Cardinalidades

- Usualmente, objetos não estão sozinhos.
  - Um objeto pode possuir um conjunto de outros objetos
- Modelo de Objetos x Modelo Relacional
  - Diagrama Entidade Relacionamento
  - Diagrama de Classes
  - Diagrama de Objetos

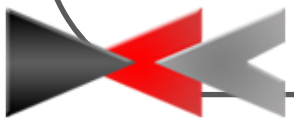
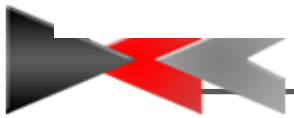


Diagrama de Classes

- Na prática o diagrama de classes é bem mais utilizado que o diagrama de objetos.
- Tanto que o modelo de objetos é também conhecido como modelo de classes.
- Esse modelo evolui durante o desenvolvimento do SSOO.
- À medida que o SSOO é desenvolvido, o modelo de objetos é incrementado com novos detalhes.



UML

- Uma classe descreve esses objetos através de atributos e operações.
  - Atributos correspondem às informações que um objeto armazena.
  - Operações correspondem às ações que um objeto sabe realizar.
- Notação na UML: “caixa” com no máximo três compartimentos exibidos.
  - Detalhamento utilizado depende do estágio de desenvolvimento e do nível de abstração desejado.

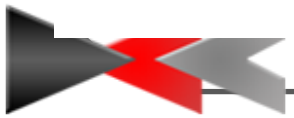
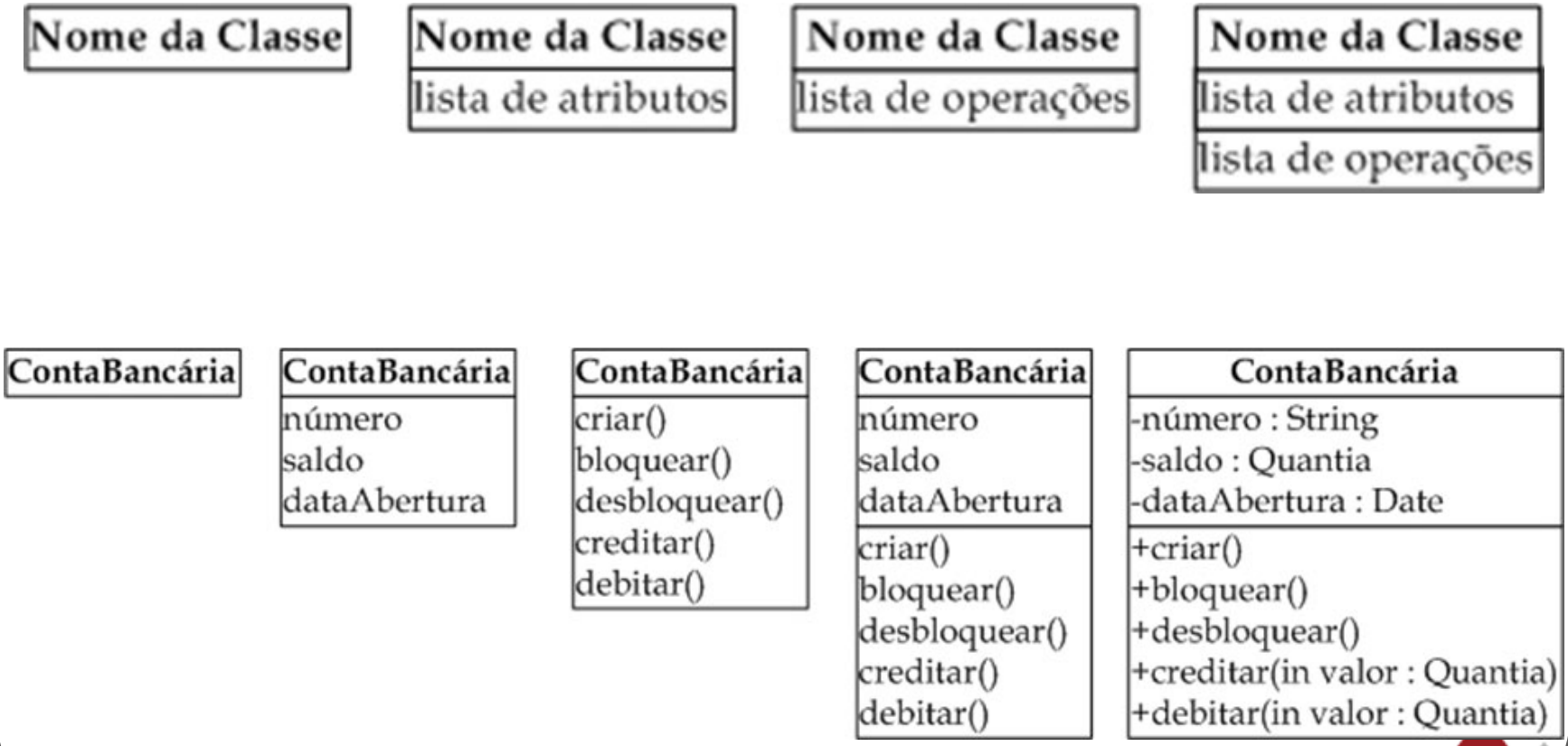
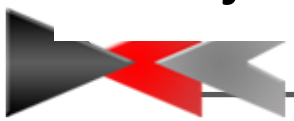


