Persistência com JDBC e JPA

Aula 4

Mapeamento Objeto-Relacional com

Java Persistence API

Marcos Alberto Lopes da Silva

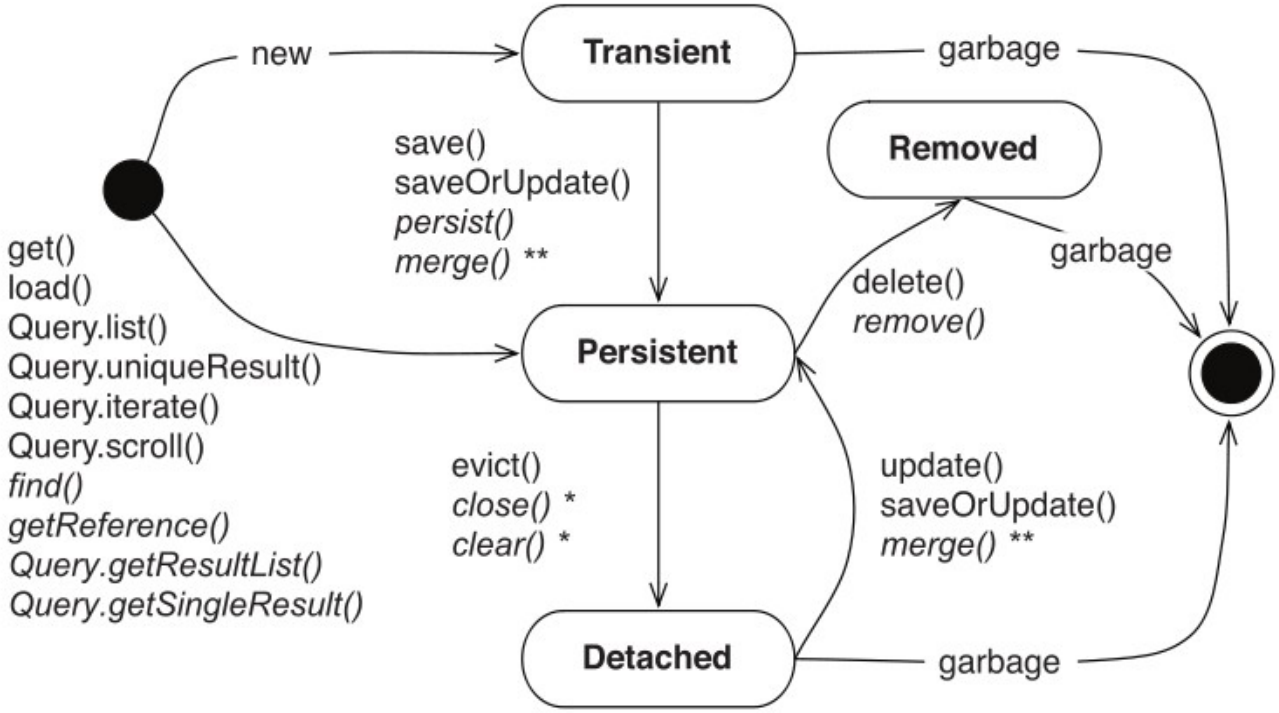
(malopes21@gmail.com)

Sumário

- JPA – Estados dos objetos
- JPA – Mais conceitos (alguns são revisões)
- Mapeamento de Herança
- Single Table, Joined, Table Per Class
- Classes Embutidas
- Mapeamentos (revisão)
- Multiplicidade (revisão)
- Cascade
- Exercício final

Estados dos objetos

- Transientes;
- Persistentes;
- Removidos;
- Desligados



Estados dos objetos

- Transientes (qualquer modificação em uma instância transiente não é sabida pelo Hibernate);
- Persistentes (sempre estão associadas a um contexto de persistência. Estão sob um cache no Hibernate, que pode detectar alterações);
- Removidos (são gerenciados pelo contexto de persistência, consequentemente deletados quando a unidade de trabalho terminar);
- Desligados (o estado não está mais sincronizado com o estado do banco). Podem voltar a ser persistentes com fundição (merge)

Conceitos

- EntityManager → fila de declarações SQL que precisam ser sincronizadas com o banco em algum ponto (assíncrono);
- EntityTransaction → define os limites da transação. Pode ser programado ou não (JTA? RESOURCE_LOCAL?);
- Dialeto → classe que informa quais variações SQL ele deve gerar para conexão com o banco;
- EntityManagerFactory – deve ser instanciada uma única vez na aplicação (ThreadSafe)

