

## Object-Relational Mapping (ORM)

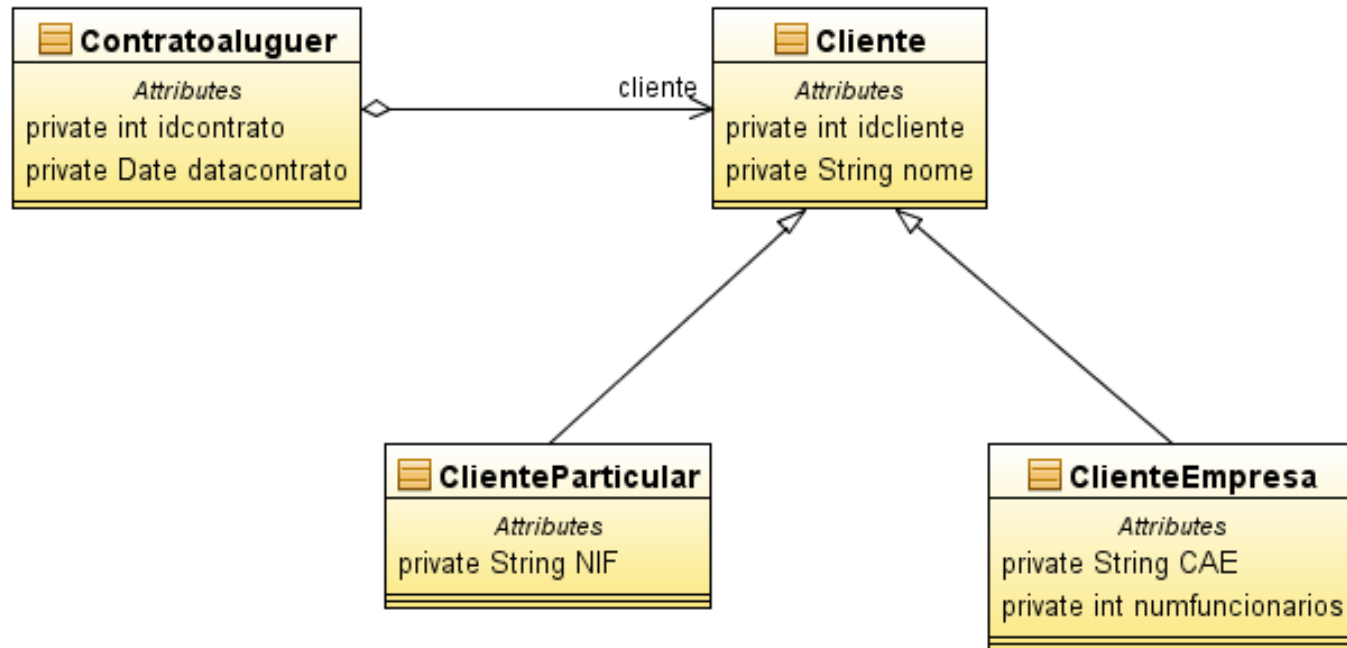
Mapeamento de Herança  
entre classes

# Mapeamento de Herança entre classes

- **Bases de dados relacionais:**
  - não têm o conceito de herança – “*is a*”;
  - não permitem associações e interrogações polimórficas.
- **Modos de representação de hierarquia de herança no modelo relacional:**
  - **SINGLE\_TABLE**
    - Uma tabela para toda a hierarquia de classes;
    - polimorfismo é obtido pela “desnormalização” do modelo relacional e introdução de um campo (“discriminator”) para definir o tipo de classe.
  - **JOINED**
    - Uma tabela por cada classe
    - representa a relação “*is-a*” por uma relação “*has-a*” (com chaves estrangeiras)
  - **TABLE\_PER\_CLASS**
    - Uma tabela para cada classe concreta;
    - herança e polimorfismo não são considerados, nem representados, no modelo relacional.