Diagrama de Classes



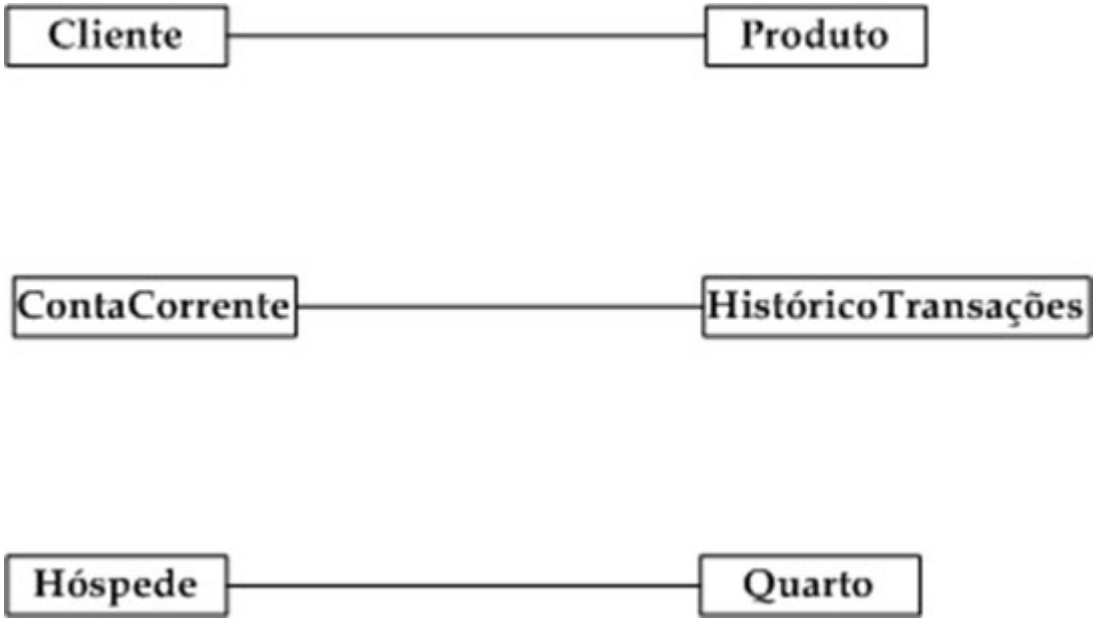
Associações

- Para representar o fato de que objetos podem se relacionar uns com os outros, utilizamos associações.
- Uma associação representa relacionamentos (ligações) que são formados entre objetos durante a execução do sistema.
- Note que, embora as associações sejam representadas entre classes do diagrama, tais associações representam ligações possíveis entre os objetos das classes envolvidas.



Notação

- Na UML associações são representadas por uma linha que liga as classes cujos objetos se relacionam.
- Exemplos:



Multiplicidades

- Representam a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode se associar.
- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha de associação.

Nome	Simbologia na UML
Apenas Um	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	$l_i..l_s$

Exemplos

- Exemplo:
  - Pode haver um cliente que esteja associado a vários pedidos.
  - Pode haver um cliente que não esteja associado a pedido algum.
  - Um pedido está associado a um, e somente um, cliente.