Conceitos

- Mapeamentos como @Table, @Column e @JoinColumn são opcionais;
- XML possui precedência sobre anotações;
- Pode ser necessário o uso de recursos específicos do motor ORM

ex:

```
EntityManager em;

HibernateEntityManager hib =
    (HibernateEntityManager)em;

Session s = hib.getSession();
```

Conceitos

- Exemplos de metadados do fornecedor que não existem na JPA:

  @org.hibernate.annotations.BatchSize

  @org.hibernate.annotations.Formula
- Jpa está automaticamente em um contexto Java EE;
- Modelos de domínio não devem consultar o banco (persistência transparente);
- Anotações devem ser colocadas nos campos ou nos métodos de acesso?;
- Atributos @Transient – são ignorados pelo contexto de persistência

Mapeamento de Herança

A especificação JPA define três estratégias para realizar o mapeamento de herança:

- Single Table
- Joined
- Table Per Class

Herança – Single Table

A estratégia Single Table é a mais comum e a que possibilita melhor desempenho em relação a velocidade das consultas.

Nessa estratégia, a super classe deve ser anotada com

```
@Inheritance(strategy=InheritanceType.SINGLE_TABLE).
```

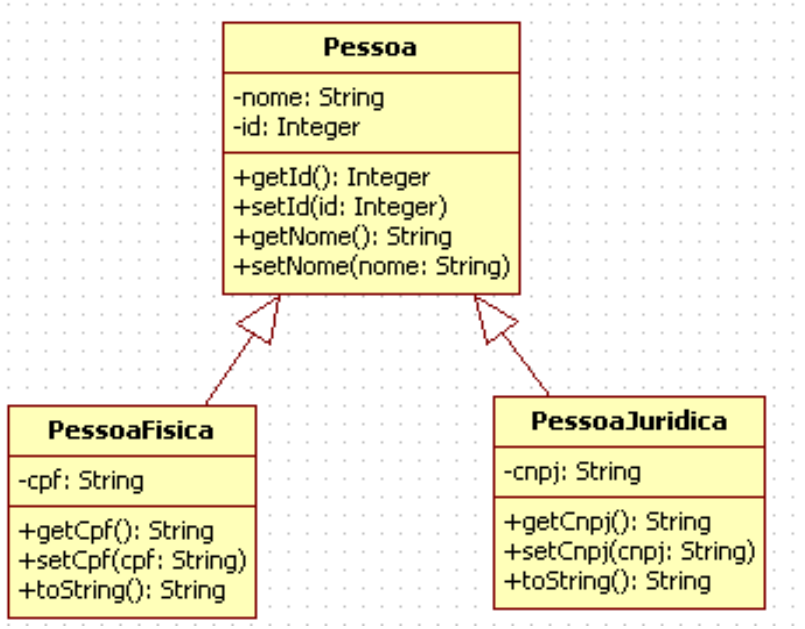
O provedor JPA criará apenas uma tabela com o nome da super classe.

Todos os atributos da super classe e os das sub classes serão mapeados para colunas dessa tabela.

Além disso, uma coluna especial chamada DTYPE será utilizada para identificar a classe do objeto/registro.

Herança – Single Table

```
mysql> describe pessoa;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| ID    | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| DTYPE | varchar(31)  | YES  |     | NULL    |                |
| NOME  | varchar(255) | YES  |     | NULL    |                |
| CNPJ  | varchar(255) | YES  |     | NULL    |                |
| CPF   | varchar(255) | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

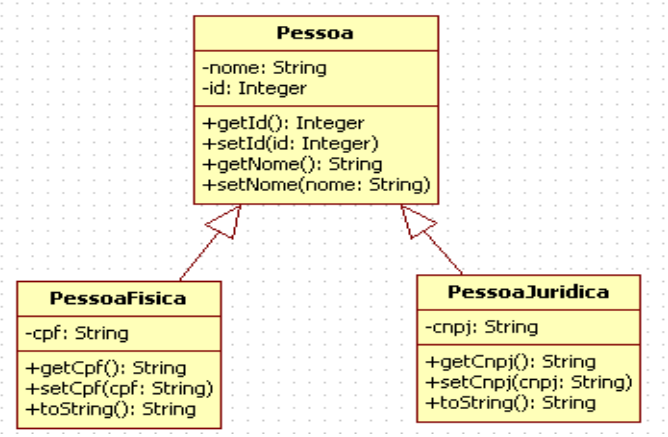



