- Binária
- Multiplicidade
- Direcionamento
- Reflexiva
- Particularidades

Associação entre Classes

Prof.Dr. Enzo Seraphim

Associações binárias

Binária

- Uma associação é uma ligação entre objetos das classes participantes (um objeto de cada classe em cada ligação)
- Essa ligação é uma instância de uma associação
- Implementado através de uma referência entre os objetos relacionados
- Pode haver mais do que uma associação (com nomes diferentes) entre o mesmo par de classes

Papel

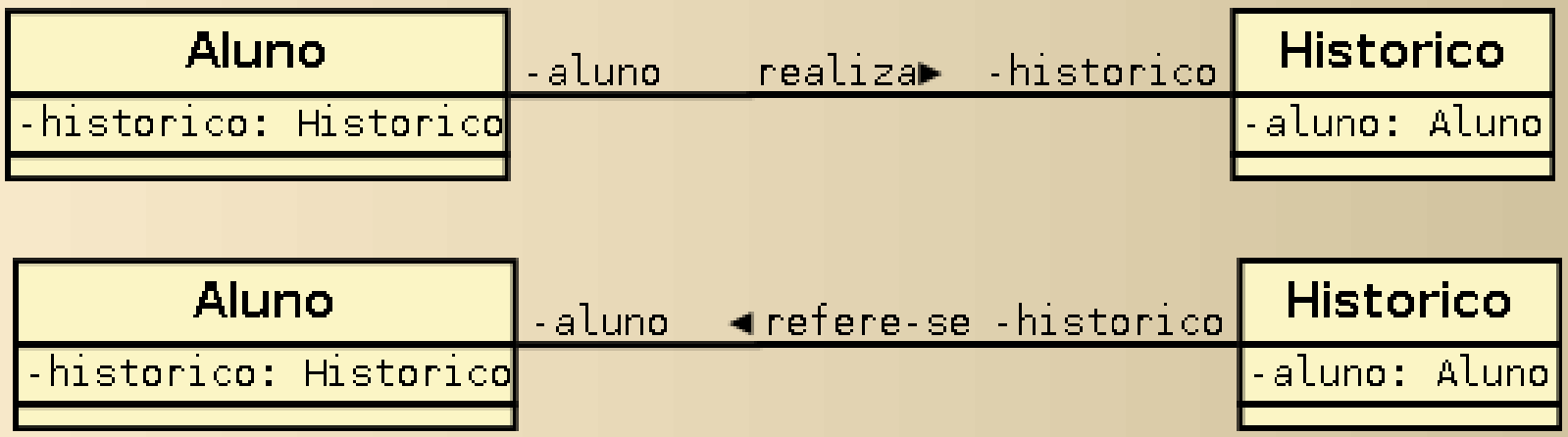
Binária

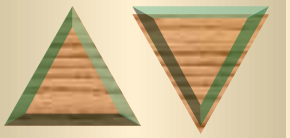
- São apresentados nos extremos das classes participantes da associação
- Indicam nome do atributo a ser repassado para classe do extremo oposto
- Podem ter visibilidade (+, -, #)

Nome de associação

Associação

- É opcional e normalmente utliza-se verbo
- O nome é indicado no meio da linha que une as classes participantes
- Pode-se indicar o sentido em que se lê o nome da associação