Exemplos

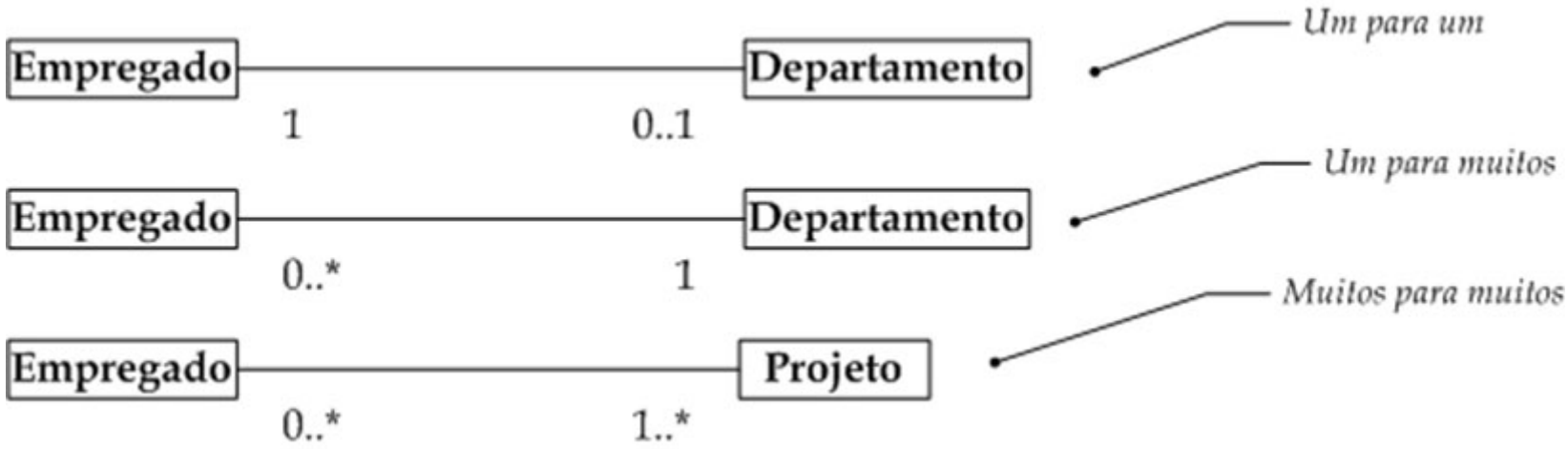
- Exemplo
  - Uma corrida está associada a, no mínimo, dois velocistas
  - Uma corrida está associada a, no máximo, seis velocistas.
  - Um velocista pode estar associado a várias corridas.

Conectividade

- A conectividade corresponde ao tipo de associação entre duas classes: “muitos para muitos”, “um para muitos” e “um para um”.
- A conectividade da associação entre duas classes depende dos símbolos de multiplicidade que são utilizados na associação.

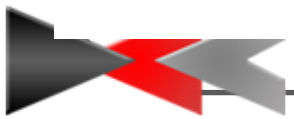
Conectividade	Em um extremo	No outro extremo
Um para um	0..1 1	0..1 1
Um para muitos	0..1 1	* 1..* 0..*
Muitos para muitos	* 1..* 0..*	* 1..* 0..*

Exemplo


Empregado
Departamento
1
0..1
Um para um
Empregado
Departamento
0..*
1
Um para muitos
Empregado
Projeto
0..*
1..*
Muitos para muitos

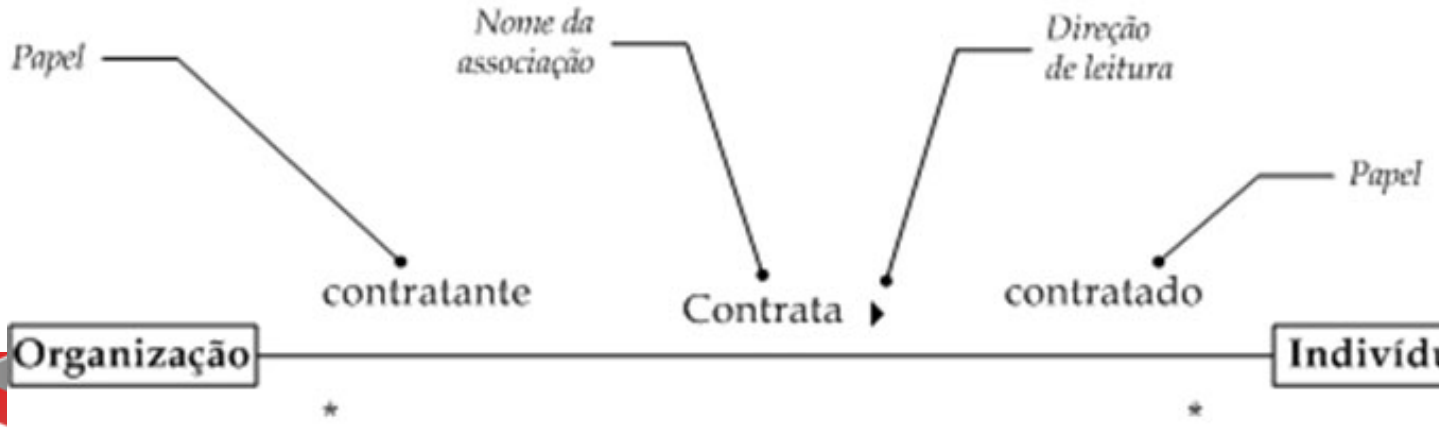
Participação

- Uma característica de uma associação que indica a necessidade (ou não) da existência desta associação entre objetos.
- A participação pode ser obrigatória ou opcional.
  - Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória
  - Caso contrário, a participação é opcional.