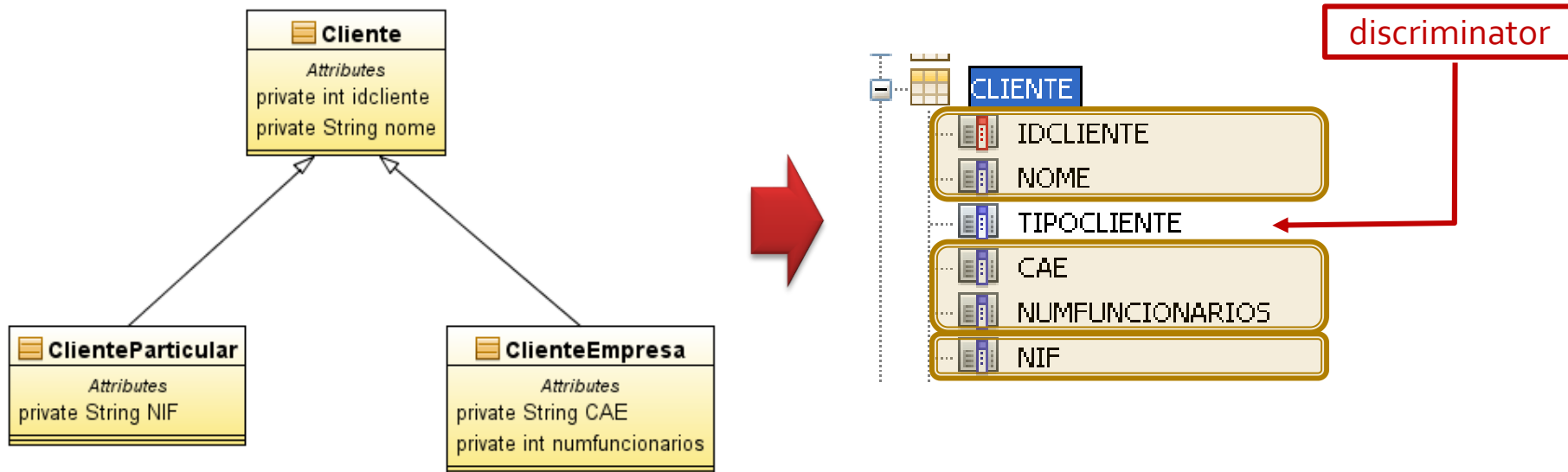
# Mapeamento de Herança entre classes

- Exemplo base:



# Estratégia SINGLE\_TABLE

- **Uma tabela para toda a hierarquia de classes: SINGLE\_TABLE**
  - Todas as classes são mapeadas numa única tabela.
  - A tabela contém todas as propriedades de todas as classes.
  - Acrescenta-se na tabela um “discriminator” - TIPOCLIENTE - que indica qual a classe concreta do registo



# Estratégia SINGLE\_TABLE

- ***Superclass Cliente***

```
@Entity
@Table(name="CLIENTE")
@Inheritance(strategy= InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISC")
public abstract class Cliente {

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

    String nome;
    (...)

    public Cliente () {}

    (...)
}
```

# Estratégia SINGLE\_TABLE

- ***Classes derivadas***

```
@Entity
@DiscriminatorValue("EMP")
public class ClienteEmpresa extends Cliente{
    private String CAE;
    private int numFuncionarios;
    public ClienteEmpresa() {}

    (...)
}
```

```
@Entity
@DiscriminatorValue("PAR")
public class ClienteParticular extends Cliente{
    private String NIF;
    public ClienteParticular() {}

    (...)
}
```

# Estratégia SINGLE\_TABLE

```
ClienteParticular cliEmpr = new ClienteEmpresa("Cliente Empresa",4, "12345");
```

```
manager.getTransaction().begin();  
manager.persist(cliEmpr);  
manager.getTransaction().commit();
```

(...)

```
// carrega o cliente empresa num objeto do tipo da superclasse Cliente
```

```
int idCliEmpresa=1;
```

```
Cliente cli=manager.find(Cliente.class, idCliEmpresa)
```

