Departamento de Engenharia de Software
Centro de Educação Superior do Alto Vale do Itajaí - CEAVI
Universidade do Estado de Santa Catarina - UDESC

Programação orientada a objetos
Relacionamentos de associação

Relacionamentos entre classes

- Classes podem se relacionar entre si, definindo um vínculo entre os objetos dessas classes.

Exemplos

- Um **cliente** possui um **endereço**.
- Uma **empresa** é composta por **funcionários**.
- Uma **moto** é um tipo de **veículo**.
- Um **restaurante** possui **pratos**.
- Uma **correspondência** possui um **remetente** e um **destinatário**.

Relacionamentos entre classes

- **Associação:** conexão entre classes.
- **Agregação e composição:** especialização de uma associação onde um todo é relacionado com suas partes (relacionamento "parte-de").
- **Dependência:** um objeto depende de alguma forma de outro (relacionamento de utilização).
- **Herança (generalização):** um dos princípios da orientação a objetos, permite a reutilização, uma nova classe pode ser definida a partir de outra já existente.
- **Realização:** um contrato que a classe segue (obrigação).

Relacionamentos entre classes

- **Associação:** 
- **Agregação** 
- **Composição:** 
- **Dependência:** 
- **Herança (generalização):** 
- **Realização:** 

Associação

- Uma associação é uma conexão entre classes e representam as relações entre os objetos.
- Associações são representadas em um diagrama de classe através de uma linha conectando as classes associadas.
- Os dados podem fluir em uma ou em ambas as direções através do link.

Associação

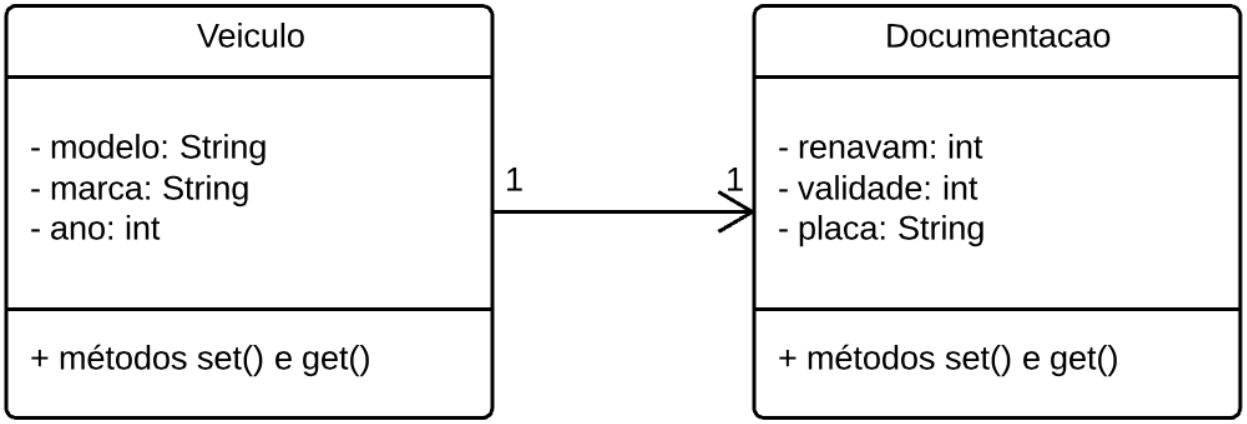
- Uma associação é uma conexão entre classes e representam as relações entre os objetos.
- Associações são representadas em um diagrama de classe através de uma linha conectando as classes associadas.
- Os dados podem fluir em uma ou em ambas as direções através do link.

Tipos de associação

- Associação com multiplicidade um (1).
  - Unidirecional e bidirecional.
- Associação com multiplicidade muitos (*).
  - Unidirecional e bidirecional.

Associação com multiplicidade um (1)

- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** um **veículo** possui uma **documentação**.
- Na associação **unidirecional,** a direção da associação deve ser especificada.

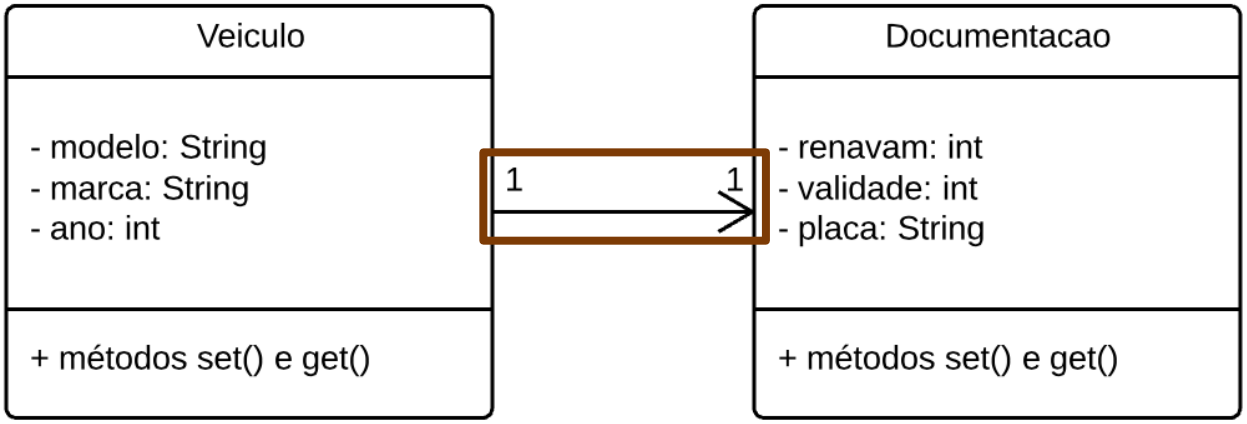


- Neste caso, a entidade **Veiculo** possui um atributo do tipo **Documentacao**.
- É possível navegar de **Veiculo** para **Documentacao** (navegabilidade).

Associação com multiplicidade um (1)

- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** um **veículo** po
- Na associação **unidireci** eve ser especificada.

A multiplicidade indica que um único veículo está vinculado a uma única documentação.

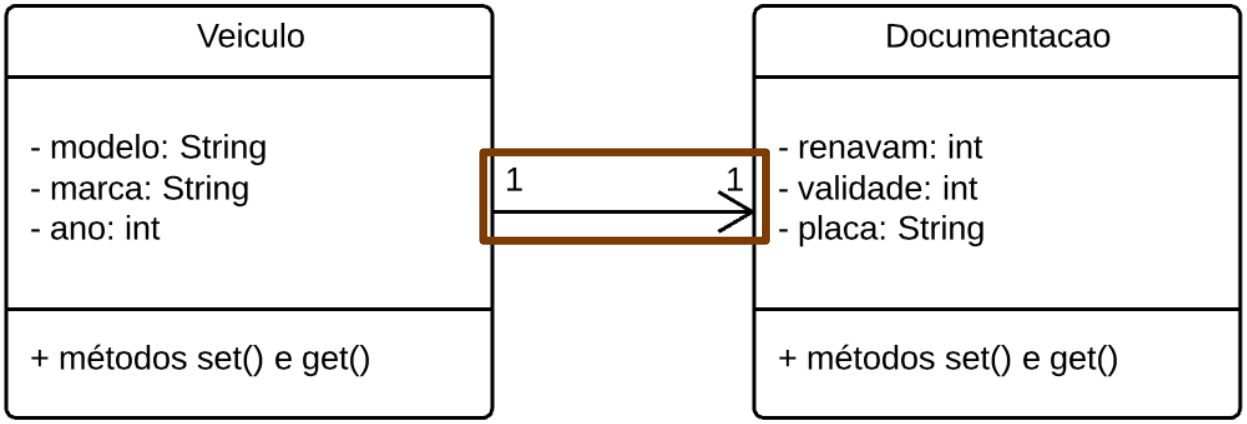


- Neste caso, a entidade **Veiculo** possui um atributo do tipo **Documentacao**.
- É possível navegar de **Veiculo** para **Documentacao** (navegabilidade).

Associação com multiplicidade um (1)

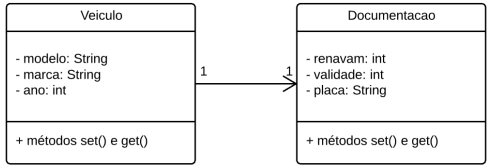
- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** um **veículo** po
- Na associação **unidireci** eve ser especificada.

Neste caso, um veículo deve estar obrigatoriamente vinculado a uma documentação.



- Neste caso, a entidade **Veiculo** possui um atributo do tipo **Documentacao**.
- É possível navegar de **Veiculo** para **Documentacao** (navegabilidade).