- Binária
- Multiplicidade
- Direcionamento
- Reflexiva
- Particularidades

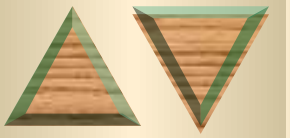
Associação entre Classes

Prof.Dr. Enzo Seraphim

Definições

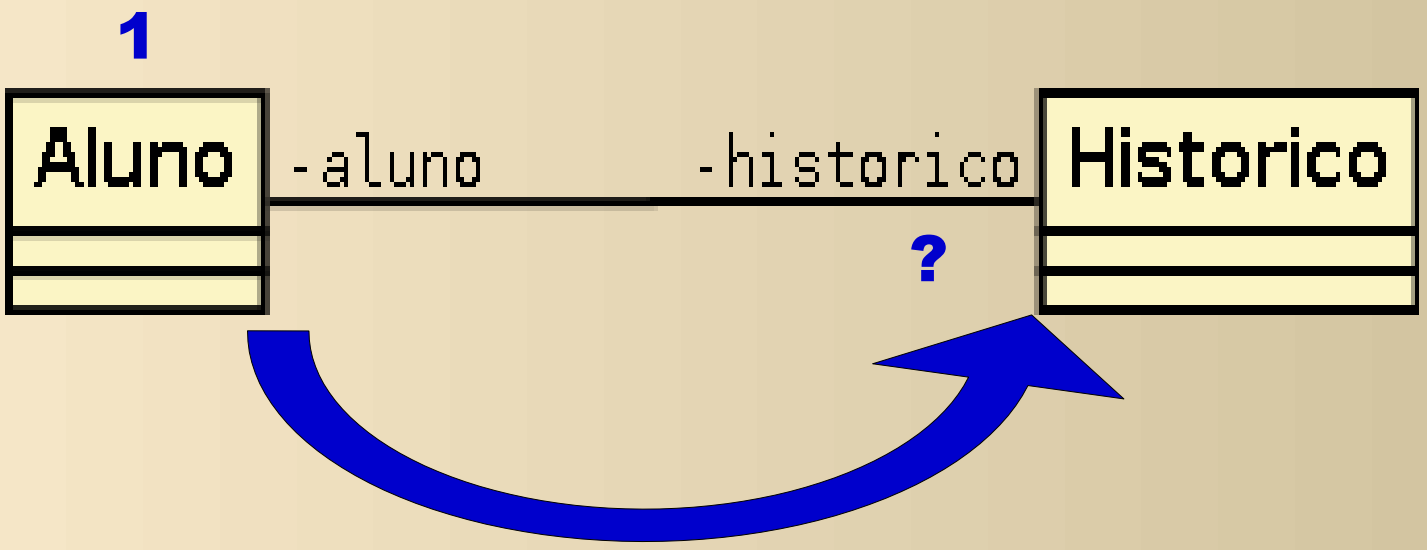
Multiplicidade

- São apresentados nos extremos das classes participantes da associação
- Indicam a participação de cada elemento com a classe do extremo oposto
- Multiplicidade depende exclusivamente do contexto ou cenário ou mini mundo ou regra negócio



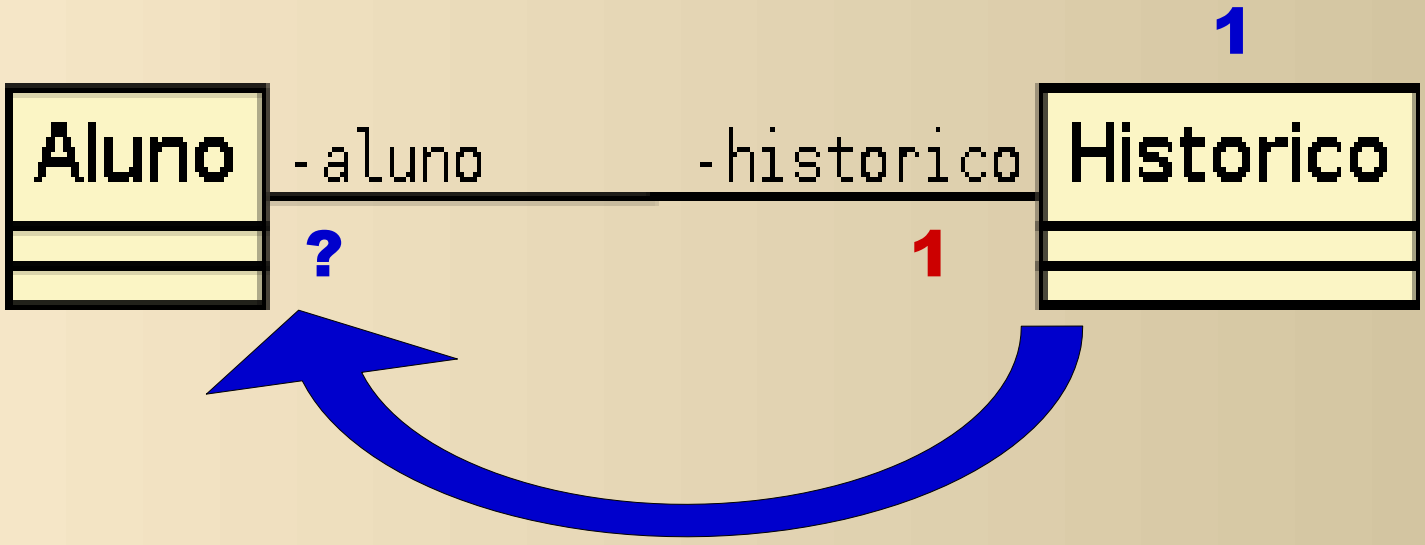
Estabelecendo Multiplicidade

Multiplicidade



- Com 1 elemento instanciado da classe pergunta-se ao extremo qual é a participação
- O aluno “Luciano” pode ter quantos históricos?