```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String nome;
}
```

Herança – Single Table

```
@Entity
public class PessoaJuridica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cnpj;
    public String getCnpj() {
        return cnpj;
    }
    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```


```
@Entity
public class PessoaFisica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cpf;

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
}
```



Herança – Single Table

```
private static void insertPF() {
     EntityManager manager = JpaUtil.getManager();
     PessoaFisica pessoaFisica = new PessoaFisica();
     pessoaFisica.setNome("Fulano"); pessoaFisica.setCpf("111.222.333-44");
     manager.persist(pessoaFisica);
     manager.getTransaction().begin();
     manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}
```

```
mysql> select * from Pessoa;
+----+----------------+---------+--------------------+-----------
| ID | DTYPE          | NOME    | CNPJ               | CPF
+----+----------------+---------+--------------------+-----------
|  1 | PessoaFisica   | Fulano  | NULL               | 111.222.3
|  2 | PessoaJuridica | Empresa | 11.444.777/0001-61 | NULL
|  3 | Pessoa         | Bla     | NULL               | NULL
+----+----------------+---------+--------------------+-----------
3 rows in set (0.00 sec)
```

```
private static void insertPJ() {
     EntityManager manager = JpaUtil.getManager();
     PessoaJuridica pessoaJuridica = new PessoaJuridica();
     pessoaJuridica.setNome("Empresa");
     pessoaJuridica.setCnpj("11.444.777/0001-61");
     manager.persist(pessoaJuridica);
     manager.getTransaction().begin();
     manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}

private static void insert() {
     EntityManager manager = JpaUtil.getManager();
     Pessoa pessoa = new Pessoa(); pessoa.setNome("Bla");
     manager.persist(pessoa);
     manager.getTransaction().begin();
     manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}
```

Insere Pessoa!
Pode-se definir como classe abstrata dependendo do modelo necessário.

Herança – Single Table

Recuperando objetos:

```
private static void findPessoa() {
    EntityManager manager = JpaUtil.getManager();
    Pessoa pessoa = manager.find(Pessoa.class, 3L);
    PessoaFisica pessoaFisica = manager.find(PessoaFisica.class, 1L);
    PessoaJuridica pessoaJuridica = manager.find(PessoaJuridica.class, 2L);
    System.out.println("OK!! "+pessoa);
    System.out.println("OK!! "+pessoaFisica);
    System.out.println("OK!! "+pessoaJuridica);
    JpaUtil.closeManager(manager);
}
```

```
[EL Finest]: connection: 2015-02-27 12:09:33.563--ServerSess
OK!! Pessoa [id=3, nome=Bla]
OK!! PessoaFisica [id= 1, cpf=111.222.333-44]
OK!! PessoaJuridica [id= 2, cnpj=11.444.777/0001-61]
[EL Finer]: transaction: 2015-02-27 12:09:33.564--UnitOfWork
```

Herança – Joined

Na estratégia Joined, uma tabela para cada classe da hierarquia é criada.

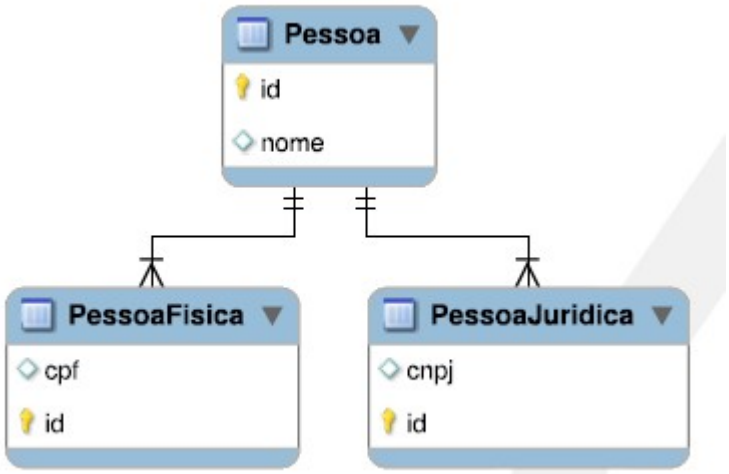
Em cada tabela, apenas os campos referentes aos atributos da classe correspondente são adicionados.