Associação com multiplicidade um (1)



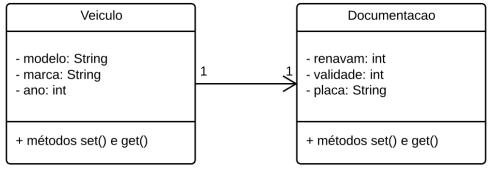
Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;
    private Documentacao doc;

    public Documentacao getDoc () {
        return doc;
    }

    public void setDoc (Documentacao doc) {
        this.doc = doc;
    }
}
```

```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;

    public int getRenavam() {
        return renavam;
    }

    public void setRenavam(int renavam) {
        this.renavam = renavam;
    }
}
```

```
public static void main(String[] args) {
    Veiculo carro = new Veiculo("Focus", "Ford", 2017);
    Documentacao doc = new Documentacao(512647522, 2018, "QWD-2573");
    carro.setDoc(doc);
    System.out.println("O veículo " + carro.getModelo() + " possui placa " +
                                                    carro.getDoc().getPlaca());
}
```

Associação com multiplicidade um (1)



Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;
    private Documentacao doc;

    public Documentacao getDoc () {
        return doc;
    }

    public void setDoc (Documentacao doc) {
        this.doc = doc;
    }
}
```

```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;

    public int getRenavam() {
        return renavam;
    }

    public void setRenavam(int renavam) {
        this.renavam = renavam;
    }
}
```

```
public static void main(Str
    Veiculo carro = new Vei
    Documentacao doc = new
    carro.setDoc(doc);
    System.out.println("O v
}
```

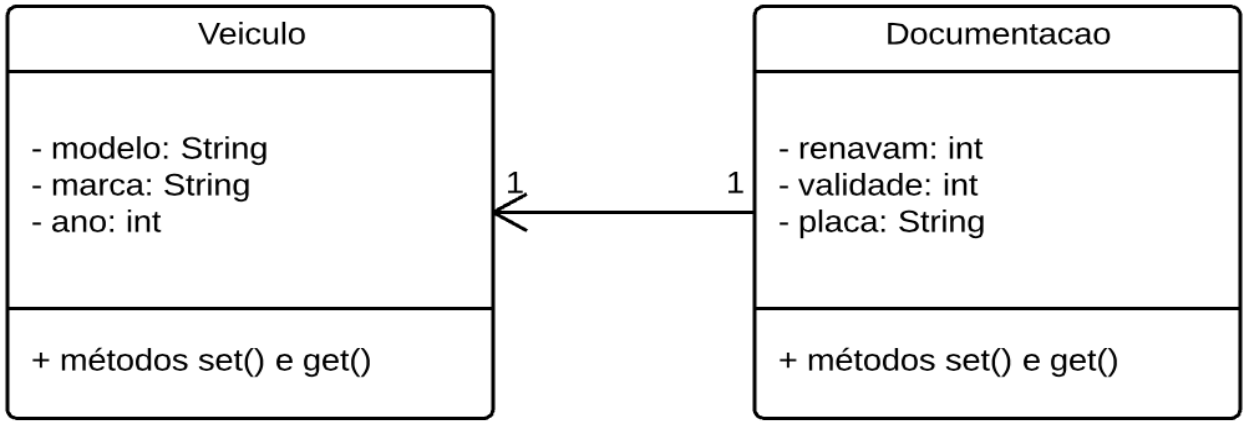
-2573");

ossui placa " +
carro.getDoc().getPlaca());

A vinculação é obrigatória, uma vez que a multiplicidade é exatamente 1.

Associação com multiplicidade um (1)

- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** uma **documentação** pertence a um **veículo**.
- Na associação **unidirecional,** a direção da associação deve ser especificada.

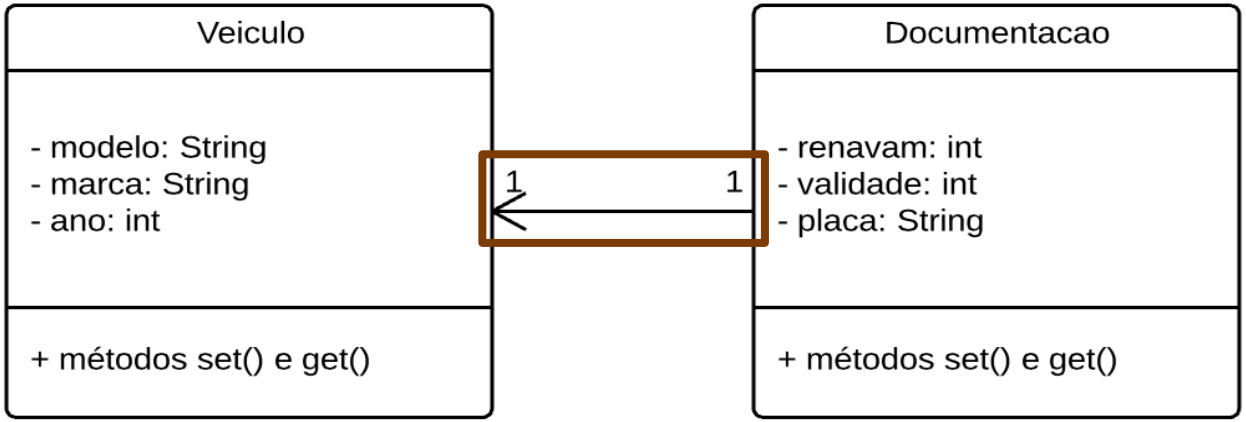


- Neste caso, a entidade **Documentacao** possui um atributo do tipo **Veiculo**.
- É possível navegar de **Documentacao** para **Veiculo** (navegabilidade).

Associação com multiplicidade um (1)

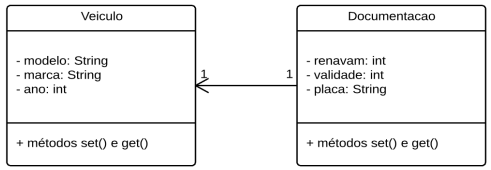
- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** uma **docume**
- Na associação **unidireci** eve ser especificada.

Neste caso, uma documentação deve estar obrigatoriamente vinculada a um veículo.



- Neste caso, a entidade **Documentacao** possui um atributo do tipo **Veiculo**.
- É possível navegar de **Documentacao** para **Veiculo** (navegabilidade).

Associação com multiplicidade um (1)



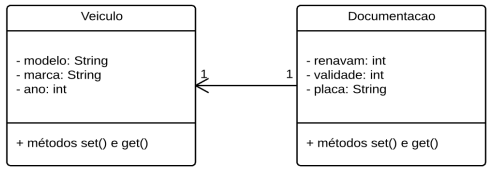
Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
}
```

```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;
    private Veiculo veiculo;

    public Veiculo getVeiculo() {
        return veiculo;
    }

    public void setVeiculo(Veiculo veiculo){
        this.veiculo = veiculo;
    }
}
```

```
public static void main(String[] args) {
    Veiculo carro = new Veiculo("Focus", "Ford", 2017);
    Documentacao doc = new Documentacao(512647522, 2018, "QWD-2573");
    doc.setVeiculo(carro);
    System.out.println("O documento " + doc.getRenavam() + " pertence ao veículo " +
                                                    doc.getVeiculo().getModelo());
}
```

Associação com multiplicidade um (1)



Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
}
```

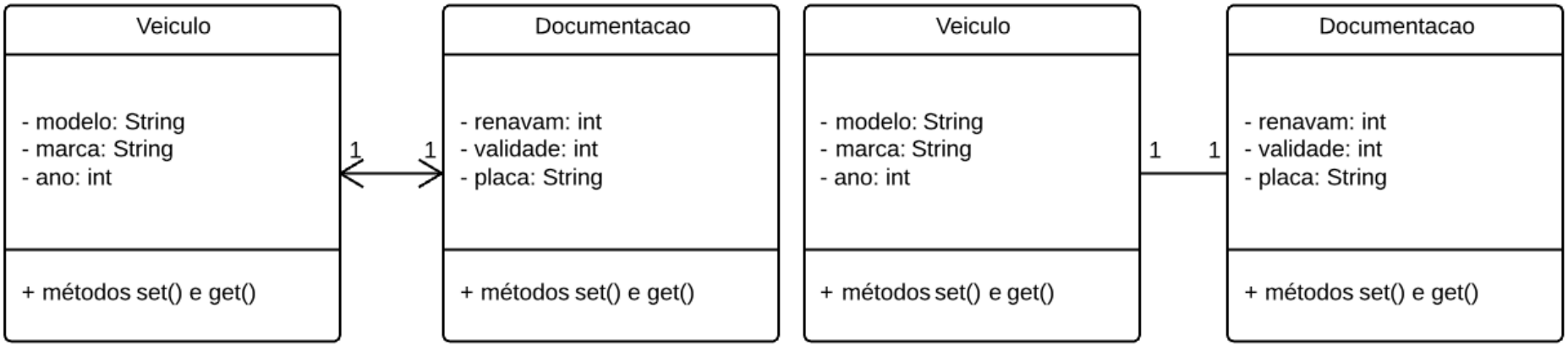
```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;
    private Veiculo veiculo;

    public Veiculo getVeiculo() {
        return veiculo;
    }

    public void setVeiculo(Veiculo veiculo){
        this.veiculo = veiculo;
    }
}
```

```
public static void main(Strin
    Veiculo carro = new Veicu
    Documentacao doc = new Do                    573");
    doc.setVeiculo(carro);
    System.out.println("O doc                    rtence ao veículo " +
                                                   doc.getVeiculo().getModelo());
}
```

A vinculação é obrigatória, uma vez que a multiplicidade é exatamente 1.