Estabelecendo Multiplicidade



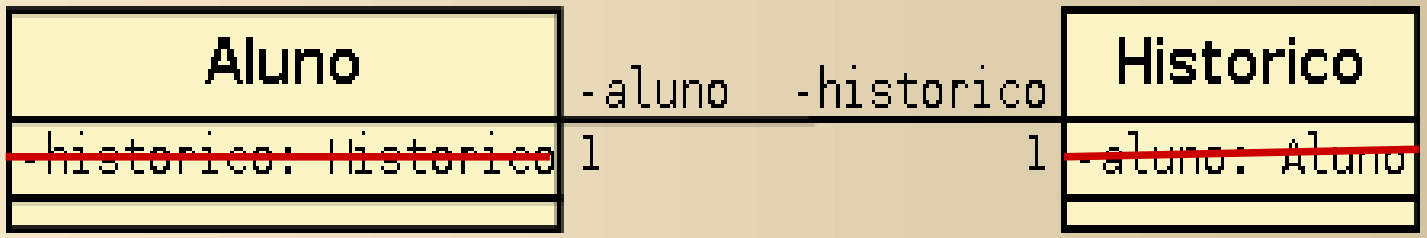
- Um histórico pode ter quantos alunos?

Estabelecendo Multiplicidade

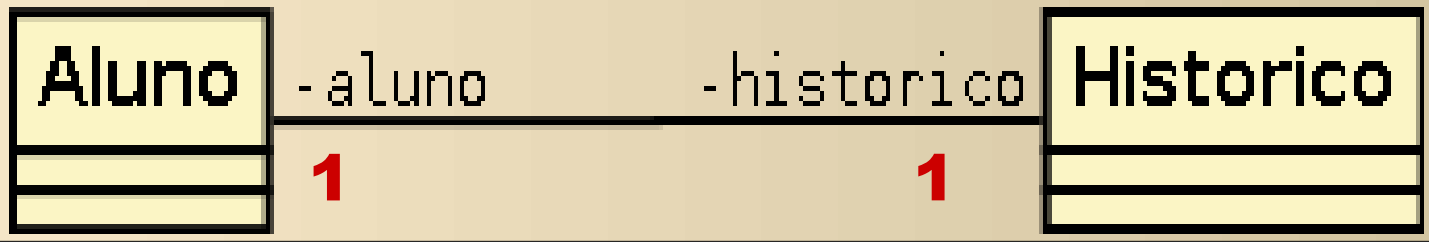
Multiplicidade

- Omissão multiplicidade indica valor **um**

- Associação não de ver declarada como atributo



Multiplicidade

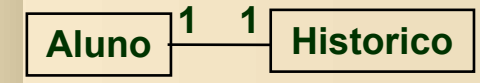


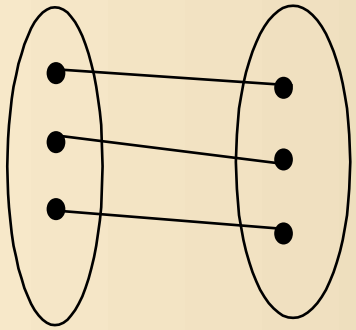

```
public class Aluno {
 private Historico historico;
 public Historico getHistorico() {
   return historico; }
 public void setHistorico(Historico historico){
   this.historico = historico; }
}
public class Historico {
 private Aluno aluno;
 public Aluno getAluno() {return aluno;}
 public void setAluno(Aluno aluno) {
   this.aluno = aluno; } }
```

Multiplicidade

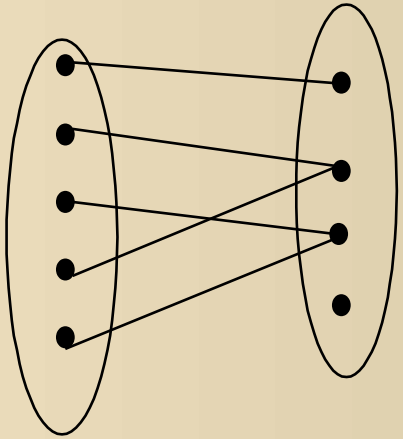
Multiplicidade

1-para-1

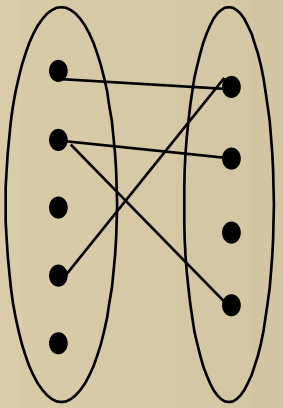




Muitos-para-1
1-para-Muitos



Muitos-para-Muitos



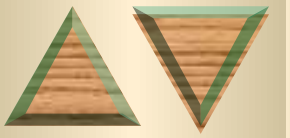
1 - exatamente um
0..1 - zero ou um (zero a 1)
* - zero ou mais
0..* - zero ou mais
1..* - um ou mais
1, 3..5- um ou três a 5