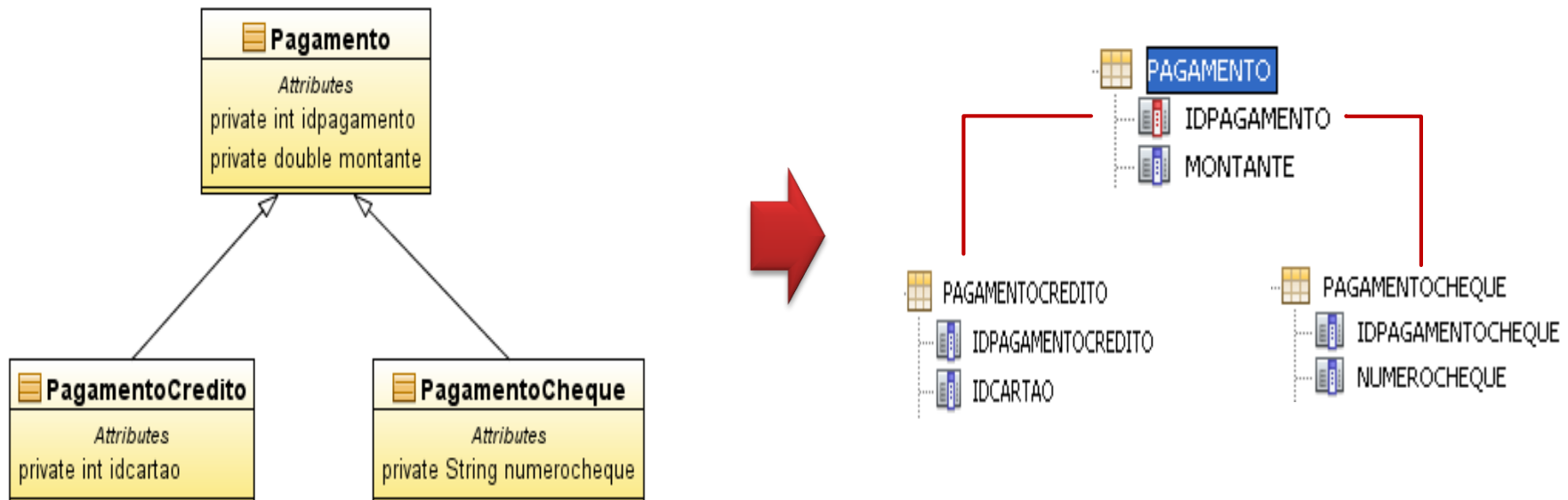
```
// escreve, por polimorfismo, a informação de um cliente empresa
```

```
System.out.println(cli);
```

#	IDCLIENTE	NOME	TIPOCLIENTE	CAE	NUMFUNCIONARIOS	NIF
1	1	Cliente Empresa	EMP	12345	4	<NULL>
2	2	Cliente particular	PAR	<NULL>	<NULL>	162107048
3	3	Cliente Empresa	EMP	12345	4	<NULL>
4	4	Cliente particular	PAR	<NULL>	<NULL>	162107048

# Estratégia JOINED

- **Uma tabela para cada classe da hierarquia: JOINED**
  - Cada classe é mapeada numa tabela
  - A tabela de mapeamento de cada classe contém:
    - colunas apenas para as propriedades declaradas na classe (*não herdadas*)
    - uma chave primária que é também uma chave estrangeira para a tabela da super classe





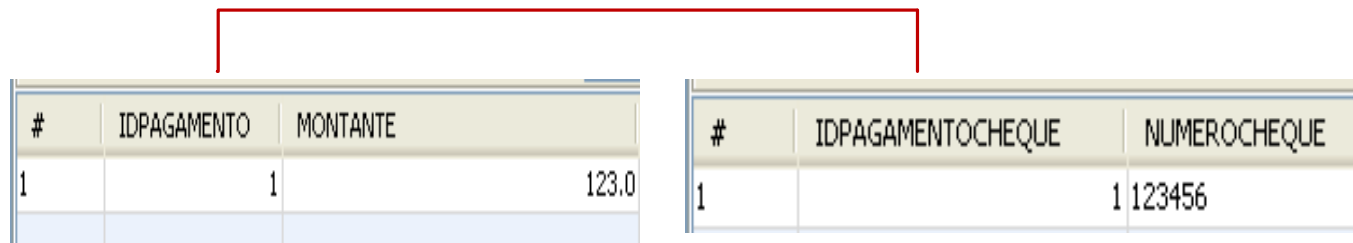
# Estratégia JOINED

- ***Uma tabela para cada classe da hierarquia***

- Instanciação da subclasse

```
PagamentoCheque cheque= new PagamentoCheque("123456",123.76);  
session.save(chegue);
```

- É criado um registo na tabela PAGAMENTO (*super*) com o montante;
- É criado um registo na tabela PAGAMENTOCHIQUE com o número do cheque;
- Os dois registos são ligados pela chave primária comum.