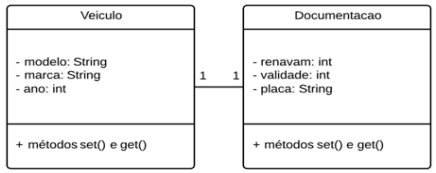
Associação com multiplicidade um (1)

- Ocorre quando um objeto está vinculado com apenas um objeto da outra classe.
- **Exemplo:** um **veículo** possui uma **documentação,** que pertence ao **veículo**.
- Na associação **bidirecional** não se especifica a direção, ou se especifica ambas.



- Neste caso, ambas as entidades possuem vínculo com a outra classe.
- É possível navegar de **Documentacao** para **Veiculo** e vice-versa (navegabilidade).

Associação com multiplicidade um (1)



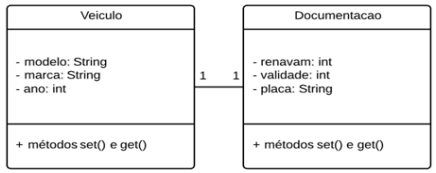
Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;
    private Documentacao doc;

    public Documentacao getDoc() {
        return doc;
    }

    public void setDoc(Documentacao doc) {
        this.doc = doc;
    }
}
```

```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;
    private Veiculo veiculo;

    public Veiculo getVeiculo() {
        return veiculo;
    }

    public void setVeiculo(Veiculo veiculo){
        this.veiculo = veiculo;
    }
}
```

```
public static void main(String[] args) {
    Veiculo carro = new Veiculo("Focus", "Ford", 2017);
    Documentacao doc = new Documentacao(512647522, 2018, "QWD-2573");
    doc.setVeiculo(carro);
    carro.setDoc(doc);
    System.out.println(doc.getRenavam() + " pertence ao " + doc.getVeiculo().getModelo());
    System.out.println(carro.getModelo() + " possui placa " + carro.getDoc().getPlaca());
}
```

Associação com multiplicidade um (1)



Implementação (métodos construtores e acessores omitidos)

```
public class Veiculo {
    private String modelo;
    private String marca;
    private int ano;
    private Documentacao doc;

    public Documentacao getDoc() {
        return doc;
    }

    public void setDoc(Documentacao doc) {
        this.doc = doc;
    }
}
```

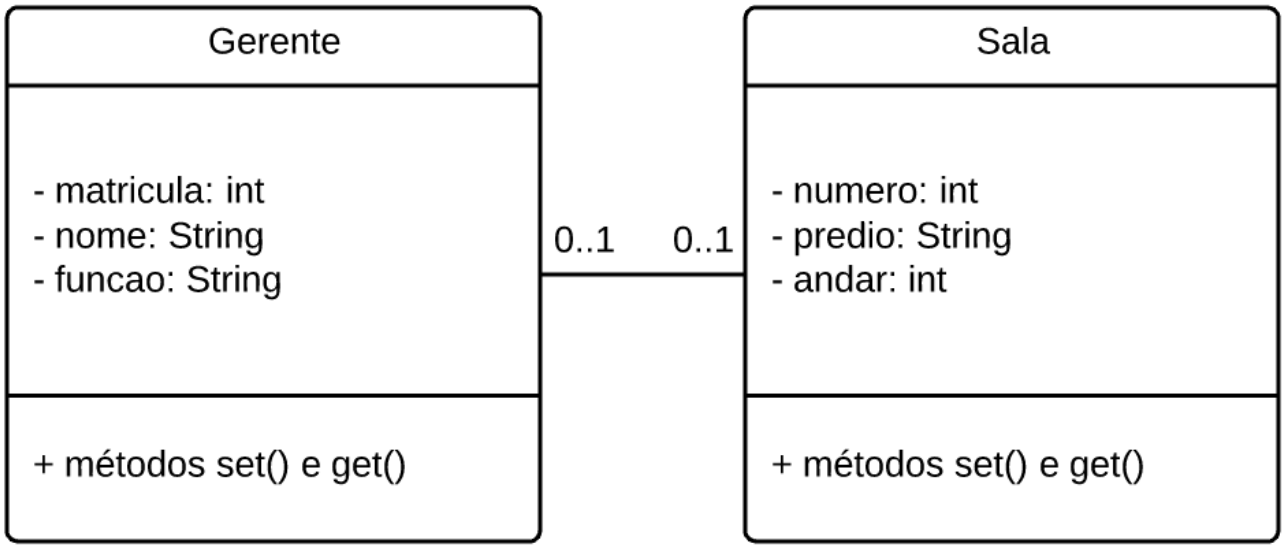
```
public class Documentacao {
    private int renavam;
    private int validade;
    private String placa;
    private Veiculo veiculo;

    public Veiculo getVeiculo() {
        return veiculo;
    }

    public void setVeiculo(Veiculo veiculo){
        this.veiculo = veiculo;
    }
}
```

```
public static void main(String[] args) {
    Veiculo carro = new Veicu
    Documentacao doc = new Do                                    573");
    doc.setVeiculo(carro);
    carro.setDoc(doc);
    System.out.println(doc.ge                                .getVeiculo().getModelo());
    System.out.println(carro.                                arro.getDoc().getPlaca());
}
```

A vinculação é obrigatória, uma vez que a multiplicidade é exatamente 1.

Associação com multiplicidade um (1)

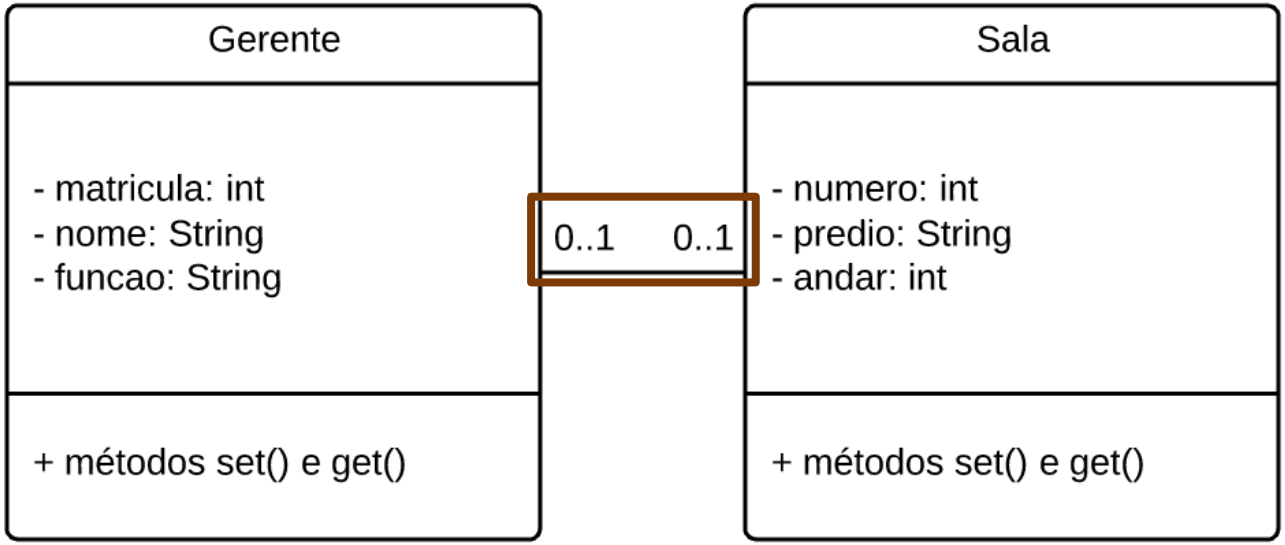
Exemplo: cada gerente da empresa possui uma sala de trabalho. Logo, a entidade **Gerente** possui um vínculo com a entidade **Sala** e vice-versa.



Atenção: os atributos da associação não são representados no diagrama.

Associação com multiplicidade um (1)

Exemplo: cada gerente da empresa possui uma sala de trabalho. Logo, a entidade **Gerente** possui um vínculo com a entidade **Sala** e vice-versa.



Neste caso, a vinculação não é obrigatória, pois a multiplicidade é 0 ou 1 (ex: pode haver um gerente sem sala).

Associação com multiplicidade um (1)

Exemplo – classe Gerente