Multiplicidade Várias (*)

Multiplicidade

- **Array** → coleção ordenada com tamanho fixo e permite repetição
- Acesso direto com tempo constante para iteração com qualquer posição da coleção
- Limitado ao tamanh do vetor
- Alocação do vetor não aloca objetos do Vetor

```
private long telefones[] = new long[2];
private String alergias[] = new String[3];
private Carro carro[] = new Carro[4];
```



Multiplicidade Várias (*)

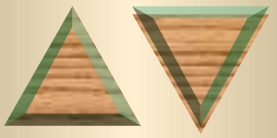
Multiplicidade

- **Lista** ↓ coleção ordenada sem tamanho fixo e permite repetição
- List é abstrata com implentações em Vetor e Encadeamento
- **ArrayList** tem tempo constante para iteração com qualquer posição da coleção
  - Inserção em um vetor cheio exige alocação de novo vetor maior com copia dos elementos

```
private List<Carro> carros = new ArrayList<Carro>();
```

- LinkedList nunca fica cheio, mas iteração com a posição **i** exige iteração com **i-1** elementos

```
private List<Carro> carros = new LinkedList<Carro>();
```

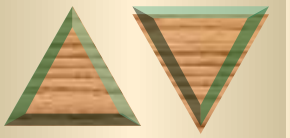


Multiplicidade Várias (*)

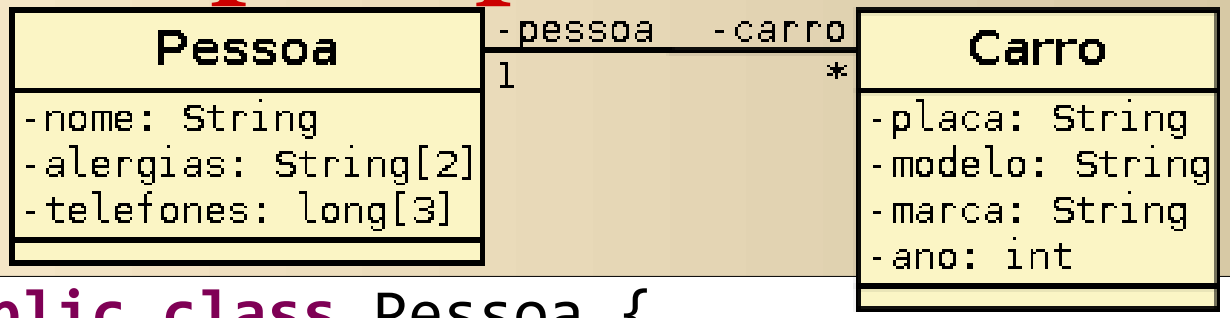
Multiplicidade

- **Conjunto** → coleção desordenada de objetos distintos (sem repetição).
- Set é abstrata com implentação em Árvore, Hash e Hash encadeado
- Coleção deve implementar interface Comparable
- Qualquer Set nunca fica cheio, mas iteração com a posição **i** exige iteração com **i-1** elementos

```
private Set<Carro> carro = new TreeSet<Carro>();
```