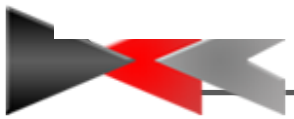
Cenário para Associações

- Para melhor esclarecer o significado de uma associação no diagrama de classes, a UML define três recursos de notação:
  - Nome da associação: fornece significado semântico.
  - Direção de leitura: indica como a associação deve ser lida
  - Papel: representa um papel específico em uma associação.



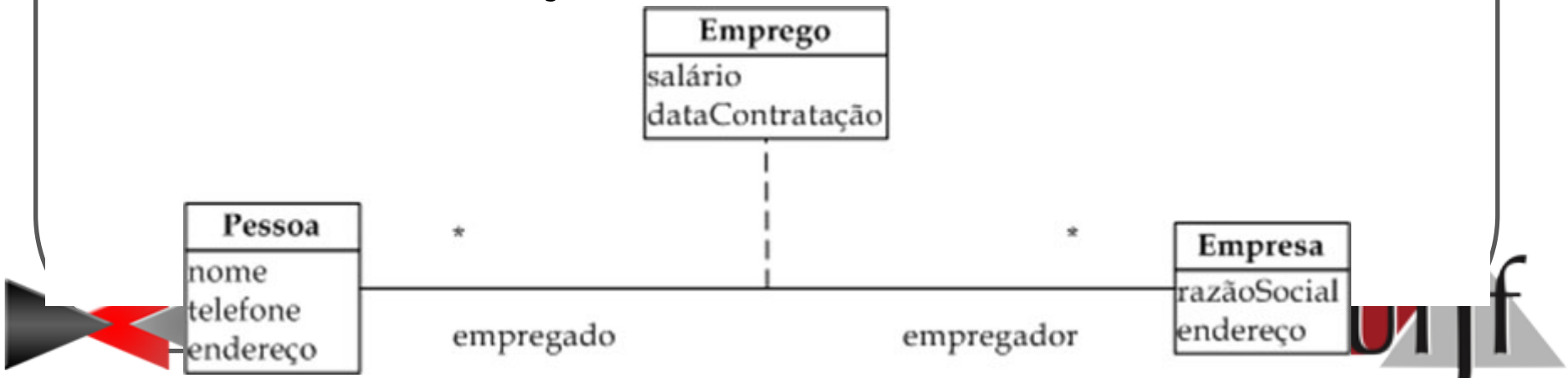
Classe Associativa

- É uma classe que está ligada a uma associação, em vez de estar ligada a outras classes.
- É normalmente necessária quando duas ou mais classes estão associadas, e é necessário manter informações sobre esta associação.
- Uma classe associativa pode estar ligada a associações de qualquer tipo de conectividade.
- Sinônimo: classe de associação



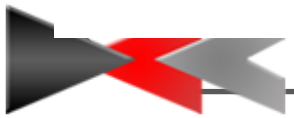
Notação para Classes Associativas

- Notação é semelhante à utilizada para classes ordinárias. A diferença é que esta classe é ligada a uma associação por uma linha tracejada.
- Exemplo: para cada par de objetos [pessoa, empresa], há duas informações associadas: salário e data de contratação.