```
public class Gerente {
    private int matricula;
    private String nome;
    private String funcao;
    private Sala sala;

    public Gerente() {}

    public Gerente(int matricula, String nome, String funcao) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
    }

    public Gerente(int matricula, String nome, String funcao, Sala sala) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
        setSala(sala);
    }

    public Sala getSala() { return sala; }

    public void setSala(Sala sala) {
        this.sala = sala;
        if(sala != null)
            sala.setGerente(this);
    }
}
```

Associação com multiplicidade um (1)

Exemplo – classe Gerente

```
public class Gerente {
    private int matricula;
    private String nome;
    private String funcao;
    private Sala sala;

    public Gerente() {}

    public Gerente(int matricula, String nome, String funcao) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
    }

    public Gerente(int matricula, String nome, String funcao, Sala sala) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
        setSala(sala);
    }

    public Sala getSala() { return sala; }

    public void setSala(Sala sala) {
        this.sala = sala;
        if(sala != null)
            sala.setGerente(this);
    }
}
```

Atributo da associação.

Associação com multiplicidade um (1)

Exemplo – classe Gerente

```
public class Gerente {
    private int matricula;
    private String nome;
    private String funcao;
    private Sala sala;

    public Gerente() {}

    public Gerente(int matricula, String nome, String funcao) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
    }

    public Gerente(int matricula, String nome, String funcao, Sala sala) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
        setSala(sala);
    }

    public Sala getSala() { return sala; }

    public void setSala(Sala sala) {
        this.sala = sala;
        if(sala != null)
            sala.setGerente(this);
    }
}
```

Métodos construtores sobrecarregados.

Associação com multiplicidade um (1)

Exemplo – classe Gerente

```
public class Gerente {
    private int matricula;
    private String nome;
    private String funcao;
    private Sala sala;

    public Gerente() {}

    public Gerente(int matricula, String nome, String funcao) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
    }

    public Gerente(int matricula, String nome, String funcao, Sala sala) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
        setSala(sala);
    }

    public Sala getSala() { return sala; }

    public void setSala(Sala sala) {
        this.sala = sala;
        if(sala != null)
            sala.setGerente(this);
    }
}
```

O método setSala não só atribui a sala ao gerente, mas atribui o gerente à sala (quando esta não é nula).

Associação com multiplicidade um (1)

Exemplo – classe Gerente

```
public class Gerente {
    private int matricula;
    private String nome;
    private String funcao;
    private Sala sala;

    public Gerente() {}

    public Gerente(int matricula, String nome, String funcao) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
    }

    public Gerente(int matricula, String nome, String funcao, Sala sala) {
        this.matricula = matricula;
        this.nome = nome;
        this.funcao = funcao;
        setSala(sala);
    }

    public Sala getSala() { return sala; }

    public void setSala(Sala sala) {
        this.sala = sala;
        if(sala != null)
            sala.setGerente(this);
    }
}
```

No construtor, o método **setSala** é chamado, já que ele executa comandos além da atribuição de sala.

Associação com multiplicidade um (1)

Exemplo – classe Sala

```
public class Sala {
    private int numero;
    private String predio;
    private int andar;
    private Gerente gerente;

    public Sala() {}

    public Sala(int numero, String predio, int andar) {
        this.numero = numero;
        this.predio = predio;
        this.andar = andar;
    }

    public Gerente getGerente() {
        return gerente;
    }

    public void setGerente(Gerente gerente) {
        if(this.gerente != null)
            this.gerente.setSala(null);

        this.gerente = gerente;
    }
}
```

Associação com multiplicidade um (1)

Exemplo – classe Sala

```
public class Sala {
    private int numero;
    private String predio;
    private int andar;
    private Gerente gerente;

    public Sala() {}

    public Sala(int numero, String predio, int andar) {
        this.numero = numero;
        this.predio = predio;
        this.andar = andar;
    }

    public Gerente getGerente() {
        return gerente;
    }

    public void setGerente(Gerente gerente) {
        if(this.gerente != null)
            this.gerente.setSala(null);

        this.gerente = gerente;
    }
}
```

Atributo da associação.

Associação com multiplicidade um (1)

Exemplo – classe Sala

```
public class Sala {
    private int numero;
    private String predio;
    private int andar;
    private Gerente gerente;

    public Sala() {}

    public Sala(int numero, String predio, int andar) {
        this.numero = numero;
        this.predio = predio;
        this.andar = andar;
    }

    public Gerente getGerente() {
        return gerente;
    }

    public void setGerente(Gerente gerente) {
        if(this.gerente != null)
            this.gerente.setSala(null);

        this.gerente = gerente;
    }
}
```

No método **setGerente**, caso exista um gerente já vinculado, este vínculo é removido, para então atribuir o novo gerente.

Associação com multiplicidade um (1)

Exemplo – classe Principal e classe Exemplo

```
public class Principal {

    public static void main(String[] args) {
        Exemplo exemplo = new Exemplo();
        exemplo.run();
    }

}
```

```
public class Exemplo {
    private List<Sala> salas = new ArrayList<Sala>();
    private List<Gerente> gerentes = new ArrayList<Gerente>();

    public void run() {
        insereSalas();
        insereGerentes();
        mostraRegistros();
    }

    //implementação dos métodos

}
```

Associação com multiplicidade um (1)

Exemplo – classe Principal e classe Exemplo

```
public class Principal {

    public static void main(String[] args) {
        Exemplo exemplo = new Exemplo();
        exemplo.run();
    }

}
```

```
public class Exemplo {
    private List<Sala> salas = new ArrayList<Sala>();
    private List<Gerente> gerentes = new ArrayList<Gerente>();

    public void run() {
        insereSalas();
        insereGerentes();
        mostraRegistros();
    }

    //implementação dos métodos

}
```

O método **main** apenas chama o método da classe responsável por implementar e executar os métodos (Exemplo).

Associação com multiplicidade um (1)

Exemplo – classe Exemplo – métodos de inserção

```
public void insereSalas() {
    Sala sala1 = new Sala(101, "Alpha", 1);
    Sala sala2 = new Sala(102, "Alpha", 1);
    Sala sala3 = new Sala(205, "Alpha", 2);
    Sala sala4 = new Sala(346, "Beta", 5);
    Sala sala5 = new Sala(12, "Gamma", 3);

    salas.add(sala1);
    salas.add(sala2);
    salas.add(sala3);
    salas.add(sala4);
    salas.add(sala5);
}

public void insereGerentes() {
    Gerente gerente1 = new Gerente(123456, "José da Silva", "Compras");
    Gerente gerente2 = new Gerente(654321, "Maria Pereira", "Vendas");
    Gerente gerente3 = new Gerente(123789, "João Assunção", "Marketing");
    Gerente gerente4 = new Gerente(987321, "Ana Maria Rodrigues", "Produção");

    gerente1.setSala(salas.get(0));
    gerente2.setSala(salas.get(1));
    gerente3.setSala(salas.get(2));
    gerente4.setSala(salas.get(0));

    gerentes.add(gerente1);
    gerentes.add(gerente2);
    gerentes.add(gerente3);
    gerentes.add(gerente4);
}
```

Associação com multiplicidade um (1)

Exemplo – classe Exemplo – métodos de inserção

```
public void insereSalas() {
    Sala sala1 = new Sala(101, "Alpha", 1);
    Sala sala2 = new Sala(102, "Alpha", 1);
    Sala sala3 = new Sala(205, "Alpha", 2);
    Sala sala4 = new Sala(346, "Beta", 5);
    Sala sala5 = new Sala(12, "Gamma", 3);

    salas.add(sala1);
    salas.add(sala2);
    salas.add(sala3);
    salas.add(sala4);
    salas.add(sala5);
}

public void insereGerentes() {
    Gerente gerente1 = new Gerente(123456, "José da Silva", "Compras");
    Gerente gerente2 = new Gerente(654321, "Maria Pereira", "Vendas");
    Gerente gerente3 = new Gerente(123789, "João Assunção", "Marketing");
    Gerente gerente4 = new Gerente(987321, "Ana Maria Rodrigues", "Produção");

    gerente1.setSala(salas.get(0));
    gerente2.setSala(salas.get(1));
    gerente3.setSala(salas.get(2));
    gerente4.setSala(salas.get(0));

    gerentes.add(gerente1);
    gerentes.add(gerente2);
    gerentes.add(gerente3);
    gerentes.add(gerente4);
}
```

Objetos da classe **Sala** são criados e armazenados em uma lista.

Associação com multiplicidade um (1)

Exemplo – classe Exemplo – métodos de inserção

