Relacionamento entre Objetos

- Objetos não existem isolados
  - Um objeto pode ser formado por outros (Composição)
  - Um objeto pode conter outros (Agregação)
  - Objetos usam outros objetos (Associação)
- Um programa OO possui vários objetos que interagem entre si
- Modelagem define quais objetos e relacionamentos usamos em um programa

Relacionamento entre Objetos

- Objetos possuem relacionamentos:

- Composição
  - Um objeto pode **ser formado por outros** objetos
  - Livro, pedido

- Agregação
  - Um objeto pode **conter outros objetos**
  - Casa, carro

- Associação
  - Um objeto usa outro objeto
  - Trem usa Estrada de Ferro

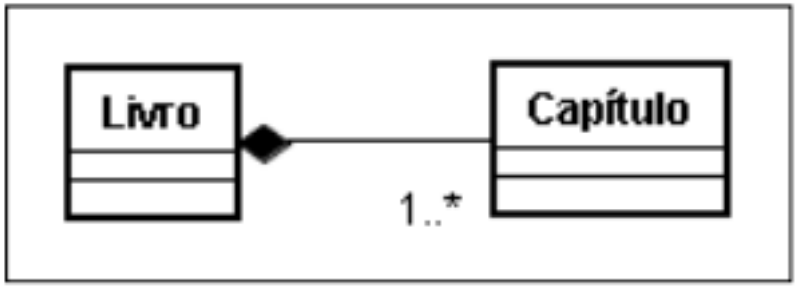
Composição

Um livro é composto de capítulos

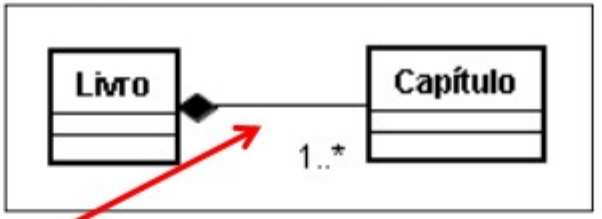
- Capítulo é parte essencial de livro
  - Se não existir capítulo, não existe livro
  - Capítulo não existe fora de livro

Linha com losângulo preenchido na classe "dominante"

- Livro é composto de 1 ou mais capítulos



Composição



Os atributos são derivados dos relacionamentos. Não existem no diagrama

```
public class Capitulo {

  private Livro livro;

  /* Definição da classe Capítulo */
}
```

```
public class Livro {

  private Capitulo[] capitulos;

  public Livro(int qtdCapitulo){
    capitulos = new Capitulo[qtdCapitulo];
  }
}
```

Referência pode ou não ser bidirecional. Capitulo não precisa ter o atributo livro

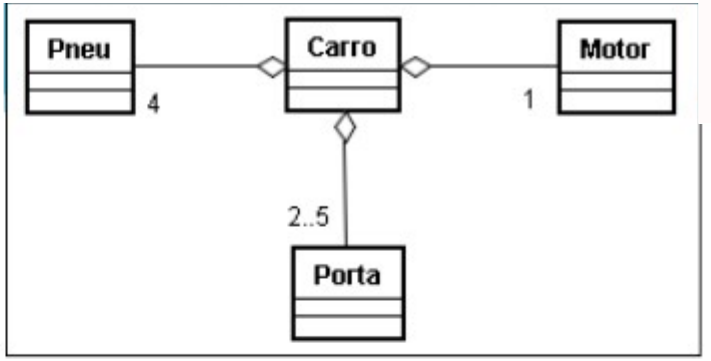
Fonte: Prof. Bruno Gomes

Agregação

```
public class Carro {

   private Motor motor;
   private Porta portas[];
   private Pneu pneus[];
   /* ... */
}
```

Pode ser implementado de mais de uma forma

```
public class Carro {

   private Motor motor;
   private Porta portas[];
   private Pneu p1, p2, p3, p4;
   /* ... */
}
```




```
public class Motor {
   /* ... */
}
```