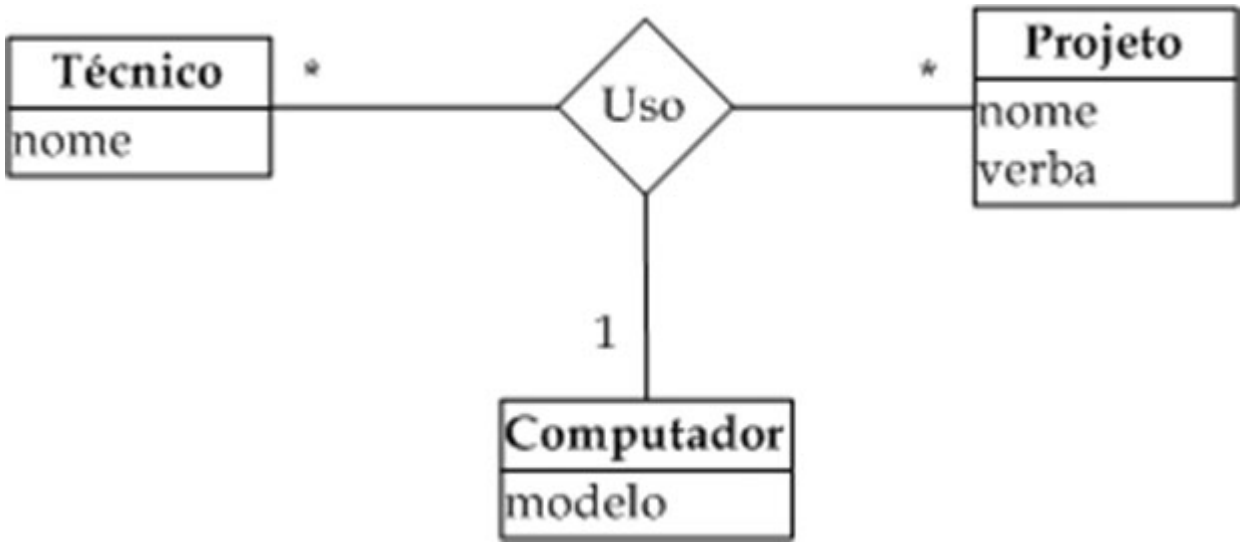
Associações n-árias

- Define-se o grau de uma associação como a quantidade de classes envolvidas na mesma.
- Na notação da UML, as linhas de uma associação n-ária se interceptam em um losango.
- Na grande maioria dos casos práticos de modelagem, as associações normalmente são binárias.
- Quando o grau de uma associação é igual a três, dizemos que a mesma é ternária.
  - Uma associação ternária é o caso mais comum (menos raro) de associação n-ária (n = 3).



Associação Ternária

- Na notação da UML, as linhas de uma associação n-ária se interceptam em um losango nomeado.
  - Notação similar ao do Modelo de Entidades e Relacionamentos

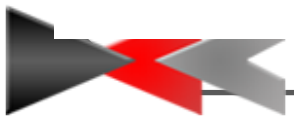


Associações Reflexivas

- Tipo especial de associação que representa ligações entre objetos que pertencem a uma mesma classe.
  - Não indica que um objeto se associa a ele próprio.
- Quando se usa associações reflexivas, a definição de papéis é importante para evitar ambiguidades na leitura da associação.
  - Cada objeto tem um papel distinto na associação.

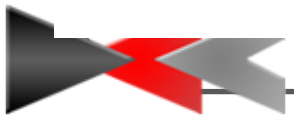
Agregações e Composições

- A semântica de uma associação corresponde ao seu significado, ou seja, à natureza conceitual da relação que existe entre os objetos da associação.
- De todos os significados diferentes que uma associação pode ter, há uma categoria especial de significados, que representa relações todo-parte.
- Uma relação todo-parte entre dois objetos indica:
  - Um objeto contém ou está contido no outro.
- UML define dois tipos de relacionamentos todo-parte, a agregação e a composição.



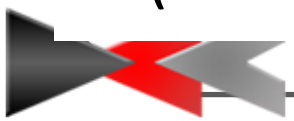
Agregações e Composições

- Particularidades das agregações/composições:
  - São assimétricas, no sentido de que, se um objeto A é parte de um objeto B, o objeto B não pode ser parte do objeto A.
  - Propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica às suas partes.
  - As partes são normalmente criadas e destruídas pelo todo. Na classe do objeto todo, são definidas operações para adicionar e remover as partes.