```
public void insereSalas() {
    Sala sala1 = new Sala(101, "Alpha", 1);
    Sala sala2 = new Sala(102, "Alpha", 1);
    Sala sala3 = new Sala(205, "Alpha", 2);
    Sala sala4 = new Sala(346, "Beta", 5);
    Sala sala5 = new Sala(12, "Gamma", 3);

    salas.add(sala1);
    salas.add(sala2);
    salas.add(sala3);
    salas.add(sala4);
    salas.add(sala5);
}

public void insereGerentes() {
    Gerente gerente1 = new Gerente(123456, "José da Silva", "Compras");
    Gerente gerente2 = new Gerente(654321, "Maria Pereira", "Vendas");
    Gerente gerente3 = new Gerente(123789, "João Assunção", "Marketing");
    Gerente gerente4 = new Gerente(987321, "Ana Maria Rodrigues", "Produção");

    gerente1.setSala(salas.get(0));
    gerente2.setSala(salas.get(1));
    gerente3.setSala(salas.get(2));
    gerente4.setSala(salas.get(0));

    gerentes.add(gerente1);
    gerentes.add(gerente2);
    gerentes.add(gerente3);
    gerentes.add(gerente4);
}
```

Objetos da classe **Gerente** são criados e armazenados em uma lista.

Associação com multiplicidade um (1)

Exemplo – classe Exemplo – métodos de inserção

```
public void insereSalas() {
    Sala sala1 = new Sala(101, "Alpha", 1);
    Sala sala2 = new Sala(102, "Alpha", 1);
    Sala sala3 = new Sala(205, "Alpha", 2);
    Sala sala4 = new Sala(346, "Beta", 5);
    Sala sala5 = new Sala(12, "Gamma", 3);

    salas.add(sala1);
    salas.add(sala2);
    salas.add(sala3);
    salas.add(sala4);
    salas.add(sala5);
}

public void insereGerentes() {
    Gerente gerente1 = new Gerente(123456, "José da Silva", "Compras");
    Gerente gerente2 = new Gerente(654321, "Maria Pereira", "Vendas");
    Gerente gerente3 = new Gerente(123789, "João Assunção", "Marketing");
    Gerente gerente4 = new Gerente(987321, "Ana Maria Rodrigues", "Produção");

    gerente1.setSala(salas.get(0));
    gerente2.setSala(salas.get(1));
    gerente3.setSala(salas.get(2));
    gerente4.setSala(salas.get(0));

    gerentes.add(gerente1);
    gerentes.add(gerente2);
    gerentes.add(gerente3);
    gerentes.add(gerente4);
}
```

Na atribuição de salas aos gerentes, os vínculos são atualizados conforme implementação.

Associação com multiplicidade um (1)

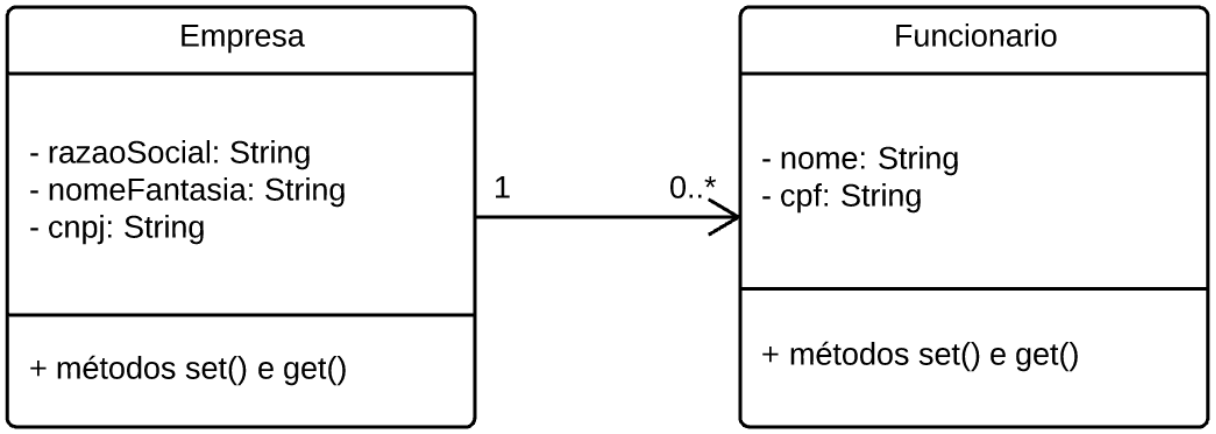
Exemplo – classe Exemplo – apresentação dos registros

```
public void mostraRegistros() {
    for(Gerente g : gerentes) {
        if(g.getSala() != null)
            System.out.println(g.getNome() + " trabalha na sala " + g.getSala().getNumero() +
                               " do prédio " + g.getSala().getPredio() + ".");
        else
            System.out.println(g.getNome() + " não possui uma sala de trabalho.");
    }

    System.out.println("");

    for(Sala s : salas) {
        if(s.getGerente() != null)
            System.out.println("Na sala " + s.getNumero() + " do prédio " + s.getPredio() +
                               " trabalha " + s.getGerente().getNome() + ".");
        else
            System.out.println("Na sala " + s.getNumero() + " do prédio " + s.getPredio() +
                               " não trabalha nenhum gerente.");
    }
}
```

```
José da Silva não possui uma sala de trabalho.
Maria Pereira trabalha na sala 102 do prédio Alpha.
João Assunção trabalha na sala 205 do prédio Alpha.
Ana Maria Rodrigues trabalha na sala 101 do prédio Alpha.