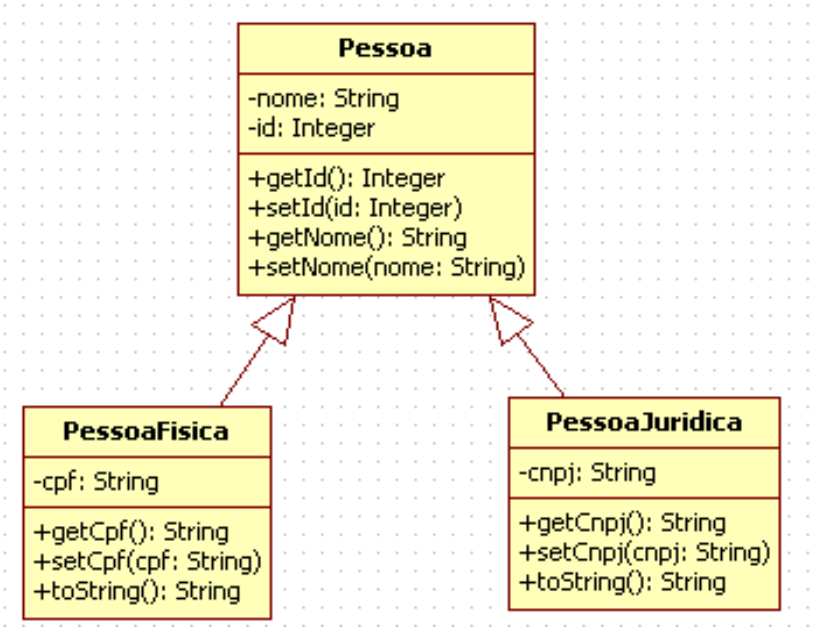
Para relacionar os registros das diversas tabelas e remontar os objetos numa consulta, as tabelas das sub-classes possuem chaves estrangeiras vinculadas à tabela da super-classe.

O consumo de espaço utilizando a estratégia Joined é menor do que o utilizado pela estratégia Single Table.

Contudo, as consultas são mais lentas, pois é necessário realizar operações de join para recuperar os objetos.

Herança – Joined

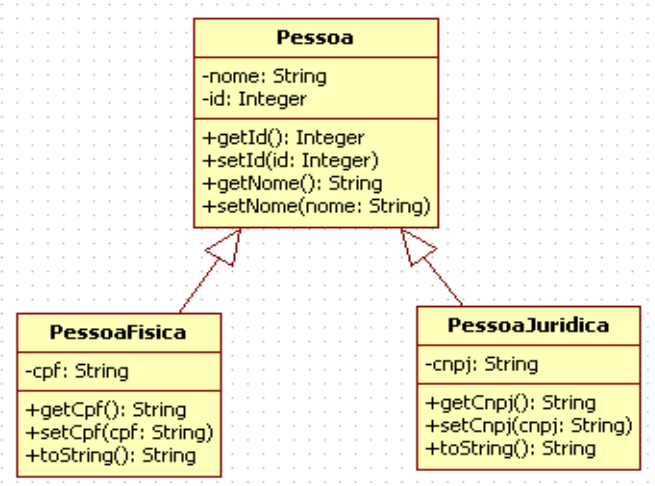




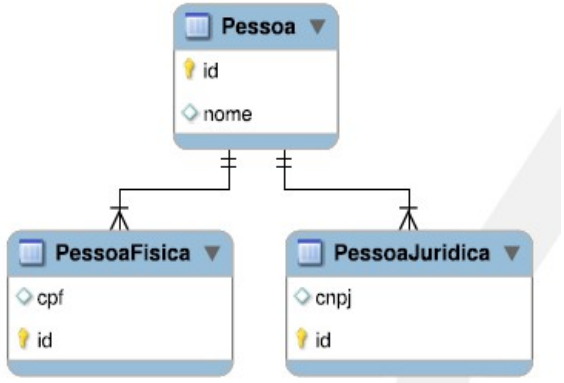

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String nome;
}
```

Herança – Joined