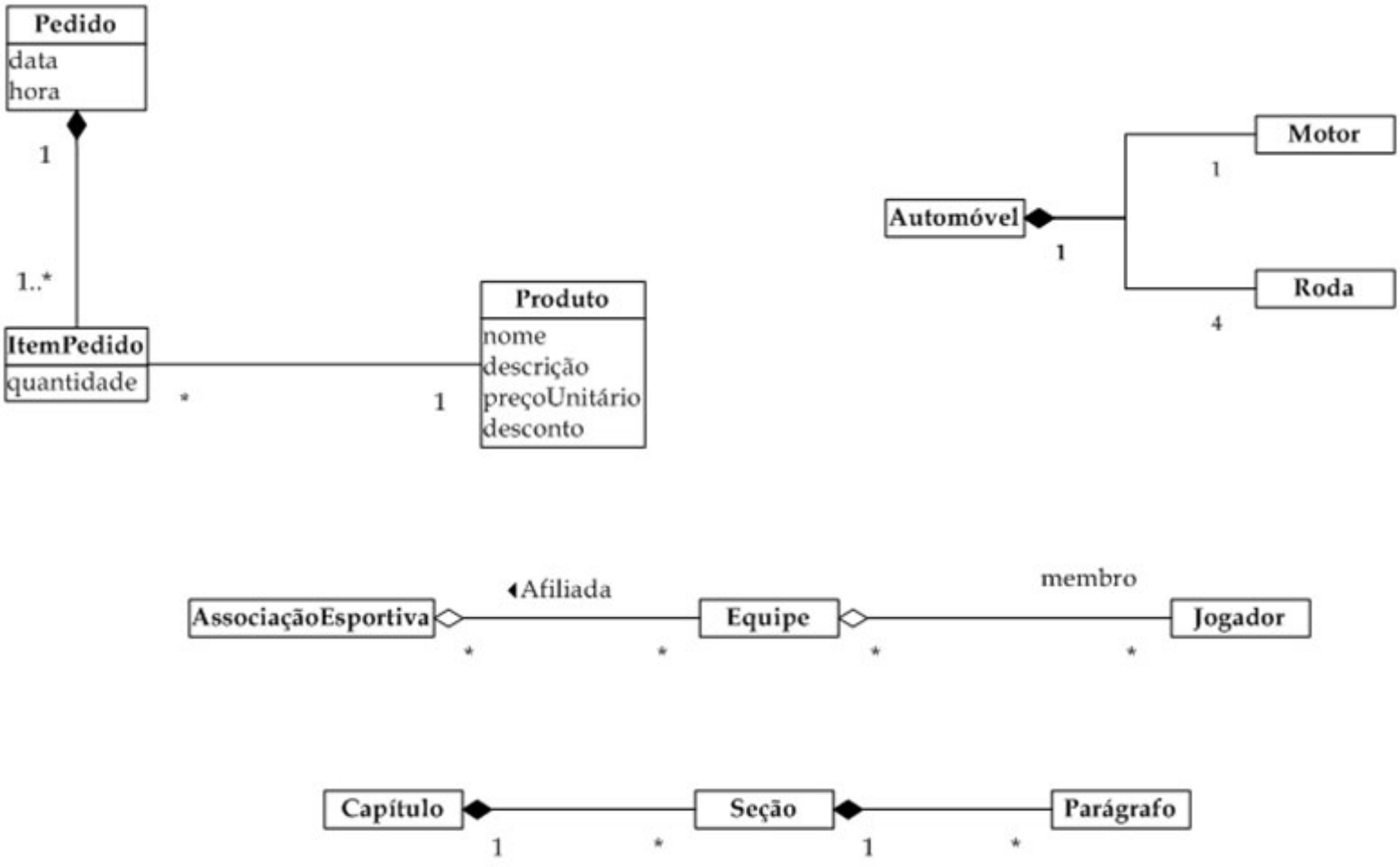


Agregações e Composições

- Se uma das perguntas a seguir for respondida com um sim, provavelmente há uma agregação onde X é todo e Y é parte.
  - X tem um ou mais Y?
  - Y é parte de X?
- Composição na UML é representada por um diamante (losango) negro.
- Agregação na UML é representada por um diamante (losango) branco.

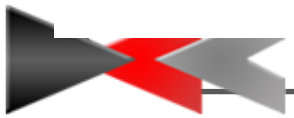


Exemplos


Pedido
data
hora
1
1..*
ItemPedido
quantidade
*
1
Produto
nome
descrição
preçoUnitário
desconto
Motor
1
Automóvel
1
Roda
4
AssociaçãoEsportiva
Afiliada
Equipe
membro
Jogador
* * * *
Capítulo
Seção
Parágrafo
1 * 1 *

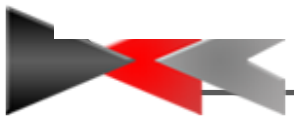
Agregações e Composições

- As diferenças entre a agregação e composição não são bem definidas. A seguir, as diferenças mais marcantes entre elas.
- Destruição de objetos
  - Na agregação, a destruição de um objeto todo não implica necessariamente na destruição do objeto parte. Exemplo: Associação esportiva.



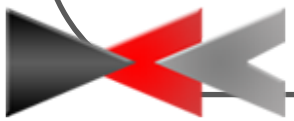
Agregações e Composições

- Pertinência
  - Na composição (losango negro), os objetos parte pertencem a um único todo. Ex. Automóvel.
    - Por essa razão, a composição é também denominada agregação não-compartilhada.
  - Em uma agregação (losango branco), pode ser que um mesmo objeto participe como componente de vários outros objetos. Ex. Associação Esportiva.
    - Por essa razão, a agregação é também denominada agregação compartilhada.