Na sala 101 do prédio Alpha trabalha Ana Maria Rodrigues.
Na sala 102 do prédio Alpha trabalha Maria Pereira.
Na sala 205 do prédio Alpha trabalha João Assunção.
Na sala 346 do prédio Beta não trabalha nenhum gerente.
Na sala 12 do prédio Gamma não trabalha nenhum gerente.
```

Associação com multiplicidade muitos (*)

- Ocorre quando um objeto está vinculado a vários objetos da outra classe.
- **Exemplo:** uma **empresa** possui vários **funcionários**.
- Na associação **unidirecional,** a direção da associação deve ser especificada.

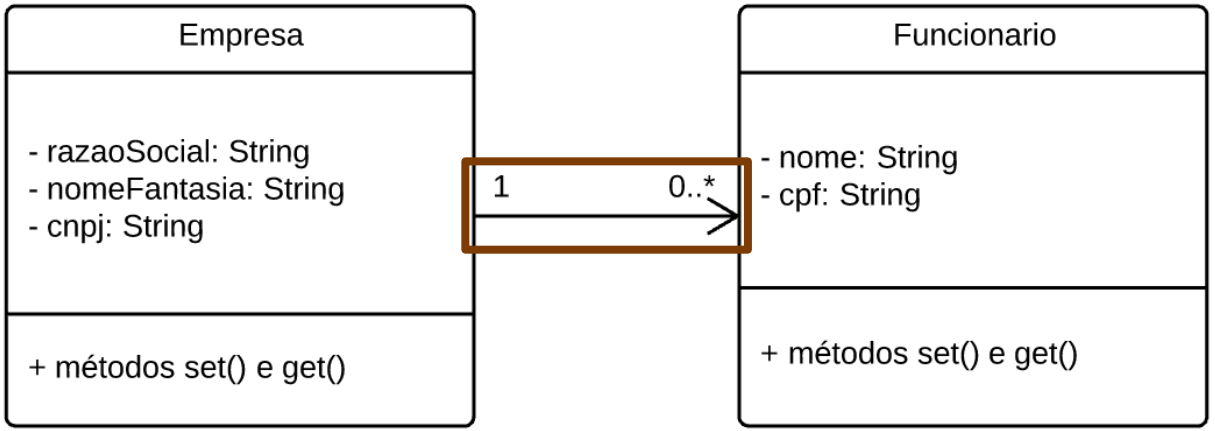


- Neste caso, a entidade **Empresa** possui uma lista de objetos do tipo **Funcionario**.
- É possível navegar de **Empresa** para **Funcionario** (navegabilidade).

Associação com multiplicidade muitos (*)

- Ocorre quando um objeto está vinculado a vários objetos da outra classe.
- **Exemplo:** uma **empresa**
- Na associação **unidireci** eve ser especificada.

A multiplicidade indica que uma empresa está vinculada a vários funcionários e um funcionário a apenas uma empresa.

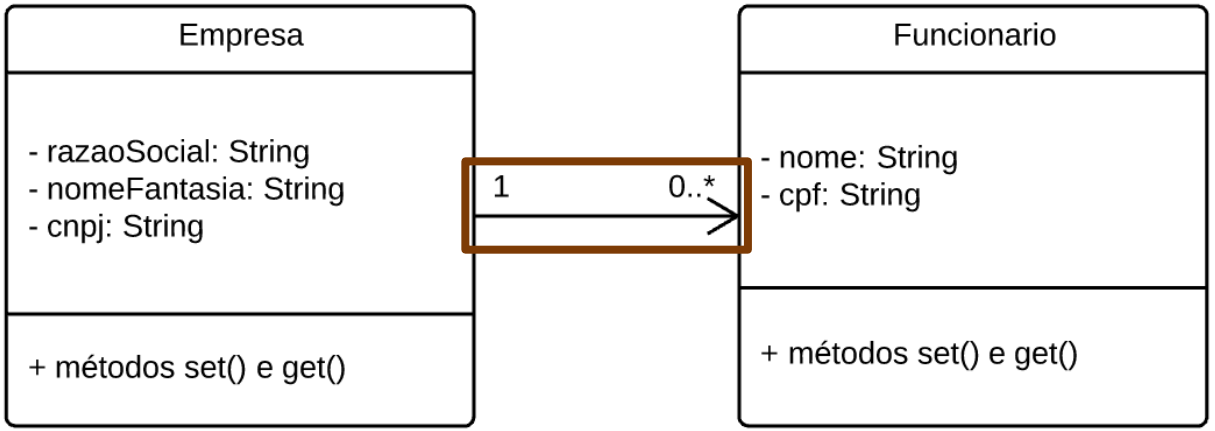


- Neste caso, a entidade **Empresa** possui uma lista de objetos do tipo **Funcionario**.
- É possível navegar de **Empresa** para **Funcionario** (navegabilidade).

Associação com multiplicidade muitos (*)

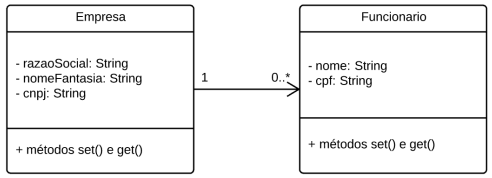
- Ocorre quando um objeto está vinculado a vários objetos da outra classe.
- **Exemplo:** uma **empresa**
- Na associação **unidireci** eve ser especificada.

A lista de funcionários pode ser vazia, pois a multiplicidade é **0..*** (não poderia ser vazia caso a multiplicidade fosse **1..***).



- Neste caso, a entidade **Empresa** possui uma lista de objetos do tipo **Funcionario**.
- É possível navegar de **Empresa** para **Funcionario** (navegabilidade).

Associação com multiplicidade muitos (*)



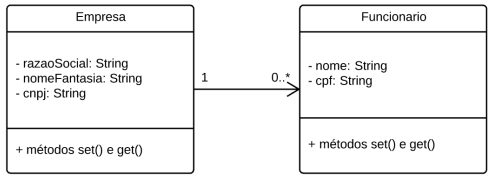
Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
  private String razaoSocial;
  private String nomeFantasia;
  private String cnpj;
  private List<Funcionario> funcionarios;

  public void addFuncionario(Funcionario f){
      this.funcionarios.add(f);
  }
}
```

```
public class Funcionario {
    private String nome;
    private String cpf;

    public String getNome() {
        return nome; }

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

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    e1.addFuncionario(f1);
    e1.addFuncionario(f2);
    e2.addFuncionario(f3);

    for(Funcionario f : e1.getFuncionarios())
        System.out.println(f.getNome() + " é funcionário na " + e1.getNomeFantasia());
}
```

Associação com multiplicidade muitos (*)



Implementação (métodos construtores e acessores omitidos)

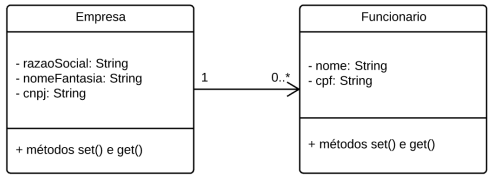
```
public class Empresa {
  private String razaoSocial;
  private String nomeFantasia;
  private String cnpj;
  private List<Funcionario> funcionarios;

  public void addFuncionario(Funcionario f){
      this.funcionarios.add(f);
  }
}
```

```
public class Funcionario {
    private String nome;
    private String cpf;

    public String getNome() {
        return nome; }

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

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    e1.addFuncionario(f1);
    e1.addFuncionario(f2);
    e2.addFuncionario(f3);

    for(Funcionario f : e1.getFuncionarios())
        System.out.println(f.getNome() + " é funcionário na "
}
```

A lista precisa ser inicializada no construtor (= new ArrayList<Funcionario).

Associação com multiplicidade muitos (*)



Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
  private String razaoSocial;
  private String nomeFantasia;
  private String cnpj;
  private List<Funcionario> funcionarios;

  public void addFuncionario(Funcionario f){
      this.funcionarios.add(f);
  }
}
```

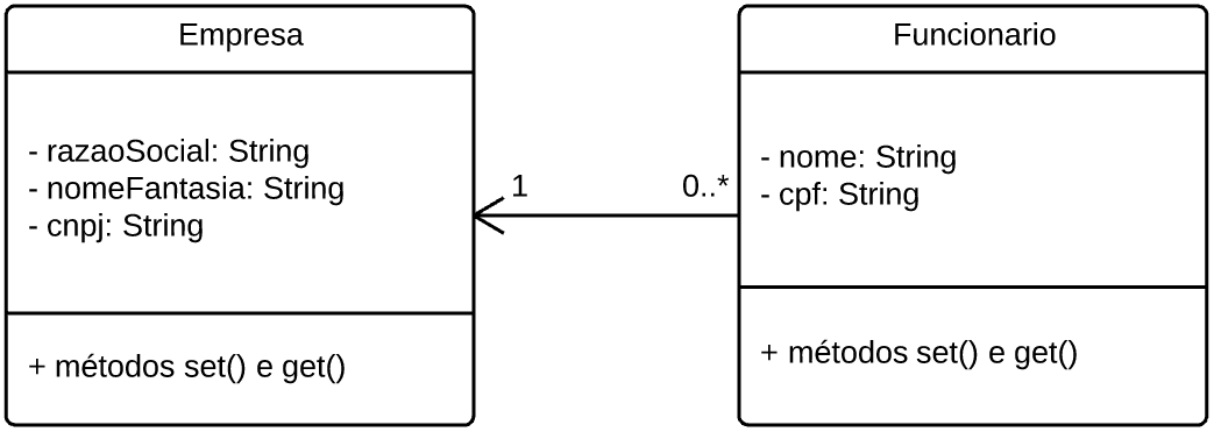
```
public class Funcionario {
    private String nome;
    private String cpf;

    public String getNome() {
        return nome; }

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

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    e1.addFuncionario(f1);
    e1.addFuncionario(f2);
    e2.addFuncionario(f3);

    for(Funcionario f : e1.getFuncionarios())
        System.out.println(f.getNome() + " é funcionário na "
}
```

Método que vincula os objetos, adicionando o funcionário à lista de funcionários da empresa.

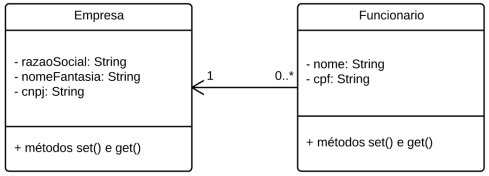
Associação com multiplicidade muitos (*)

- Ocorre quando um objeto está vinculado a vários objetos da outra classe.
- **Exemplo:** um **funcionário** pertence a uma **empresa**.
- Na associação **unidirecional,** a direção da associação deve ser especificada.



- Neste caso, a entidade **Funcionario** possui um vínculo com um objeto **Empresa**.
- É possível navegar de **Funcionario** para **Empresa** (navegabilidade).

Associação com multiplicidade muitos (*)



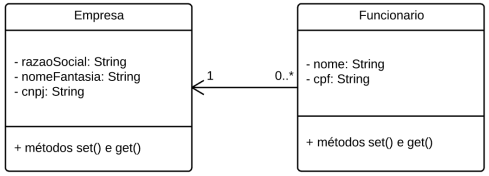
Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
    private String razaoSocial;
    private String nomeFantasia;
    private String cnpj;

    public String getRazaoSocial() {
        return razaoSocial; }

    public void setRazaoSocial(String r) {
        this.razaoSocial = r; } }
```

```
public class Funcionario {
  private String nome;
  private String cpf;
  private Empresa empresa;

  public Empresa getEmpresa() {
      return empresa; }

  public void setEmpresa(Empresa empresa) {
      this.empresa = empresa; } }
```