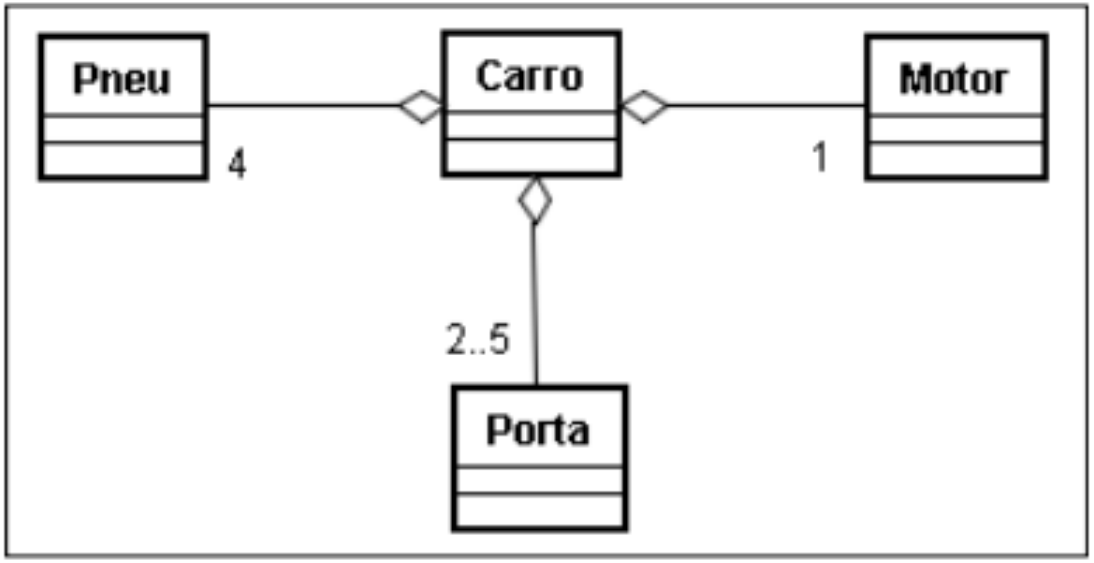
```
public class Pneu{
   /* ... */
}
```

```
public class Porta{
   /* ... */
}
```

Fonte: Prof. Bruno Gomes

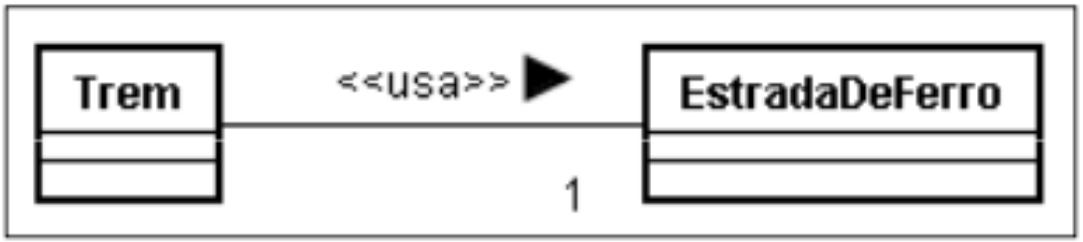
Agregação

- Carro possui Pneu, Motor e Porta
  - Não são partes essenciais do carro
    - Retirando as portas um carro continua sendo um carro
    - Pneus/portas existem como objetos independentes
- Linha com losângulo vazio na classe “dominante”



Associação

- Objetos que usam outros objetos
  - Podem ser implementados como atributos




```
public class Trem {
        private EstradaDeFerro estradaDeFerro;
        /* ... */
        public void definirEstrada(EstradaDeFerro estradaDeFerro){
                this.estradaDeFerro = estradaDeFerro;
        }
}
```