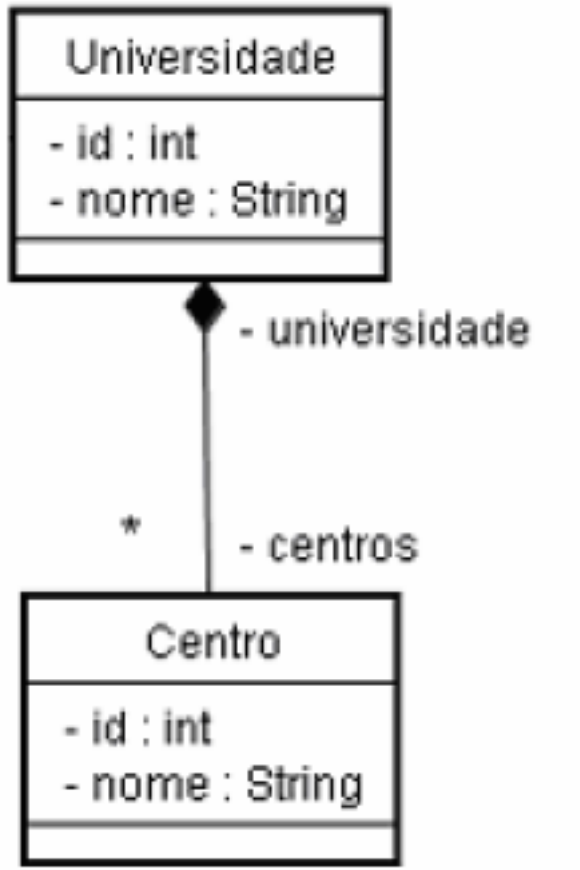


Cardinalidades

- Mapeamentos de Associações
  - 1-N
  - N-N
  - 1-1

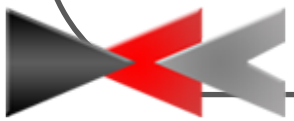


Associações 1-N



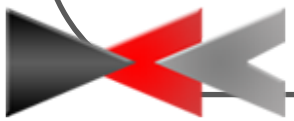
Tabelas

```
CREATE TABLE universidade (
id_universidade integer NOT NULL GENERATED
ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
nome character(100) NOT NULL, -- Nome da universidade
CONSTRAINT pk_universidade PRIMARY KEY (id_universidade))

CREATE TABLE centro (
id_centro integer NOT NULL GENERATED
ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
nome character(100) NOT NULL, -- Nome do centro
id_universidade integer NOT NULL,
-- Identificador da universidade a que o centro pertence
CONSTRAINT pk_centro PRIMARY KEY (id_centro),
CONSTRAINT fk_centro_universidade FOREIGN KEY (id_universidade)
REFERENCES universidade (id_universidade))


CREATE SEQUENCE universidade_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE SEQUENCE centro_seq START WITH 1 INCREMENT BY 1 NO CYCLE;
```



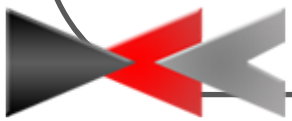
Classe Universidade

```java
package hibernate;

import java.util.Collection;
import javax.persistence.*;
import org.hibernate.annotations.Cascade;
import org.hibernate.annotations.CascadeType;

@SequenceGenerator(name = "seq", sequenceName = "universidade_seq")
@Entity
@Table
public class Universidade {
    @Id
    @GeneratedValue(generator = "seq")
    @Column(name="id_universidade")
    private int id;
    private String nome;
```



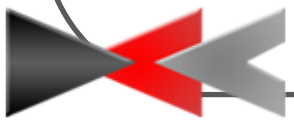
Classe Universidade

```java
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public Collection<Centro> getCentros() {
    return centros;
}

public void setCentros(Collection<Centro> centros) {
    this.centros = centros;
}
```



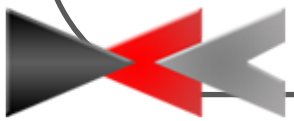
Classe Universidade

```
@OneToMany(mappedBy="universidade", fetch = FetchType.LAZY)
@Cascade(CascadeType.ALL)
private Collection<Centro> centros;
```



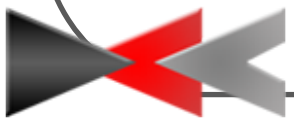
Universidade

- Anotação @OneToMany
  - Define a relação 1-N
- Atributo mappedBy
  - nome do campo da classe centro que se refere à universidade – chave estrangeira
- Atributo fetch
  - Indica quando o conteúdo será trazido da base de dados
  - EAGER: na instanciação da classe
  - LAZY: quando o conteúdo for utilizado



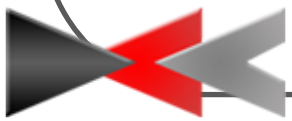
fetch = FetchType.LAZY

```
Hibernate: select universida0_.id_universidade as id1_3_0_,
      universida0_.nome as nome3_0_
from anotacoes.Universidade universida0_
where universida0_.id_universidade=?
```



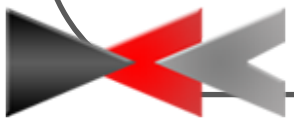
fetch = FetchType.EAGER

```
Hibernate: select universida0_.id_universidade as id1_3_1_,
      universida0_.nome as
      nome3_1_, centros1_.id_universidade as id3_3_,
      centros1_.id_centro as id1_3_, centros1_.id_centro as id1_1_0_,
      centros1_.nome as nome1_0_,
      centros1_.id_universidade as id3_1_0_
from anotacoes.Universidade universida0_
left outer join anotacoes.Centro centros1_ on
      universida0_.id_universidade=centros1_.id_universidade
where universida0_.id_universidade=?
```



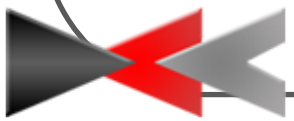
Universidade

- Anotação @Cascade
  - Define a relação entre as classes para as operações do banco.
  - Com a opção ALL, todas as operações sobre essa classe serão refletidas nas classes que contém os relacionamentos
  - Muito associado as operações de agregação
  - Diversas opções: REMOVE, PERSISTE, etc.