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    f1.setEmpresa(e1);
    f2.setEmpresa(e1);
    f3.setEmpresa(e2);
    System.out.println(f1.getNome() + " é func. na " + f1.getEmpresa().getNomeFantasia());
    System.out.println(f2.getNome() + " é func. na " + f2.getEmpresa().getNomeFantasia());
    System.out.println(f3.getNome() + " é func. na " + f3.getEmpresa().getNomeFantasia());
}
```

Associação com multiplicidade muitos (*)



Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
    private String razaoSocial;
    private String nomeFantasia;
    private String cnpj;

    public String getRazaoSocial() {
        return razaoSocial; }

    public void setRazaoSocial(String r) {
        this.razaoSocial = r; } }
```

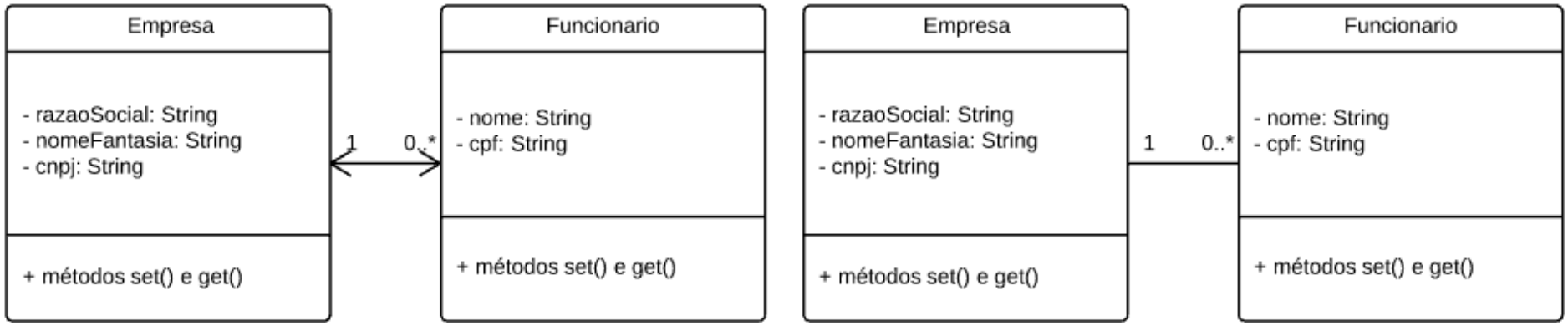
```
public class Funcionario {
  private String nome;
  private String cpf;
  private Empresa empresa;

  public Empresa getEmpresa() {
      return empresa; }

  public void setEmpresa(Empresa empresa) {
      this.empresa = empresa; } }
```

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    f1.setEmpresa(e1);
    f2.setEmpresa(e1);
    f3.setEmpresa(e2);
    System.out.println(f1.getNome() + " é func. na " + f1.get
    System.out.println(f2.getNome() + " é func. na " + f2.get
    System.out.println(f3.getNome() + " é func. na " + f3.get
}
```

A vinculação é obrigatória, uma vez que a multiplicidade é 1 (o funcionário possui vínculo com exatamente uma empresa).

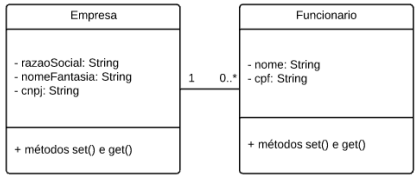
Associação com multiplicidade muitos (*)

- Ocorre quando um objeto está vinculado a vários objetos da outra classe.
- **Exemplo:** uma **empresa** possui vários **funcionários,** que pertencem à **empresa**.
- Na associação **bidirecional** não se especifica a direção, ou se especifica ambas.



- Neste caso, ambas as entidades possuem vínculo com a outra classe.
- É possível navegar de **Empresa** para **Funcionario** e vice-versa (navegabilidade).

Associação com multiplicidade muitos (*)



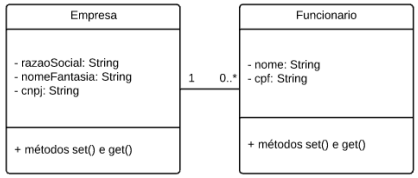
Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
  private String razaoSocial;
  private String nomeFantasia;
  private String cnpj;
  private List<Funcionario> funcionarios;

  public void addFuncionario(Funcionario f){
      this.funcionarios.add(f);
  }
}
```

```
public class Funcionario {
    private String nome;
    private String cpf;
    private Empresa empresa;

    public void setEmpresa(Empresa empresa){
        this.empresa = empresa;
        empresa.addFuncionario(this);
    }
}
```

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    f1.setEmpresa(e1);
    f2.setEmpresa(e1);
    f3.setEmpresa(e2);

    for(Funcionario f : e1.getFuncionarios())
        System.out.println(f.getNome() + " trabalha na " + e1.getNomeFantasia());
}
```

Associação com multiplicidade muitos (*)



Implementação (métodos construtores e acessores omitidos)

```
public class Empresa {
  private String razaoSocial;
  private String nomeFantasia;
  private String cnpj;
  private List<Funcionario> funcionarios;

  public void addFuncionario(Funcionario f){
      this.funcionarios.add(f);
  }
}
```

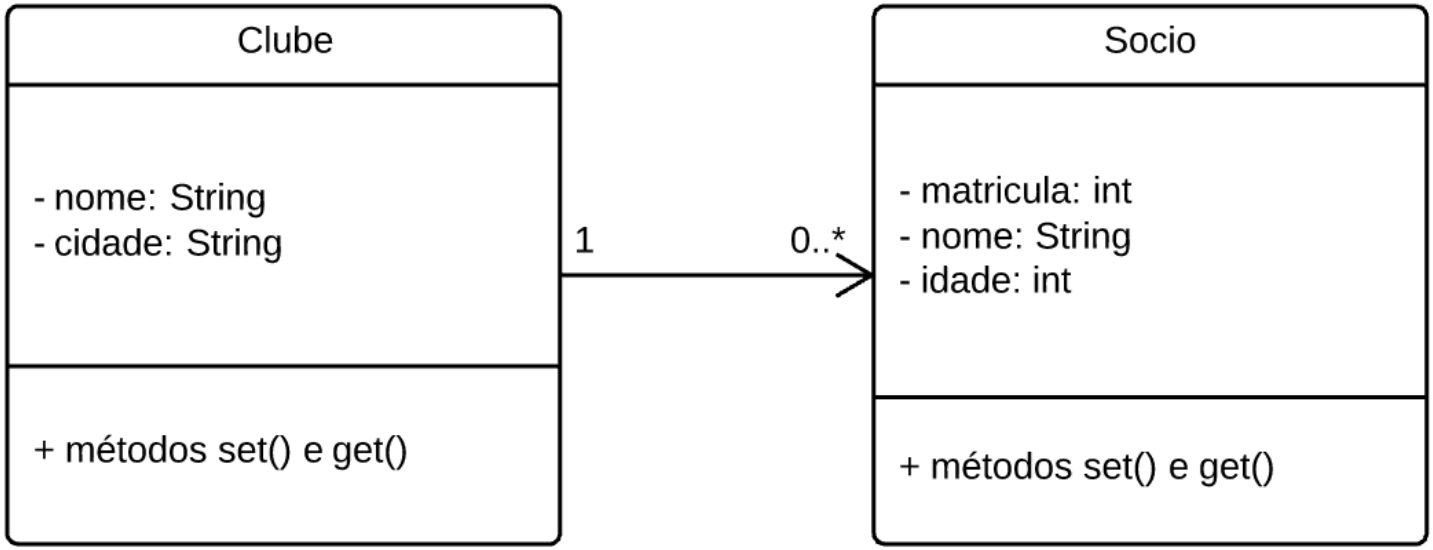
```
public class Funcionario {
    private String nome;
    private String cpf;
    private Empresa empresa;

    public void setEmpresa(Empresa empresa){
        this.empresa = empresa;
        empresa.addFuncionario(this);
    }
}
```

```
public static void main(String[] args) {
    Funcionario f1 = new Funcionario("José da Silva", "012.541.379-33");
    Funcionario f2 = new Funcionario("Maria Pereira", "062.411.632-12");
    Funcionario f3 = new Funcionario("Pedro Ferreira", "178.219.475-25");
    Empresa e1 = new Empresa("Empresa 1 LTDA", "Empresa 1", "123.456.789/0001-01");
    Empresa e2 = new Empresa("Empresa 2 LTDA", "Empresa 2", "123.456.789/0001-02");
    f1.setEmpresa(e1);
    f2.setEmpresa(e1);
    f3.setEmpresa(e2);

    for(Funcionario f : e1.getFuncionarios())
        System.out.println(f.getNome() + " trabalha na " + e1
}
```

O método **setEmpresa** faz o vínculo bidirecional, associando a empresa na instância do funcionário.

Associação com multiplicidade um (1)

Exemplo: um **clube** possui vários **sócios**. Logo, a entidade **Clube** possui vínculos com cada objeto da entidade **Socio** (lista).



Atenção: os atributos da associação não são representados no diagrama.

Associação com multiplicidade muitos (*)

Exemplo – classe Socio