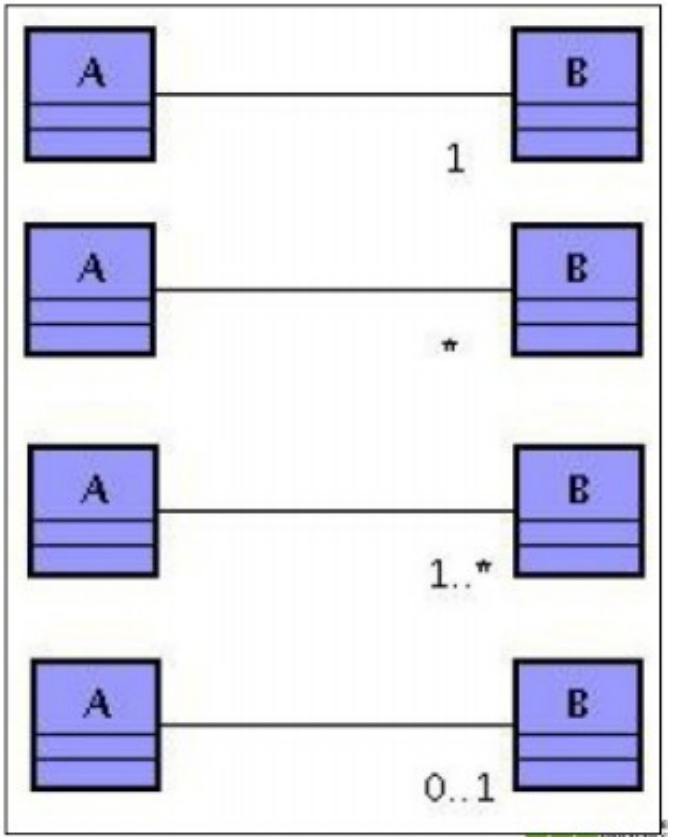
Fonte: Prof. Bruno Gomes

Relacionamentos entre Objetos

- Composição, Agregação e Associação
  - Mesma forma de implementar
  - Muda apenas o **conceito**
    - Comportamento diferente
  - Muito comum usar apenas notação da associação
    - Sem o losângulo
    - Composições e Agregações são “tipos” de associações
  - Representam relacionamento **“tem um”**
    - Carro “tem uma” roda
    - Livro “tem um” capitulo
    - Trem “tem uma” estrada de ferrro

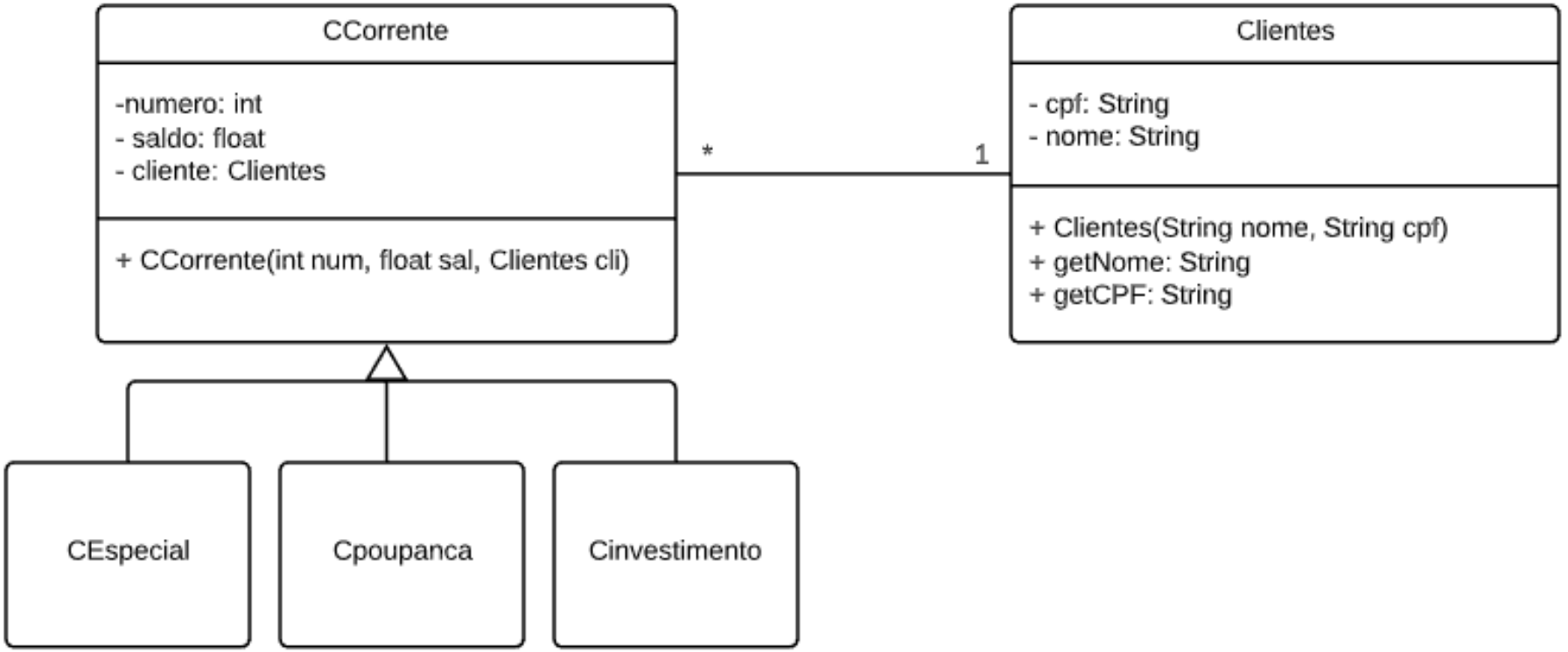
Multiplicidade

- Indica quantidade de objetos referenciados
- Principais:
  - A possui exatamente 1 B
  - A possui vários B
  - A possui 1 ou mais B
  - A possui 0 ou 1 B