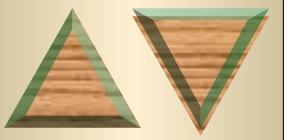


Exemplo 1-para-Muitos

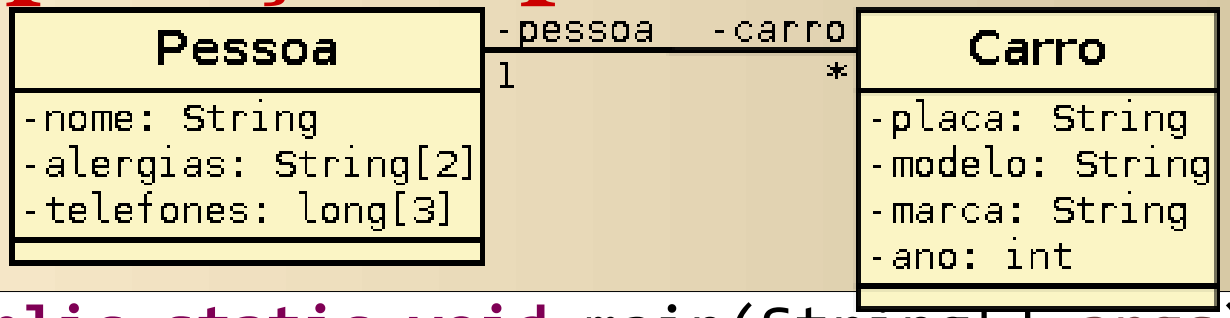


Multiplicidade

```
public class Pessoa {
    private String nome;
    private long telefones[] = new long[2];
    private String alergias[] = new String[3];
    private List<Carro> carro =
        new ArrayList<Carro>();
    //gets e sets }
public class Carro {
    private String placa;
    private String modelo;
    private String marca;
    private int ano;
    private Pessoa pessoa;     //gets e sets }
```

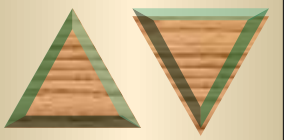


Aplicação 1-para-Muitos

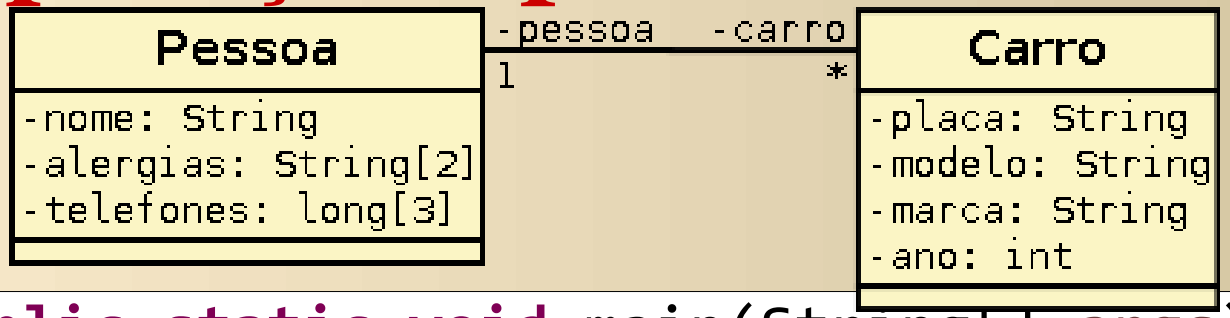


Multiplicidade

```
public static void main(String[] args) {
    Carro c1 = new Carro();
    c1.setPlaca("ABC1234");
    Carro c2 = new Carro();
    c2.setPlaca("XYZ9876");
    Pessoa p = new Pessoa();
    p.getCarro().add(c1);
    p.getCarro().add(c2);
    for(Carro c : p.getCarro()) {//para cada
        System.out.println(c.getPlaca());
    }
}
```

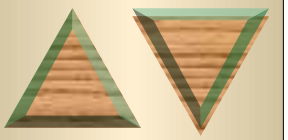


Aplicação 1-para-Muitos

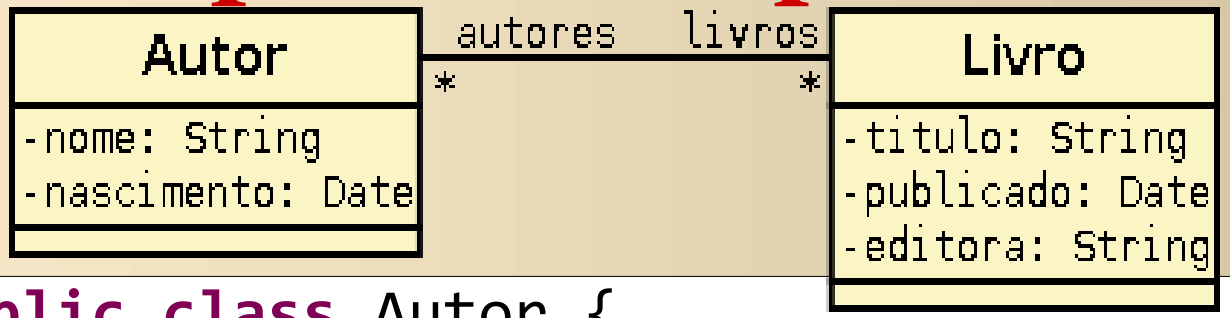


Multiplicidade

```
public static void main(String[] args) {
 Carro c1 = new Carro();
 c1.setPlaca("ABC1234");
 Carro c2 = new Carro();
 c2.setPlaca("XYZ9876");
 Pessoa p = new Pessoa();
 p.getCarro().add(c1);
 p.getCarro().add(c2);
 Iterator<Carro> it=p.getCarro().iterator();
 while (it.hasNext()) {//iterator
    Carro c = it.next();
    System.out.println(c.getPlaca());
 } }
```