```
public class Socio {
    private int matricula;
    private String nome;
    private int idade;

    public Socio() {}

    public Socio(int matricula, String nome, int idade) {
        this.matricula = matricula;
        this.nome = nome;
        this.idade = idade;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }
}
```

Associação com multiplicidade muitos (*)

Exemplo – classe Clube

```
public class Clube {
    private String nome;
    private String cidade;
    private List<Socio> socios;

    public Clube() { socios = new ArrayList<Socio>(); }

    public Clube(String nome, String cidade) {
        this.nome = nome;
        this.cidade = cidade;
        this.socios = new ArrayList<Socio>();
    }

    public void addSocio(Socio socio) {
        this.socios.add(socio);
    }

    public boolean removeSocio(int matricula) {
        for(Socio s : socios) {
            if(s.getMatricula() == matricula) {
                this.socios.remove(s);
                return true;
            }
        }
        return false;
    }

    public List<Socio> getSocios() { return socios; }

    public void setSocios(List<Socio> socios) { this.socios = socios; }
}
```

Associação com multiplicidade muitos (*)

Exemplo – classe Clube

```
public class Clube {
    private String nome;
    private String cidade;
    private List<Socio> socios;

    public Clube() { socios = new ArrayList<Socio>(); }

    public Clube(String nome, String cidade) {
        this.nome = nome;
        this.cidade = cidade;
        this.socios = new ArrayList<Socio>();
    }

    public void addSocio(Socio socio) {
        this.socios.add(socio);
    }

    public boolean removeSocio(int matricula) {
        for(Socio s : socios) {
            if(s.getMatricula() == matricula) {
                this.socios.remove(s);
                return true;
            }
        }
        return false;
    }

    public List<Socio> getSocios() { return socios; }

    public void setSocios(List<Socio> socios) { this.socios = socios; }
}
```

Lista que implementa a associação.

Associação com multiplicidade muitos (*)

Exemplo – classe Clube

```
public class Clube {
    private String nome;
    private String cidade;
    private List<Socio> socios;

    public Clube() { socios = new ArrayList<Socio>(); }

    public Clube(String nome, String cidade) {
        this.nome = nome;
        this.cidade = cidade;
        this.socios = new ArrayList<Socio>();
    }

    public void addSocio(Socio socio) {
        this.socios.add(socio);
    }

    public boolean removeSocio(int matricula) {
        for(Socio s : socios) {
            if(s.getMatricula() == matricula) {
                this.socios.remove(s);
                return true;
            }
        }
        return false;
    }

    public List<Socio> getSocios() { return socios; }

    public void setSocios(List<Socio> socios) { this.socios = socios; }
}
```

Método para adicionar um novo sócio à lista de sócios do clube.

Associação com multiplicidade muitos (*)

Exemplo – classe Clube

```
public class Clube {
    private String nome;
    private String cidade;
    private List<Socio> socios;

    public Clube() { socios = new ArrayList<Socio>(); }

    public Clube(String nome, String cidade) {
        this.nome = nome;
        this.cidade = cidade;
        this.socios = new ArrayList<Socio>();
    }

    public void addSocio(Socio socio) {
        this.socios.add(socio);
    }

    public boolean removeSocio(int matricula) {
        for(Socio s : socios) {
            if(s.getMatricula() == matricula) {
                this.socios.remove(s);
                return true;
            }
        }
        return false;
    }

    public List<Socio> getSocios() { return socios; }

    public void setSocios(List<Socio> socios) { this.socios = socios; }
}
```

Método para remover um sócio (buscado pela matrícula) da lista de sócios do clube.

Associação com multiplicidade muitos (*)

Exemplo – classe Principal e classe Exemplo

```
public class Principal {

    public static void main(String[] args) {
        Exemplo exemplo = new Exemplo();
        exemplo.run();
    }

}
```

```
public class Exemplo {
    private List<Socio> socios = new ArrayList<Socio>();
    private List<Clube> clubes = new ArrayList<Clube>();

    public void run() {
        insereRegistros();
        removeSocio();
        mostraRegistros();
    }

    //implementação dos métodos

}
```

O método **main** apenas chama o método da classe responsável por implementar e executar os métodos (Exemplo).

Associação com multiplicidade muitos (*)

Exemplo – classe Exemplo – métodos de manipulação

```
private void insereRegistros() {
    Socio s1 = new Socio(123456, "Maria da Rosa", 45);
    Socio s2 = new Socio(654321, "José da Silva", 25);
    Socio s3 = new Socio(147852, "Ana Lúcia da Silva", 30);
    Socio s4 = new Socio(369852, "Pedro Ferreira", 28);
    socios.add(s1);
    socios.add(s2);
    socios.add(s3);
    socios.add(s4);

    Clube c1 = new Clube("Clube ABC", "Ibirama");
    Clube c2 = new Clube("Clube XYZ", "Blumenau");
    clubes.add(c1);
    clubes.add(c2);

    c1.addSocio(s1);
    c1.addSocio(s2);
    c2.addSocio(s3);
    c2.addSocio(s4);
}

public void removeSocio() {
    Clube c = clubes.get(0);
    Socio s = socios.get(0);
    if(c.removeSocio(s.getMatricula())) {
       System.out.println("Sócio [" + s.getMatricula() + "] removido do clube " + c.getNome());
       socios.remove(s);
    } else
       System.out.println("Sócio [" + s.getMatricula() + "] não pertence ao " + c.getNome());
}
```

Associação com multiplicidade muitos (*)

Exemplo – classe Exemplo – métodos de manipulação

```
private void insereRegistros() {
    Socio s1 = new Socio(123456, "Maria da Rosa", 45);
    Socio s2 = new Socio(654321, "José da Silva", 25);
    Socio s3 = new Socio(147852, "Ana Lúcia da Silva", 30);
    Socio s4 = new Socio(369852, "Pedro Ferreira", 28);
    socios.add(s1);
    socios.add(s2);
    socios.add(s3);
    socios.add(s4);

    Clube c1 = new Clube("Clube ABC", "Ibirama");
    Clube c2 = new Clube("Clube XYZ", "Blumenau");
    clubes.add(c1);
    clubes.add(c2);

    c1.addSocio(s1);
    c1.addSocio(s2);
    c2.addSocio(s3);
    c2.addSocio(s4);
}

public void removeSocio() {
    Clube c = clubes.get(0);
    Socio s = socios.get(0);
    if(c.removeSocio(s.getMatricula())) {
       System.out.println("Sócio [" + s.getMatricula() + "] removido do clube " + c.getNome());
       socios.remove(s);
    } else
       System.out.println("Sócio [" + s.getMatricula() + "] não pertence ao " + c.getNome());
}
```

Criação de sócios, criação de clubes e vinculação dos sócios aos clubes (e armazenamento).

Associação com multiplicidade muitos (*)

Exemplo – classe Exemplo – métodos de manipulação

```
private void insereRegistros() {
    Socio s1 = new Socio(123456, "Maria da Rosa", 45);
    Socio s2 = new Socio(654321, "José da Silva", 25);
    Socio s3 = new Socio(147852, "Ana Lúcia da Silva", 30);
    Socio s4 = new Socio(369852, "Pedro Ferreira", 28);
    socios.add(s1);
    socios.add(s2);
    socios.add(s3);
    socios.add(s4);

    Clube c1 = new Clube("Clube ABC", "Ibirama");
    Clube c2 = new Clube("Clube XYZ", "Blumenau");
    clubes.add(c1);
    clubes.add(c2);

    c1.addSocio(s1);
    c1.addSocio(s2);
    c2.addSocio(s3);
    c2.addSocio(s4);
}

public void removeSocio() {
    Clube c = clubes.get(0);
    Socio s = socios.get(0);
    if(c.removeSocio(s.getMatricula())) {
       System.out.println("Sócio [" + s.getMatricula() + "] removido do clube " + c.getNome());
       socios.remove(s);
    } else
       System.out.println("Sócio [" + s.getMatricula() + "] não pertence ao " + c.getNome());
}
```

O método **removeSocio** retorna verdadeiro quando o sócio é removido e falso quando ele não é encontrado.

Associação com multiplicidade muitos (*)

Exemplo – classe Exemplo – apresentação dos registros

```
private void mostraRegistros() {
    for(Clube c : clubes) {
        System.out.println("Sócios do clube " + c.getNome());
        for(Socio s : c.getSocios()) {
            System.out.println("\t" + s.getNome() + " [" + s.getMatricula() + "]");
        }
        System.out.println("");
    }
}
```

```
Sócios do clube Clube ABC
     José da Silva [654321]

Sócios do clube Clube XYZ
     Ana Lúcia da Silva [147852]
     Pedro Ferreira [369852]
```

Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (http://www.tutorialspoint.com/java).