```
@Entity
public class PessoaJuridica extends Pessoa implements Ser

    private static final long serialVersionUID = 1L;
    private String cnpj;
    public String getCnpj() {
        return cnpj;
    }
    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```

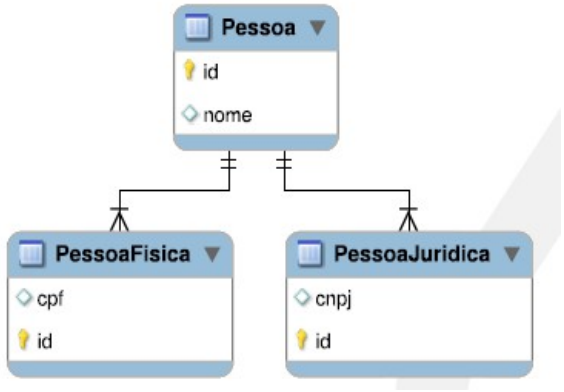



```
@Entity
public class PessoaFisica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cpf;

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
}
```



Herança – Joined

```
mysql> describe pessoa;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| ID     | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| DTYPE  | varchar(31)  | YES  |     | NULL    |                |
| NOME   | varchar(255) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)

mysql> describe pessoafisica;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| ID    | bigint(20)   | NO   | PRI | NULL    |       |
| CPF   | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> describe pessoajuridica;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| ID    | bigint(20)   | NO   | PRI | NULL    |       |
| CNPJ  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```



Mesmo teste do exemplo SINGLE_TABLE

```
mysql> select * from pessoa;
+----+----------------+----------+
| ID | DTYPE          | NOME     |
+----+----------------+----------+
|  1 | PessoaFisica   | Fulano   |
|  2 | PessoaJuridica | Empresa  |
|  3 | Pessoa         | Bla      |
+----+----------------+----------+
3 rows in set (0.00 sec)

mysql> select * from pessoafisica;
+----+----------------+
| ID | CPF            |
+----+----------------+
|  1 | 111.222.333-44 |
+----+----------------+
1 row in set (0.00 sec)

mysql> select * from pessoajuridica;
+----+--------------------+
| ID | CNPJ               |
+----+--------------------+
|  2 | 11.444.777/0001-61 |
+----+--------------------+
1 row in set (0.00 sec)
```

Herança – Joined

Recuperando objetos:

```
private static void findPessoa() {
    EntityManager manager = JpaUtil.getManager();
    Pessoa pessoa = manager.find(Pessoa.class, 3L);
    PessoaFisica pessoaFisica = manager.find(PessoaFisica.class, 1L);
    PessoaJuridica pessoaJuridica = manager.find(PessoaJuridica.class, 2L);
    System.out.println("OK!! "+pessoa);
    System.out.println("OK!! "+pessoaFisica);
    System.out.println("OK!! "+pessoaJuridica);
    JpaUtil.closeManager(manager);
}
```