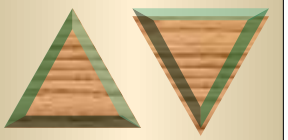


Exemplo Muitos-para-Muitos



Binária

```
public class Autor {
 private String nome;
 private Date nascimento;
 private List<Livro> livros =
    new ArrayList<Livro>();
 //gets e sets }
public class Livro {
 private String titulo;
 private Date publicacao;
 private String editora;
 private List<Autor> autores =
    new ArrayList<Autor>();
 //gets e sets }
```



- Binária
- Multiplicidade
- Direcionamento
- Reflexiva
- Particularidades

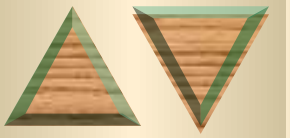
Associação entre Classes

Prof.Dr. Enzo Seraphim

Direcionamento

Direcinamento

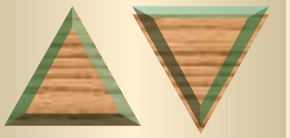
- Navegabilidade – necessidade de acesso por atributo de uma classe para outra e podem ser:
  - Bidirecional → ambos objetos possuem referência.
  - Unidirecional → classe que recebe a seta recebe também a referência.
- Para o armazenamento da informação basta a existência de uma referência
  - Necessidades de acesso podem impor referências mutuas



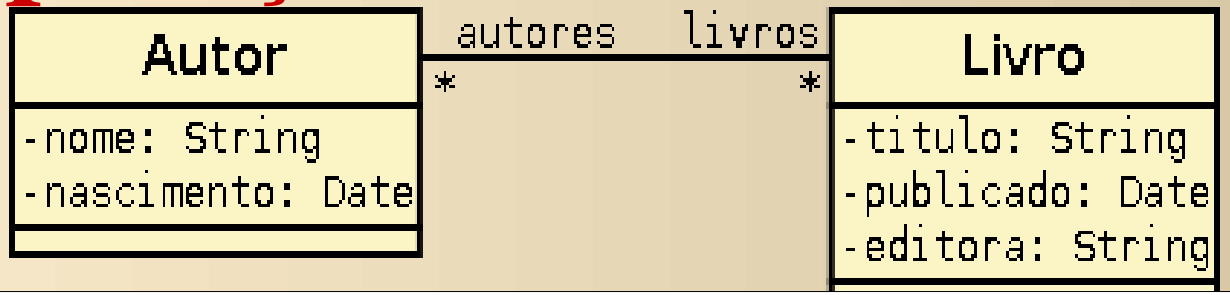
Associação Bidirecional

Direcinamento

- Representado por uma reta entre duas classes sem setas
- Atributos de referência entre as duas classes são repassados mutuamente.
- Atribuição ou remoção de referência de um dos lados exige atribuição ou remoção da referência do outro lado



Aplicação Bidirecional



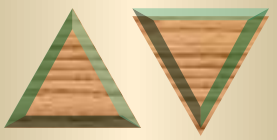
Binária

```
public static void main(String[] args) {
    Autor a = new Autor();
    a.setNome("Booch");
    Livro l = new Livro();
    l.setTitulo("UML");
    l.getAutores().add(a);
    //bidirecional
    a.getLivros().add(l);
}
```