# Estratégia JOINED

- ***Superclass Pagamento***

```
@Entity
@Table(name="PAGAMENTO")
@Inheritance(strategy=InheritanceType.JOINED)
public class Pagamento {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="IDPAGAMENTO")
    protected int idPagamento;
    protected double montante;

    public Pagamento() { }

    void setMontante(double montante) {
        this.montante=montante;
    }
}
```

# Estratégia JOINED

- **Classes derivadas**

```
@Entity
```

```
@Table(name="PAGAMENTOCHIQUE")
```

```
public class PagamentoCheque extends Pagamento {  
    private String numeroCheque;  
    public PagamentoCheque(){}  
    public PagamentoCheque(String numeroCheque,double montante) {  
        super.setMontante(montante);  
        this.numeroCheque=numeroCheque;  
    }  
    (...)  
}
```

```
@Entity
```

```
@Table(name="PAGAMENTOCREDITO")
```

```
public class PagamentoCredito extends Pagamento {  
    private String idCartao;  
    public PagamentoCredito() { }  
    (...)  
}
```

# Estratégia JOINED

- ***Código exemplo:***

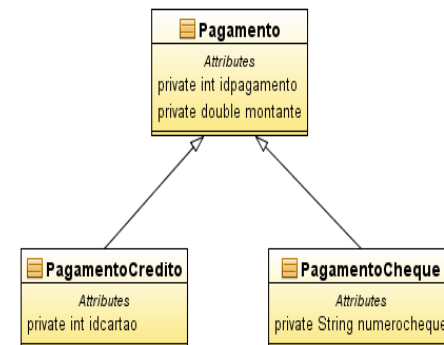
```
EntityManager manager = PersistenceInit.getEntityManager();  
  
PagamentoCheque pagCheque = new PagamentoCheque("12345678",200);  
PagamentoCredito pagCredito = new PagamentoCredito("xpto12345",300);  
  
manager.getTransaction().begin();  
manager.persist(pagCheque);  
  
(...)
```

# Estratégia JOINED

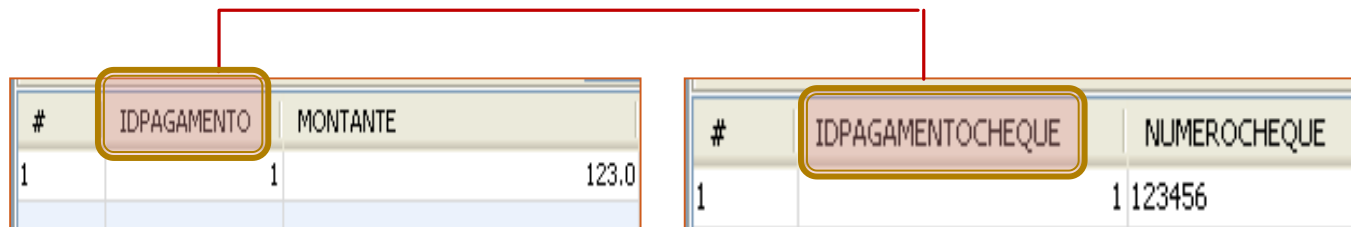
## ■ Resumindo

### ■ Instanciar uma subclasse

```
PagamentoCheque cheque =  
    new PagamentoCheque("123456",123.76);  
session.save(chegue);
```



- É criado um registo na tabela PAGAMENTO (*super*) com o montante;
- É criado um registo na tabela PAGAMENTOOCHEQUE com o número do cheque;
- Os dois registos são ligados pela chave primária comum.



# Estratégia TABLE\_PER\_CLASS

- ***Uma tabela por classe concreta: TABLE\_PER\_CLASS***

superclass

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int idCliente;
    private String nome;
    (...)
}
```

subclass

```
@Entity
public class ClienteEmpresa extends Cliente {
    private String CAE;
    private int numFuncionarios;
    (...)
}
```

subclass

```
@Entity
public class ClienteParticular extends Cliente {
    private String NIF;
    (...)
}
```

# Estratégia TABLE\_PER\_CLASS

- **Código exemplo:**

```
public static void main(String[] args) {
    EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("JPAClientesPU");
    EntityManager em = emf.createEntityManager();