```
[EL Finest]: connection: 2015-02-27 12:09:33.563--ServerSess
OK!! Pessoa [id=3, nome=Bla]
OK!! PessoaFisica [id= 1, cpf=111.222.333-44]
OK!! PessoaJuridica [id= 2, cnpj=11.444.777/0001-61]
[EL Finer]: transaction: 2015-02-27 12:09:33.564--UnitOfWork
```

Herança – Table Per Class

Na estratégia Table Per Class, uma tabela para cada classe concreta da hierarquia é criada.

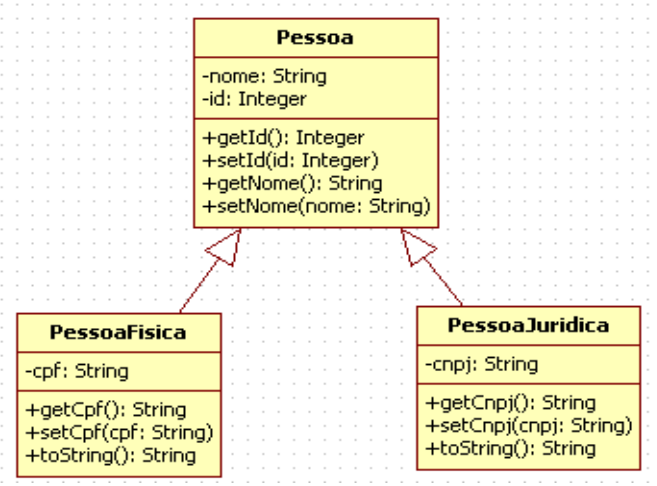
Contudo, os dados de um objeto não são colocados em tabelas diferentes.

Dessa forma, para remontar um objeto não é necessário realizar operações de join.

A desvantagem desse modo é que não existe um vínculo explícito no banco de dados entres as tabelas correspondentes às classes da hierarquia.

Nessa estratégia, não podemos utilizar a geração automática de chave primárias simples e numéricas. Tente!!!

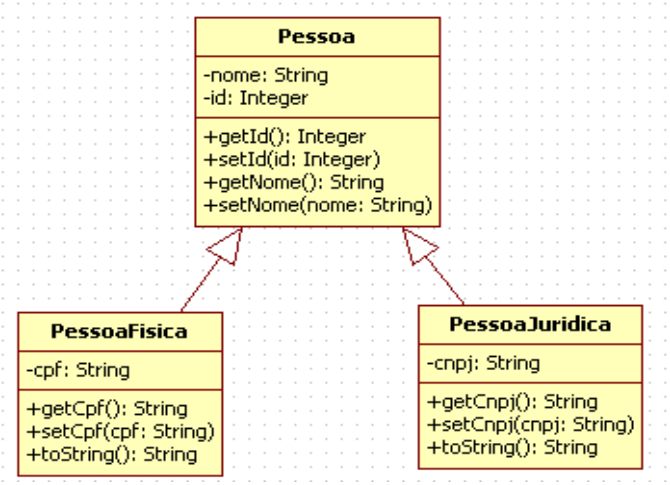
Herança – Table Per Class



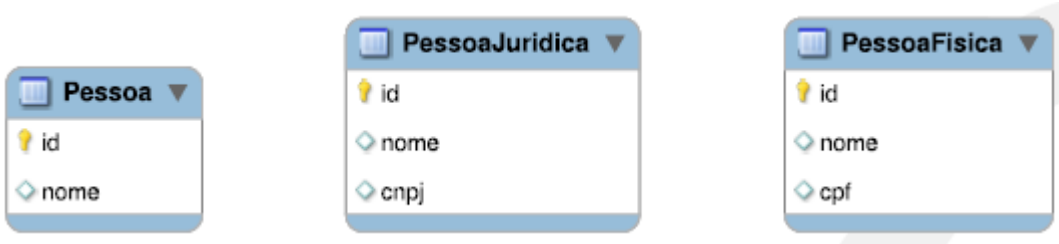

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    //@GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String nome;
}
```

Herança – Table Per Class

```
@Entity
public class PessoaJuridica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cnpj;
    public String getCnpj() {
        return cnpj;
    }
    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```




```
@Entity
public class PessoaFisica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cpf;

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
}
```

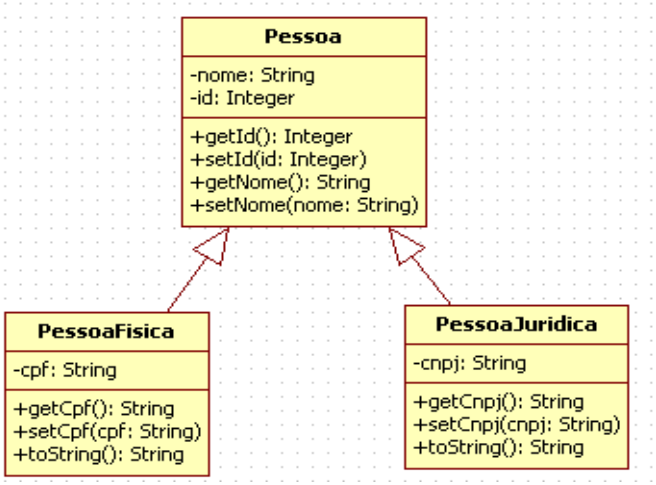


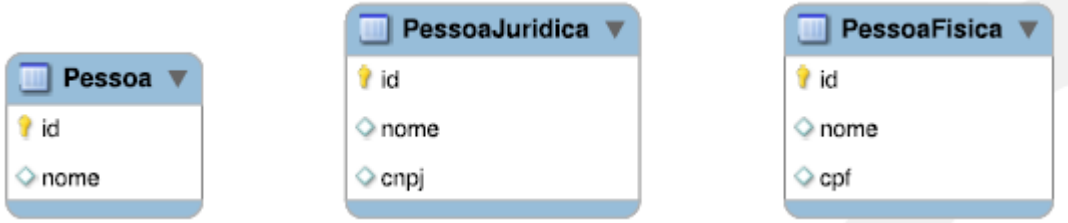
Herança – Table Per Class