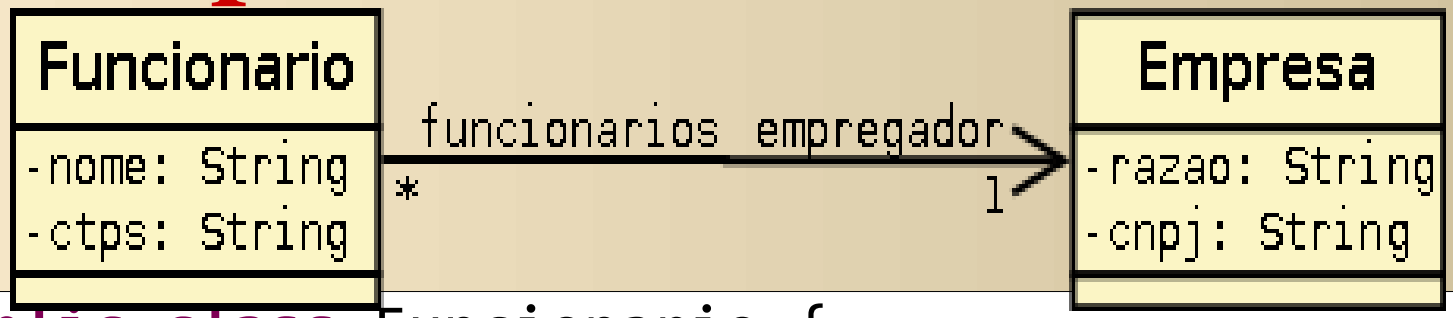
Associação unidirecional

Direcinamento

- Representado por uma reta entre duas classes, sendo que em uma das extremidades tem seta
- Seta indica que o atributo será repassado para a classe sem seta (apontamento)
- Atributos de referência na classe sem seta
- Em qual lado deve haver navegabilidade
  - Lado com multiplicidade de valor 1 possibilita enconomia de referências
- Sempre indique multiplicidade em ambos lados



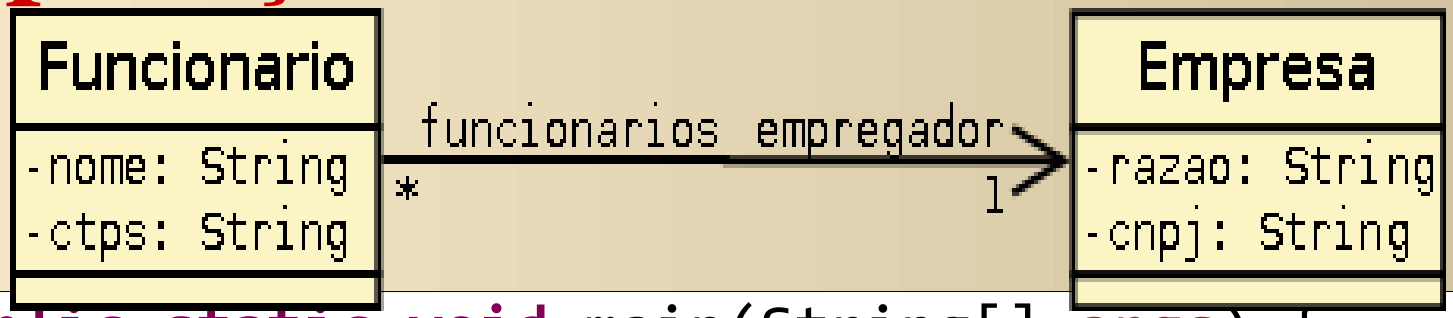
Exemplo Unidirecional



Binária

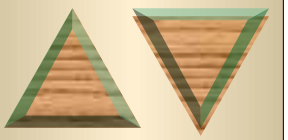
```
public class Funcionario {
    private String nome;
    private String ctps;
    private Empresa empregador;
    //gets e sets
}
public class Empresa {
    private String razao;
    private String cnpj;
    //gets e sets
}
```

Aplicação Unidirecional



Binária

```
public static void main(String[] args) {
    Empresa e = new Empresa();
    e.setRazao("UNIFEI");
    Funcionario f1 = new Funcionario();
    f1.setNome("Enzo");
    f1.setEmpregador(e);
    Funcionario f2 = new Funcionario();
    f2.setNome("Thatyana");
    f2.setEmpregador(e);
}
```



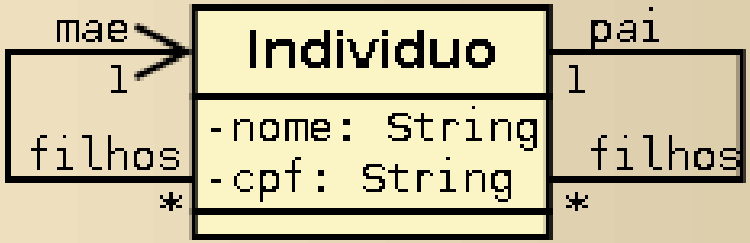
- Binária
- Multiplicidade
- Direcionamento
- Reflexiva
- Particularidades

