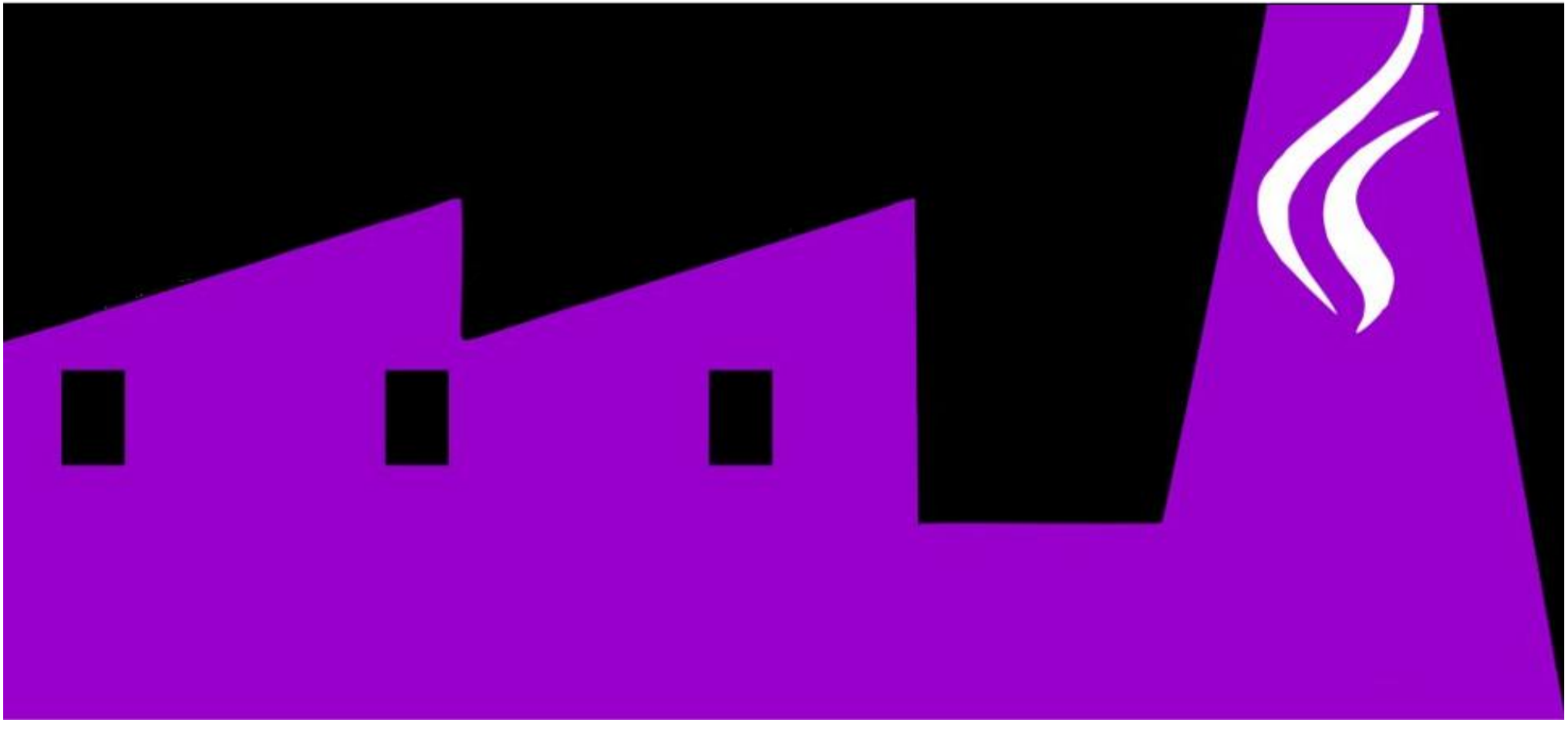
Fábrica de Software

Prof[es.] Ivan L. Süptitz e Evandro Franzen

Associação, Agregação e Composição



Objetivo da aula

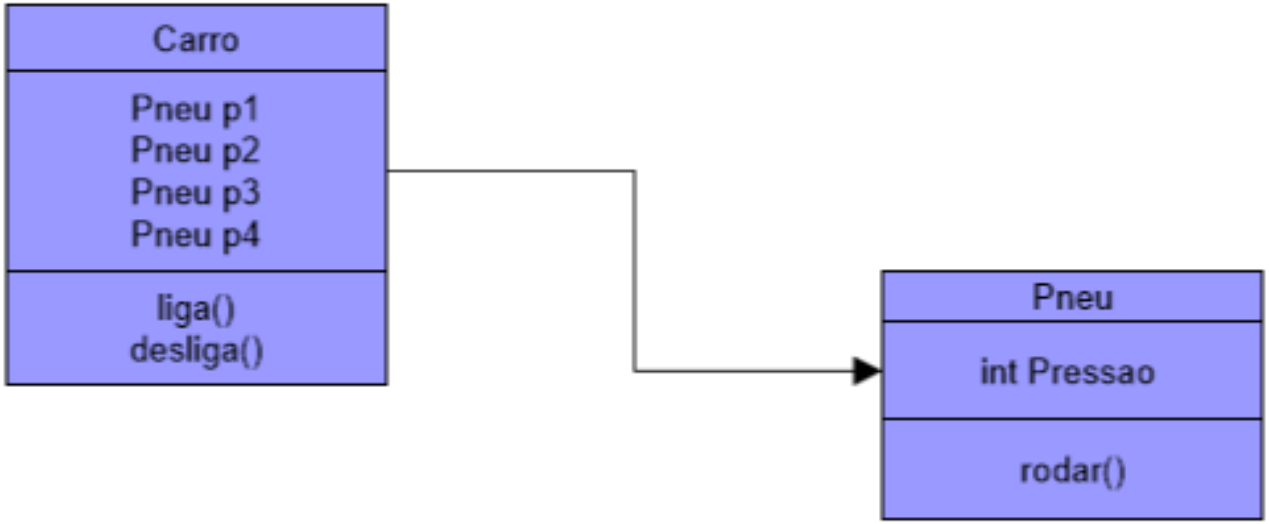
- No desenvolvimento de software, geralmente temos relacionamentos entre os vários objetos.

- Nosso objetivo hoje é identificar e distinguir os diferentes tipos de relacionamentos entre objetos em um sistema de software.

- A composição, agregação e associação são conceitos fundamentais de modelagem de objetos, e a compreensão desses conceitos é essencial para o desenvolvimento de sistemas de software robustos e flexíveis.

Associação

- Associação ocorre quando uma classe possui atributos do tipo de outra classe.