```
private static void insertPF() {
     EntityManager manager = JpaUtil.getManager();
     PessoaFisica pessoaFisica = new PessoaFisica();
     pessoaFisica.setId(1L);
     pessoaFisica.setNome("Fulano");   pessoaFisica.setCpf("111.222.333-44");
     manager.persist(pessoaFisica);
     manager.getTransaction().begin(); manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}

private static void insertPJ() {
     EntityManager manager = JpaUtil.getManager();
     PessoaJuridica pessoaJuridica = new PessoaJuridica();
     pessoaJuridica.setId(2L);
     pessoaJuridica.setNome("Empresa"); pessoaJuridica.setCnpj("11.444.777/0001-61");
     manager.persist(pessoaJuridica);
     manager.getTransaction().begin(); manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}

private static void insert() {
     EntityManager manager = JpaUtil.getManager();
     Pessoa pessoa = new Pessoa();
     pessoa.setId(3L); pessoa.setNome("Bla");
     manager.persist(pessoa);
     manager.getTransaction().begin(); manager.getTransaction().commit();
     JpaUtil.closeManager(manager);
}
```

Herança – Table Per Class

```
mysql> describe pessoa;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| ID    | bigint(20)   | NO   | PRI | NULL    |       |
| NOME  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> describe pessoafisica;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| ID    | bigint(20)   | NO   | PRI | NULL    |       |
| CPF   | varchar(255) | YES  |     | NULL    |       |
| NOME  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> describe pessoajuridica;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| ID    | bigint(20)   | NO   | PRI | NULL    |       |
| CNPJ  | varchar(255) | YES  |     | NULL    |       |
| NOME  | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```



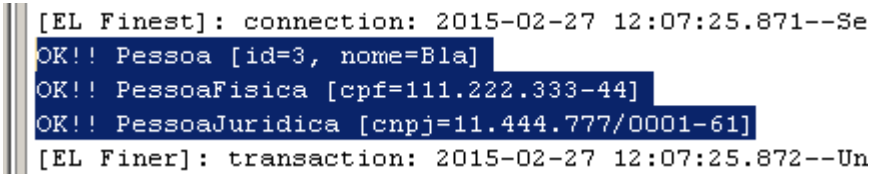



```
mysql> select * from pessoa;
+----+------+
| ID | NOME |
+----+------+
|  3 | Bla  |
+----+------+
1 row in set (0.00 sec)

mysql> select * from pessoafisica;
+----+----------------+--------+
| ID | CPF            | NOME   |
+----+----------------+--------+
|  1 | 111.222.333-44 | Fulano |
+----+----------------+--------+
1 row in set (0.00 sec)

mysql> select * from pessoajuridica;
+----+--------------------+---------+
| ID | CNPJ               | NOME    |
+----+--------------------+---------+
|  2 | 11.444.777/0001-61 | Empresa |
+----+--------------------+---------+
1 row in set (0.00 sec)
```

Herança – Table Per Class

Recuperando objetos:

```
private static void findPessoa() {
    EntityManager manager = JpaUtil.getManager();
    Pessoa pessoa = manager.find(Pessoa.class, 3L);
    PessoaFisica pessoaFisica = manager.find(PessoaFisica.class, 1L);
    PessoaJuridica pessoaJuridica = manager.find(PessoaJuridica.class, 2L);
    System.out.println("OK!! "+pessoa);
    System.out.println("OK!! "+pessoaFisica);
    System.out.println("OK!! "+pessoaJuridica);
    JpaUtil.closeManager(manager);
}
```