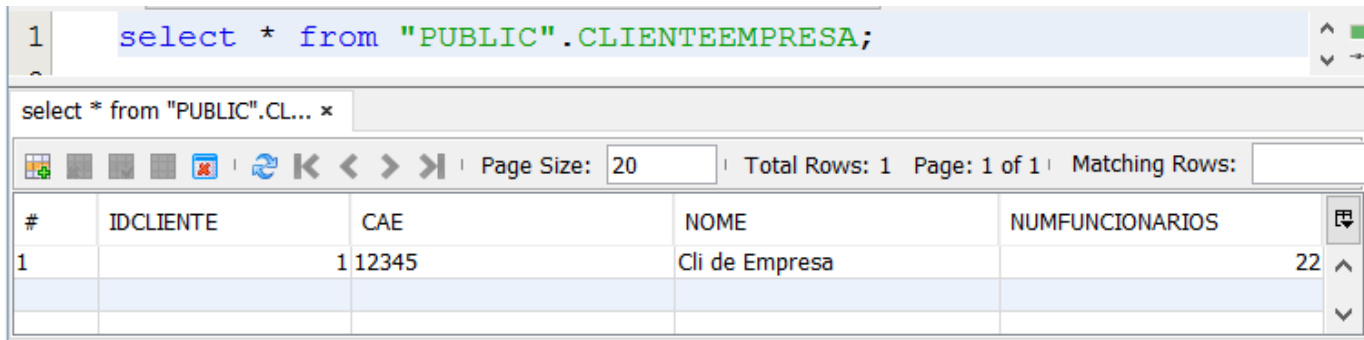
    ClienteParticular c1 = new ClienteParticular("Ana", "123456789");
    ClienteEmpresa c2 = new ClienteEmpresa("Cli de Empresa", "12345", 22);
    ClienteParticular c3 = new ClienteParticular("Cli Part", "987654321");

    em.getTransaction().begin();
    em.persist(c1);
    em.persist(c2);
    em.persist(c3);
    em.getTransaction().commit();

    em.close();
    emf.close();
}
```

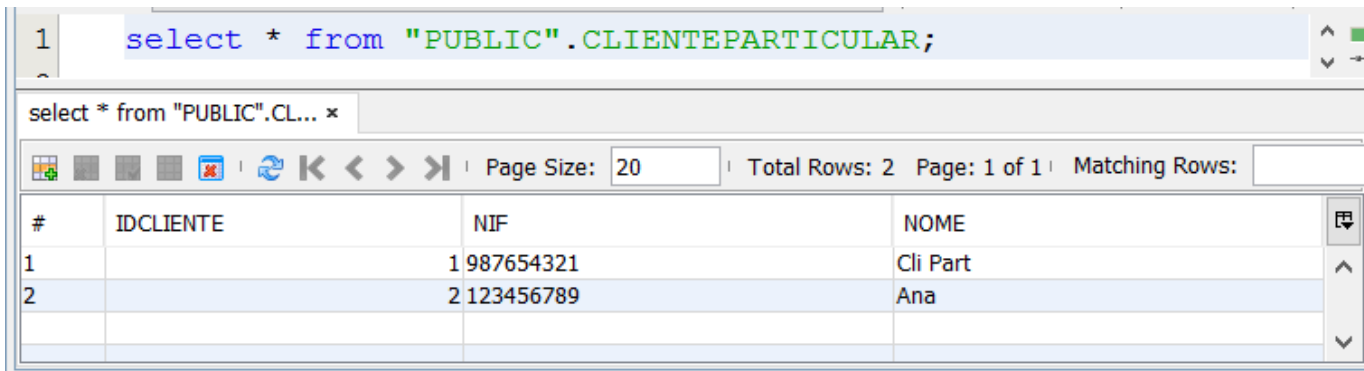
# Estratégia TABLE\_PER\_CLASS

- **Foram criadas duas tabelas:**



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `select * from "PUBLIC".CLIENTEMPRESA;`. Below the query, a table of results is displayed. The table has five columns: #, IDCLIENTE, CAE, NOME, and NUMFUNCIONARIOS. There is one row of data.

#	IDCLIENTE	CAE	NOME	NUMFUNCIONARIOS
1		1 12345	Cli de Empresa	22



The screenshot shows a database query tool interface. At the top, a SQL query is entered: `select * from "PUBLIC".CLIENTEPARTICULAR;`. Below the query, a table of results is displayed. The table has four columns: #, IDCLIENTE, NIF, and NOME. There are two rows of data.

#	IDCLIENTE	NIF	NOME
1		1 987654321	Cli Part
2		2 123456789	Ana

- Uma **tabela por classe concreta**;
- Cada tabela inclui **todo o estado** de uma instância da correspondente classe, incluindo propriedades herdadas.