Nota : Neste caso estamos dizendo que carro possui pneu (4 pneus)

O carro continuaria sendo um carro sem os pneus? E o Pneu continuaria existindo sem Carro. SIM, então é uma associação

Associação

- A associação pode ser representada em Java da seguinte forma:

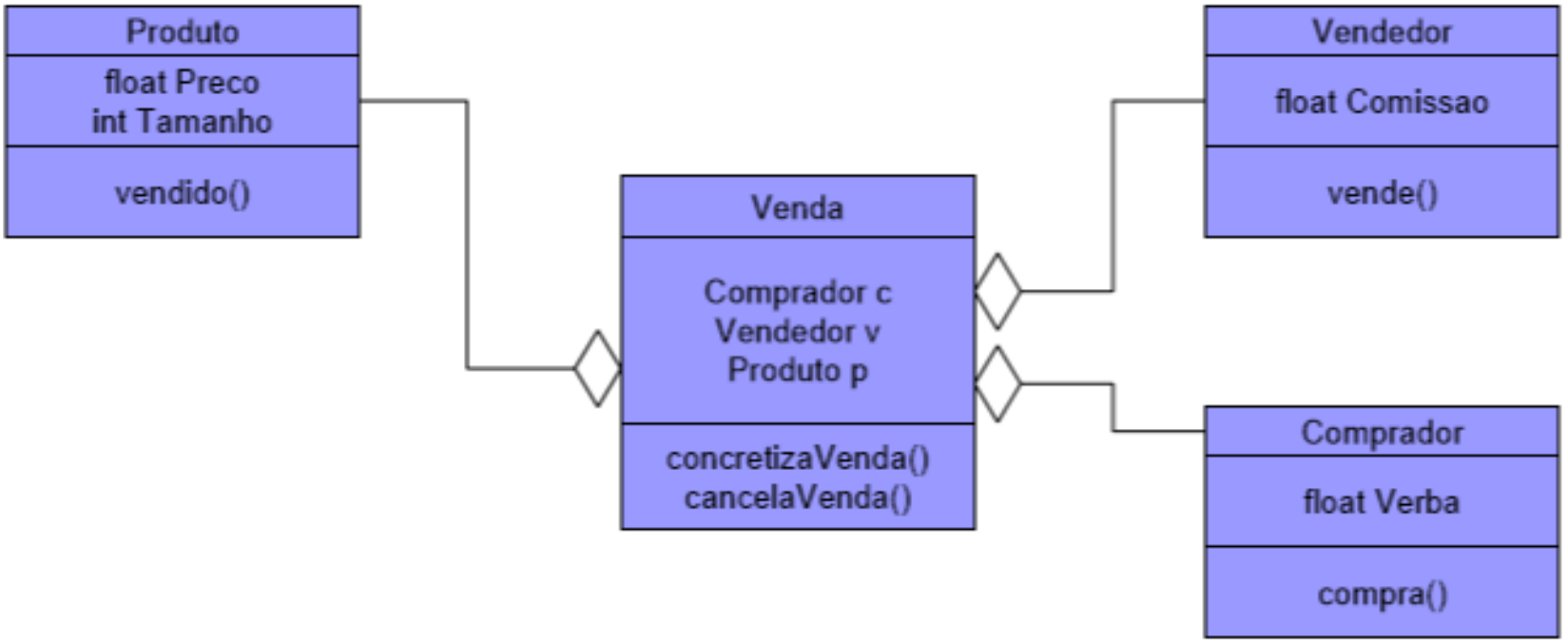
```
public class Pneu {
        int Pressao;

        void roda() {
                System.out.println("Pneu em movimento");
        }
}
```

```
public class Carro {
        Pneu p1;
        Pneu p2;
        Pneu p3;
        Pneu p4;

        void liga() {
                System.out.println("Carro ligado");
        }

        void desliga() {
                System.out.println("Carro desligado");
        }
}
```