Classes embutidas

```java
@Entity
public class Pessoa implements Serializable {

 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;

 private String nome;

 //@Embedded
 private Endereco endereco;

}

@Embeddable
public class Endereco implements Serializable {

 private String pais;
 private String estado;
 private String cidade;
 private String logradouro;
 private int numero;
 private String complemento;
 private int cep;
}
```

```
mysql> describe pessoa;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| DTYPE       | varchar(31)  | YES  |     | NULL    |                |
| NOME        | varchar(255) | YES  |     | NULL    |                |
| CEP         | int(11)      | YES  |     | NULL    |                |
| CIDADE      | varchar(255) | YES  |     | NULL    |                |
| COMPLEMENTO | varchar(255) | YES  |     | NULL    |                |
| ESTADO      | varchar(255) | YES  |     | NULL    |                |
| LOGRADOURO  | varchar(255) | YES  |     | NULL    |                |
| NUMERO      | int(11)      | YES  |     | NULL    |                |
| PAIS        | varchar(255) | YES  |     | NULL    |                |
| CNPJ        | varchar(255) | YES  |     | NULL    |                |
| CPF         | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
12 rows in set (0.01 sec)
```

Classes embutidas

Suponha que a tabela Endereco esteja relacionada apenas coma tabela Pessoa.

Nesse caso, seria interessante se pudéssemos guardar os endereços das pessoas na própria tabela Pessoa, tornando desnecessária a existência da tabela Endereco.

No entanto, gostaríamos de manter as classes Pessoa e Endereco.

Na classe Endereco, devemos substituir a anotação @Entity por @Embeddable.

Mapeamentos

- Projeto Mapeamentos (classe Acao e TesteAcao);
- Mapeamento de data/hora;

Mapping type	Java type	Standard SQL built-in type
`date`	`java.util.Date` or `java.sql.Date`	`DATE`
`time`	`java.util.Date` or `java.sql.Time`	`TIME`
`timestamp`	`java.util.Date` or `java.sql.Timestamp`	`TIMESTAMP`
`calendar`	`java.util.Calendar`	`TIMESTAMP`
`calendar_date`	`java.util.Calendar`	`DATE`

Mapeamentos

- Projeto Mapeamentos (classe Anexo e TesteAnexo);
- Mapeamento para binário e valor grande;

Mapping type	Java type	Standard SQL built-in type
`binary`	`byte[]`	`VARBINARY`
`text`	`java.lang.String`	`CLOB`
`clob`	`java.sql.Clob`	`CLOB`
`blob`	`java.sql.Blob`	`BLOB`
`serializable`	Any Java class that implements `java.io.Serializable`	`VARBINARY`

Mapeamentos

- Mapeamento de enums;
- Automático;

Mapeamento entidades

- A maioria das dificuldades envolvidas na implementação de uma solução ORM está relacionada ao gerenciamento da associação;
- O ciclo de vida as entidades são independentes;
- Associações do JPA são por natureza unidirecionais (se alterar de um lado, não necessariamente o outro lado será alterado)

Multiplicidade

- Muitos para um @ManyToOne;
- Um para muitos @OneToMany;
- Um para um @OneToOne;
- Muitos para muitos @ManyToMany

Multiplicidade

- Mapeamentos bidirecionais são opcionais. Obviamente, sem o uso deles, querys são inevitáveis;
- Mapeamentos bidirecionais são uma característica, não uma necessidade;
- Em relacionamentos bidirecionais, é bom usar mappedBy para informar a JPA que a coleção é uma imagem da associação do outro lado;
- Com mappedBy, a JPA é ”informado” sobre qual a extremidade não deve sincronizar com o DB

Multiplicidade – cascatear o estado do objeto

- Cascatear = um tomar conta do outro (tratamento automático);
- Novos objetos são transientes e é necessário que se tornem persistentes para guardar no banco;
- Opções:

  Cuidar das instâncias independentes;

  Persistência transitiva (cascades). Pode-se escolher quais operações deseja-se que sejam transitivas;

Multiplicidade – cascatear o estado do objeto

- Cascata é para economizar linhas de código necessárias para gerenciar o outro lado (estado do objeto transitivo)
- Opções cascade JPA:

  PERSIST

  MERGE

  REMOVE

  REFRESH

  DETACH

  ALL (todos os mencionados acima)

Exercícios

- Implementar todos exemplos desenvolvidos em aula com casos de testes para operações CRUD sobre as entidades envolvidas;
- Implementa um app. Swing java para cadastro e leitura de dados de pessoa física e pessoa jurídica, Usar herança no modelo de dados OO e escolher alguma estratégia para o mapeamento.

Referências bibliográficas

[1] Bauer, Christian e King, Gavin – Java persistence com Hibernate. Rio de Janeiro, Ed. Ciência Moderna, 2007;