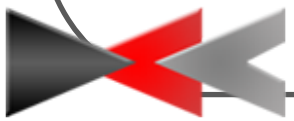
@Cascade(CascadeType.ALL)

```
Hibernate: insert into anotacoes.Universidade
      (nome, id_universidade) values (?, ?)
Hibernate: insert into anotacoes.Centro
      (nome, id_universidade, id_centro) values (?, ?, ?)
Hibernate: insert into anotacoes.Centro
      (nome, id_universidade, id_centro) values (?, ?, ?)
```



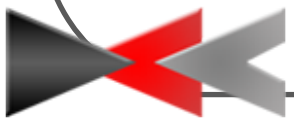
Classe Centro

```java
import javax.persistence.*;
import org.hibernate.annotations.Cascade;
import org.hibernate.annotations.CascadeType;
import org.hibernate.annotations.Fetch;
import org.hibernate.annotations.FetchMode;

@SequenceGenerator(name = "seq", sequenceName = "centro_seq")
@Entity
@Table
public class Centro {
    @Id
    @GeneratedValue(generator = "seq")
    @Column(name="id_centro")
    private int id;
    private String nome;
```



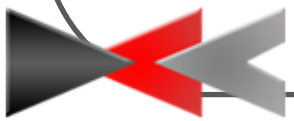
Classe Centro

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name="id_universidade", insertable=true, updatable=true)
@Fetch(FetchMode.JOIN)
@Cascade(CascadeType.SAVE_UPDATE)
private Universidade universidade;
```



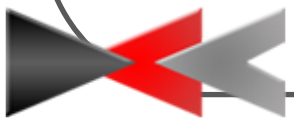
Classe Centro

```
public Universidade getUniversidade() {
    return universidade;
}
public void setUniversidade(Universidade universidade) {
    this.universidade = universidade;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getNome() {
    return nome;
}
public void setNome(String nome) {
    this.nome = nome;
}
```



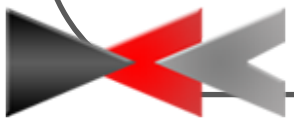
Centro

- Anotação @ManyToOne
  - Define a relação 1-N
- Atributo fetch
  - Indica quando o conteúdo será trazido da base de dados
- Anotação @JoinColumn
  - Informa o nome da coluna que corresponde à chave estrangeira
- Atributos Insertable e Updatable



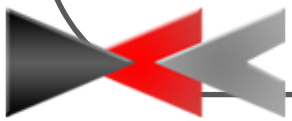
Centro

- Anotação @Fetch
  - Define como o atributo será recuperado da base de dados
- Opções JOIN, SELECT e SUBSELECT
  - Definem a forma como a consulta será construída pelo Hibernate



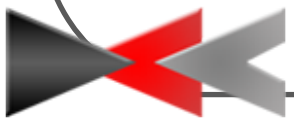
Main

```java
import java.util.HashSet;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class Hibernate {

    public static void main(String[] args) {
        // TODO code application logic here
        //Cria objeto que receberá as configurações
        Configuration cfg = new Configuration();
        //Informe o arquivo XML que contém a configurações
        cfg.configure("hibernate.cfg.xml");
        //Cria uma fábrica de sessões.
        //Deve existir apenas uma instância na aplicação
        SessionFactory sf = cfg.buildSessionFactory();
        // Abre sessão com o Hibernate
        Session session = sf.openSession();
        //Cria uma transação
        Transaction tx = session.beginTransaction();
```



Main

```
Universidade univ = new Universidade();
univ.setNome("Universidade Federal de Juiz de Fora");
Centro centro1 = new Centro();
centro1.setNome("Centro de Tecnologia");
centro1.setUniversidade(univ);
Centro centro2 = new Centro();
centro2.setNome("Centro de Humanas");
centro2.setUniversidade(univ);
univ.setCentros(new HashSet<Centro>());
univ.getCentros().add(centro1);
univ.getCentros().add(centro2);
session.save(univ);
```



Main

```
        univ =(Universidade)session.get(Universidade.class, 2);
//Acesso à coleção centros, de forma que os dados serão buscados
        univ.getCentros().iterator();
        System.out.println(univ.getNome());

        tx.commit(); // Finaliza transação
        session.close(); // Fecha sessão
    }
}
```