Associação entre Classes

Prof.Dr. Enzo Seraphim

Associação reflexiva

- Associação de uma classe com ela própria
- Papéis são diferentes)




```
public class Individuo {
 private String nome;
 private String cpf;
 private Individuo pai;
 private Individuo mae;
 private List<Individuo> filhos =
     new ArrayList<Individuo>();
 //gets e sets
}
```

Reflexiva

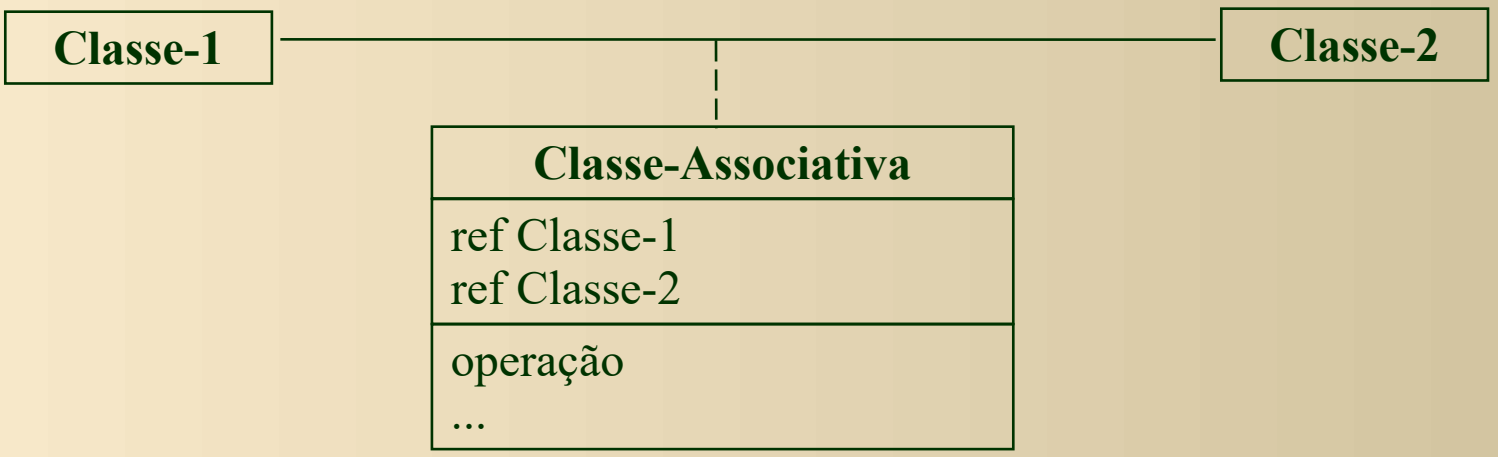
- Binária
- Multiplicidade
- Direcionamento
- Reflexiva
- Particularidades

Associação entre Classes

Prof.Dr. Enzo Seraphim

Classe Associativa

Particularidades

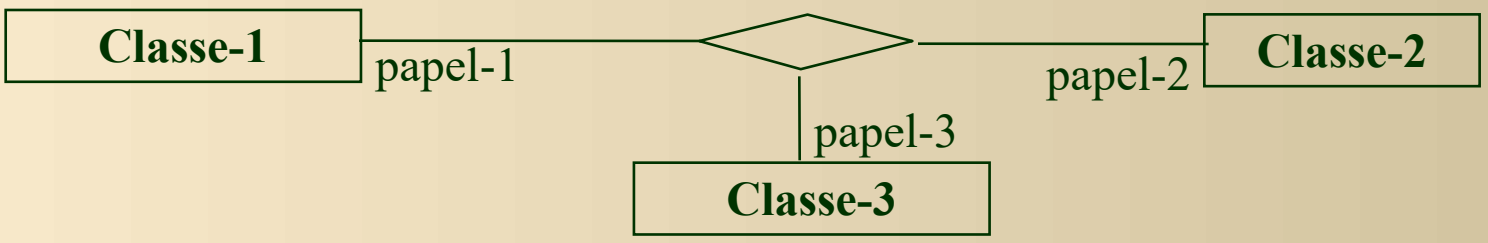


- Ocorrem para criar atributos em um relacionamento para modificação mínima do diagrama
- Pode ser substituida através referências explicitas entre Classe-1/Classe-2 e a Classe Associativa

Associações n-árias

Particularidades

- Notação



- Cada classe receberá um par de refefências das classes restantes
- Pode ser substituida através referências explicitas ou criação de classe N-ária que recebe todas referências

Prof.Dr.
Enzo Seraphim
seraphim@unifei.edu.br
IESTI/UNIFEI

Associação entre Classes