Agregação

- Ocorre quando uma classe usa outras classes em suas operações. As classes utilizadas participam da classe principal, mas a classe principal não contém estas classes utilizadas como sendo partes suas.

Agregação



Nota : Neste caso **Venda** é o objeto definido como sendo o **todo.** E este objeto somente pode existir caso os demais objetos que o compõem também existam.

A Venda continuaria existindo sem Produto, Vendedor e Comprador?
NÃO, então é uma Agregação

Agregação

- A agregação pode ser representado da seguinte forma

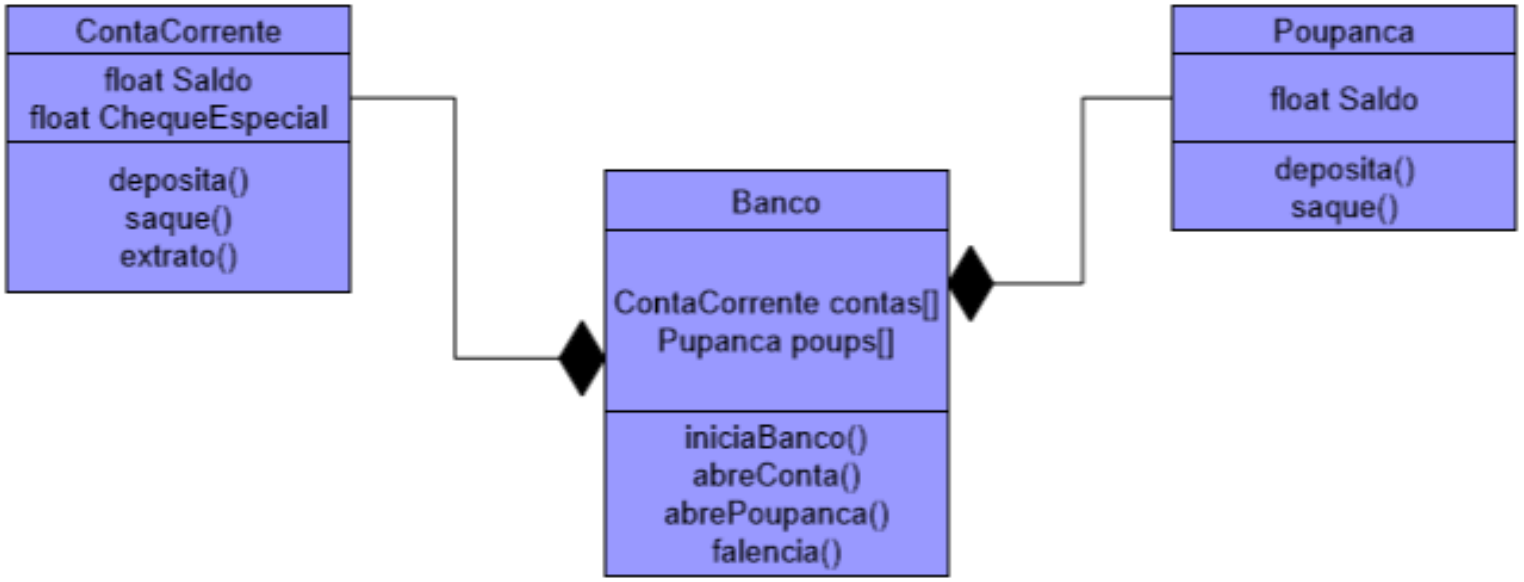
```
public class Vendedor {
        float Comissao;

        void vende() {
                System.out.println("Vendido");
        }    }

public class Comprador {
        float Verba;

        void compra() {
                System.out.println("Comprado");
        }    }

public class Produto {
        float Preco;
        int Tamanho

        void vendido() {
                System.out.println("Vendido");
        }    }
```

```
public class Venda {
        Comprador c;
        Vendedor v;
        Produto p;

        void concretizaVenda() {
                System.out.println("Venda efetuada");
                c.Verba -= p.Preco;
                v.Comissao += p.Preco * 0.1f;
                p.vendido();
        }

        void cancelaVenda() {
                System.out.println("Venda cancelada");
        }    }
```

