo 1 – Criar a unidade de persistência



Passo 2 – Crie a Entidade

```
@Entity
@Table(name="pessoa")
public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name="id_pessoa")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private int idade;

    //...
}
```

Passo 3 – Teste a Entidade

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("TesteJPA_1PU");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Pessoa p = new Pessoa();
em.persist(p);
em.getTransaction().commit();
System.out.println("Pessoa cadastrada com sucesso -
                                          Id ="+p.getId());
em.close();
emf.close();
```

Passo 4 – Crie a Classe DAO

O padrão de projeto DAO tem como objetivo evitar a exposição da camada de persistência para outras camadas.

O DAO isola as operações JDBC, como criação, recuperação, atualização e exclusão, para um objeto de negócio

Vamos analisar a classe PessoaDAO e verificar como ela se comporta?

PessoaDAO - salvar

```
public void salvar(Pessoa pessoa) {
        em = JPAUtil.getEntityManager();
        em.getTransaction().begin();
        em.persist(pessoa);
        em.getTransaction().commit();
        em.close();
}
```

PessoaDAO - atualizar

```
public void atualizar(Pessoa pessoa) {
        em = JPAUtil.getEntityManager();
        em.getTransaction().begin();
        em.merge(pessoa);
        em.getTransaction().commit();
        em.close();
}
```

PessoaDAO - remover

```
public void remover(long id) {
      em = JPAUtil.getEntityManager();
      em.getTransaction().begin();
      Pessoa entity = em.find(Pessoa.class, id);
      if (entity != null) { em.remove(entity);
      } else {
         throw new DAOException("Não existe o id: " + id);
      }
      em.getTransaction().commit();
      em.close();   }
```

PessoaDAO– buscar (por id)

```
public Pessoa buscar(long id) {

        em = JPAUtil.getEntityManager();

        Pessoa pessoa= em.find(Pessoa.class, id);

        em.close();

        return pessoa;

}
```

PessoaDAO– buscar (por nome)

```
public List<Pessoa> buscar(String nome) {
    em = JPAUtil.getEntityManager();
    TypedQuery<Pessoa> query = em.createQuery(
       "SELECT p FROM Pessoa p "
        + "where lower(p.nome) like '%"
        + nome.toLowerCase() + "%'", Pessoa.class);
        List<Pessoa> pessoas= query.getResultList();
        em.close();
        return pessoas;
}
```

PessoaDAO - buscarTodos

```
public List<Pessoa> buscarTodos() {
        em = JPAUtil.getEntityManager();
        TypedQuery<Pessoa> query
                = em.createQuery(
                        "SELECT p FROM Pessoa p",
                         Pessoa.class);
        List<Pessoa> pessoas = query.getResultList();
        em.close();
        return pessoas;
    }
```

Testes

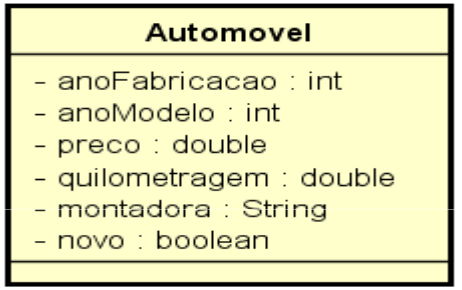
```
PessoaDAO objDAO = new PessoaDAO();

//Cria uma nova instância de usuário e salva
objDAO.salvar(new Pessoa("Beltrano", 50));
System.out.print("Pessoa com nome Beltrano foi salva!!!");


System.out.println("\nLISTAR TODOS");
for (Pessoa p : objDAO.buscarTodos())
     System.out.printf(p.toString());
```

EXERCÍCIOS

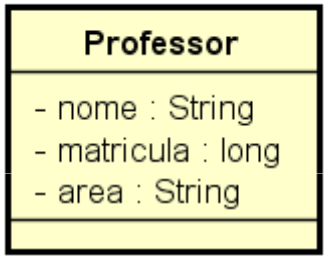
1) Crie a classe AutomovelDAO, usando como base a classe Automovel abaixo:



2) Monte uma classe de testes para verificar se as ações com o Automovel estão funcionando corretamente

EXERCÍCIOS

1) Crie a classe Professor e ProfessorDAO, usando o esquema abaixo:



2) Na classe ProfessorDAO crie os métodos:

pesquisarPorNome(),
pesquisarPorMatricula(),
pesquisarPorArea()

EXERCÍCIOS

3) Monte uma classe de testes para verificar se as ações com os objetos Professor e ProfessorDAO estão funcionando corretamente