Fonte: Prof. Bruno Gomes

Associação Contas Corrente e Cliente



Associações

- Exemplo em Conta Corrente – Criação da classe Clientes

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

public Clientes(String nome, String cpf){
    this.nome = nome;
    this.cpf = cpf;
}

public String getNome(){
    return this.nome;
}

public String getCpf(){
    return this.cpf;
}
```

Associações

- Exemplo em Conta Corrente – Alteração da classe CCorrente

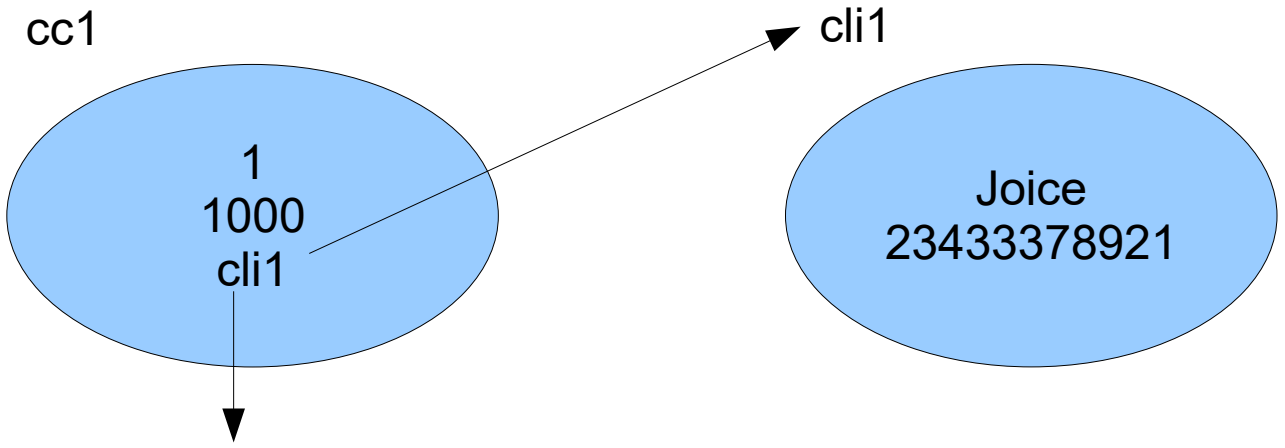
```
class CCorrente {

    private int numero;
    private double saldo;
    private Clientes cliente;
```

```
 public CCorrente (int num, float sal,Clientes cli)
{
        this.numero = num;
        this.saldo = sal;
        this.cliente = cli;
    }
```

Associações

- Exemplo em Conta Corrente – Criando os objetos

```
Clientes cli1 = new Clientes("Joice", "23433378921");
CCorrente cc1 = new CCorrente(1, 1000, cli1);
```



Endereço onde está armazenado o objeto que pertence a classe Cliente

O método toString do java

• O método toString() vem de java.lang.Object que é o “pai” de todos os objetos em java, e todas as classes java herdam este método e podem reimplementá-lo (polimorfismo)

• Ele é bastante útil para obter informações sobre os estados dos objetos

• Se formos imprimir diretamente um objeto o que será impresso é a referência (endereço do objeto na memória)

• Para que possamos imprimir o estado (atributos de um objeto utilizamos o método toString()

O método toString do java
Exemplo

- Se tentarmos imprimir o objeto cc1 sem usarmos o método ToString
  - System.out.println(cc1)
  - Teremos como resultado algo do tipo:
    - `contas.CCorrente@1f32e575 (endereço do objeto)`
- Ao usarmos o método toString()

```
@Override
    public String toString(){
     return ("Conta:" + this.numero + " Saldo:" + this.saldo + this.cliente);

    }

@Override
    public String toString(){
     return (" Cliente:" + this.nome);

    }
```

- System.out.println(cc1)
- Teremos como resultado:
  - `Conta:1 Saldo:1000.0 Cliente:Joice`

Atividade Prática

- Implemente a Classe Cliente em Contas, juntamente com as alterações necessárias e o método toString para imprimir as contas mais facilmente