Composição

- Semelhante a agregação, a composição também é um conjunto onde há uma classe representando o **todo** e classes satélites funcionando como **partes**.
- Sua principal diferença ocorre que quando o objeto **todo** deixar de existir os seus objetos **partes** deverão deixar de existir também.

Composição



Nota : No caso desta composição uma vez que o **Objeto** banco for destruído todas os objetos **Poupanca** e **ContaCorrente** deverão ser destruídos também.

Pode existir uma ContaCorrente sem um Banco? NÃO, então é uma Composição

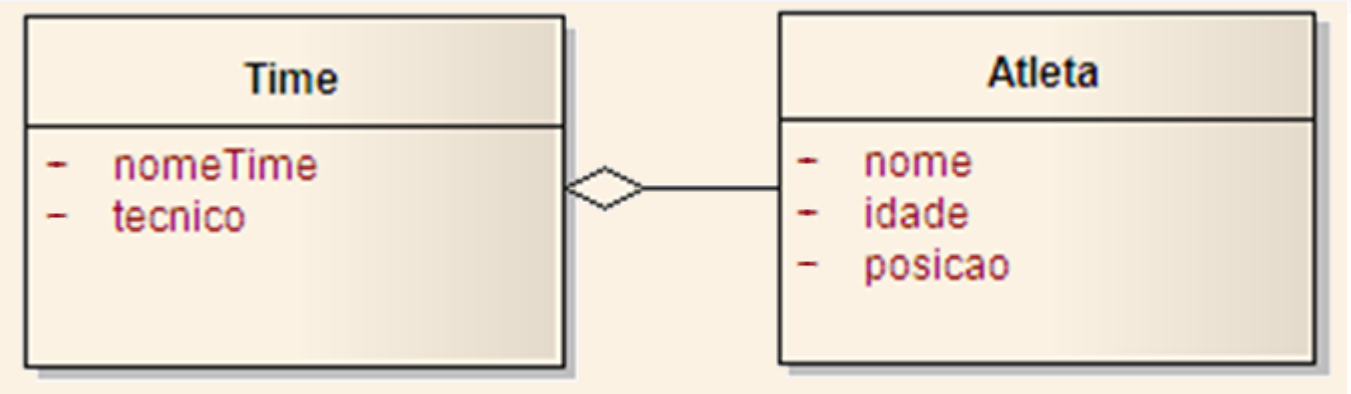
Composição

- A composição pode ser representado da seguinte forma:

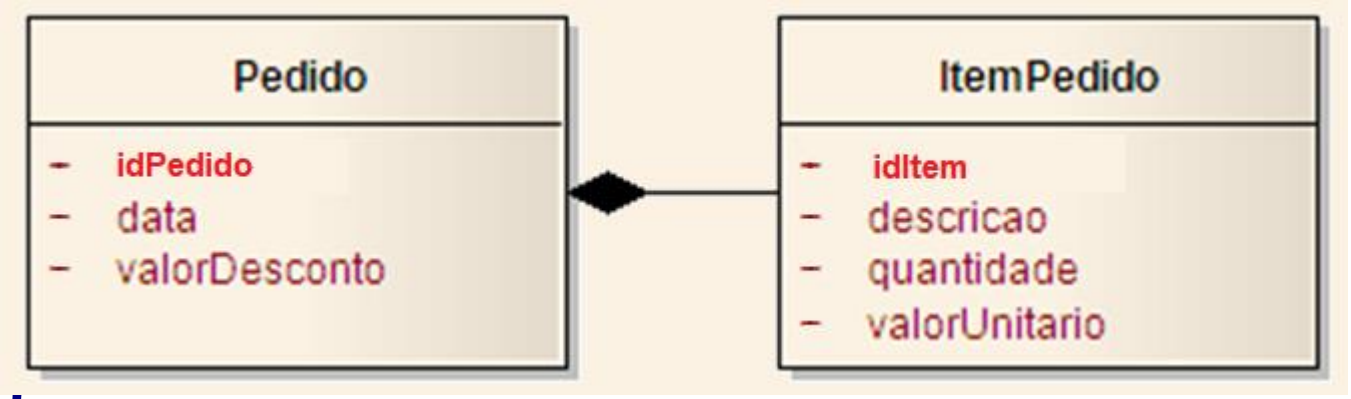
```
public class Poupanca {
        float Saldo;

        void saque() {
                Saldo -= 10.0f;
                System.out.println("Novo Saldo →" + Saldo);
        }
        void deposito() {
                Saldo += 10.0f;
                System.out.println("Novo Saldo →" + Saldo);
        }    }
public class ContaCorrente {
        float Saldo;

        void saque() {
                Saldo -= 100.0f;
                System.out.println("Novo Saldo →" + Saldo);
        }
        void saque() {
                Saldo -= 100.0f;
                System.out.println("Novo Saldo →" + Saldo);
        }    }
```

```
public class Banco {
        Poupanca[] pops;
        ContaCorrente[] cc;
        int numConta, numPoupanca;
        void iniciaBanco() {
                pops = new Poupanca[100];
                cc = new ContaCorrente[100];
                numConta = 1;
                numPoupanca = 1;
        }
        void abreConta() {
                cc[ numConta ] = new ContaCorrente();
                numConta++;
        }
        void abrePoupanca() {
                pops[ numConta ] = new Poupanca();
                numPoupanca++;
        }
        void falencia() {
                for (int i = 0; i< 100; i++) {
                   pops[ i ] = null;
                   cc[ i ] = null;
                }
        }    }
```

Agregação



Composição



Perguntas

Atletas existem sem time?

Item de pedido existem sem um pedido?

Exercício 9.1

Criar as estruturas de Agregação e Composição da página anterior e um programa para testar cada uma dos diagramas.

No programa Time, adicionar um método que calcula e retorna a idade média dos atletas.

No programa do Pedido, adicionar 2 métodos:

- adicionarItem(descricao, qtde, valor): deve inserir um novo produto na lista de itens;
- totalizarPedido(): deve percorrer a lista de itens, obter o valor total e diminuir o desconto.

Exercício 9.2

Objetivo: utilizar agregação de objetos

Enunciado: um computador possui uma memória, um processador, um teclado e um monitor. A memória tem uma capacidade limitada, podendo ser de vários tamanhos múltiplos de 2 a partir de 1024Kb. Um processador tem um clock, um teclado é de um tipo (pt, en, ...) e um monitor tem um tamanho. Faça um programa que instancie e mostre os detalhes de um computador.

Continuação do Exercício 9.2

Instancie dez computadores com características aleatórias diferentes. Coloque-os em um vetor de computadores. Crie um conceito de *Loja* para salvar o vetor dos computadores. Faça um método na loja que imprime todas as características de todos os computadores.

Uso de random (números aleatórios):

```
import java.util.Random;
Random rand = new Random();
int r = rand.nextInt(1000);//aleatório de valor
  máximo = mil;
```

Continuação do Exercício 9.2

É possível usar a assinatura completa

Vantagem: não precisa do import

Uso de random (números aleatórios):

```
import java.util.Random;

java.util.Random rand =
                    new java.util.Random();
int r = rand.nextInt(1000);
```