# REVISÃO ORIENTAÇÃO A OBJETO

Drª. Alana Morais

alanamm.prof@gmail.com

# IMPORTANTE!

A orientação a objetos, por si só, não garante sistemas reusáveis e extensíveis

# CONCEITOS DE OO

Classe

Objeto

Método

Instância

Construtor

# Qual a diferença entre Declarar e Instanciar?

# CLASSE

Composta por:

- Atributos
- Construtor
- Métodos

```
public class [nome da Classe]{
        [atributos]
        [métodos (um deles é o construtor)]
    }
```

# O QUE ESTÁ ERRADO NESTA CLASSE?

```
public class Exemplo{
	private String nome;
	public Exemplo(String n){
		this.nome = n;
	}
	public void getNome(){
		return this.nome;
	}
	public String setNome(String n){
		this.nome = n;
	}
}
```

# MODIFICADORES

Modificadores de Acesso

- private
- public
- protected

São aplicados:

- Métodos
- Atributos
- Classe

# MODIFICADORES

PRIVATE

- Quando aplicado a um **atributo** ou a um **método** indica que os mesmos só podem ser acessados de dentro da classe que os criou.

- Como acessar atributos e métodos privativos em uma classe?

# MODIFICADORES

PROTECTED

- Indica que o método, classe ou a variável, assim declarada, só pode ser acessada dentro do pacote.

PUBLIC

- A classe, método ou variável pode ser acessada em qualquer lugar e a qualquer momento da execução do programa.

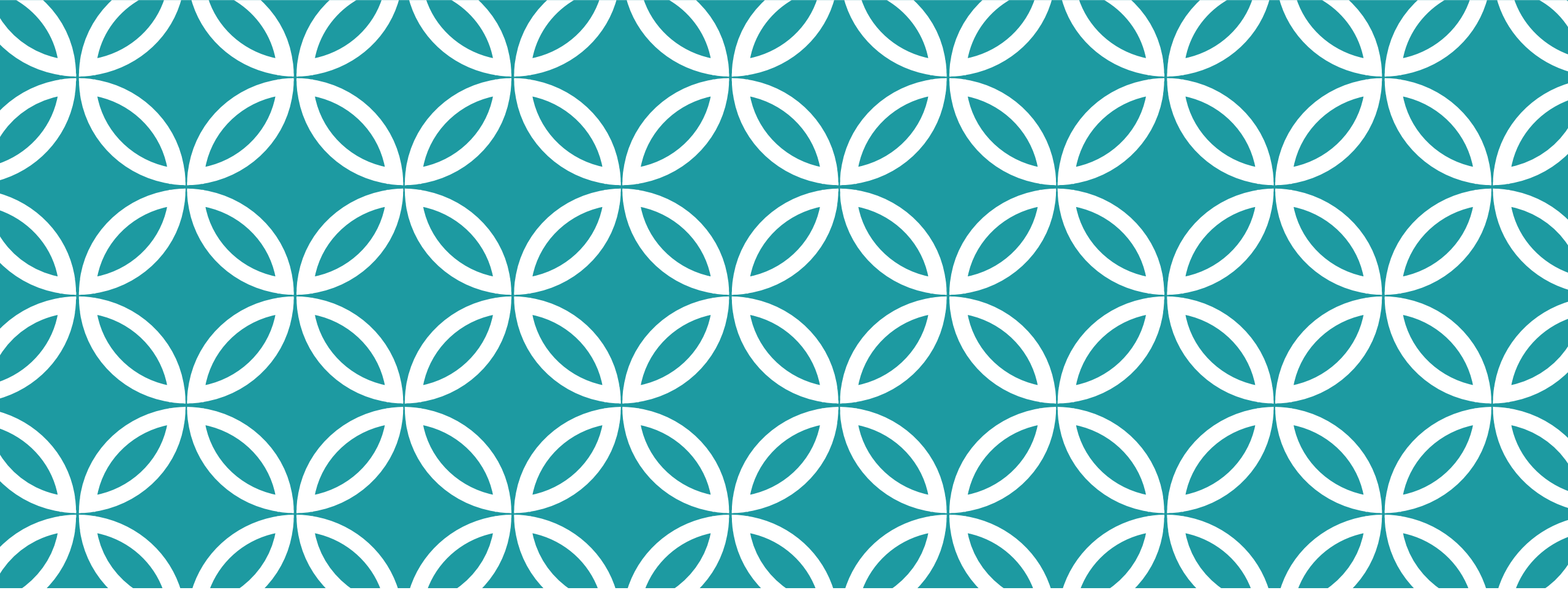
# CICLO DE VIDA DE UM OBJETO

**Instanciação**: o objeto é criado na memória e passa a ser referenciado por uma variável de referência;

**Uso**: o objeto recebe mensagens de outros objetos e, com isso, executa parte da funcionalidade do sistema;

**Destruição**: quando o objeto não é mais referenciado (inacessível) ele torna-se elegível para a coleta de lixo.

**Coletor de Lixo (Em JAVA)**: Limpa a memória ocupada pelos objetos inacessíveis (quando há falta de memória).

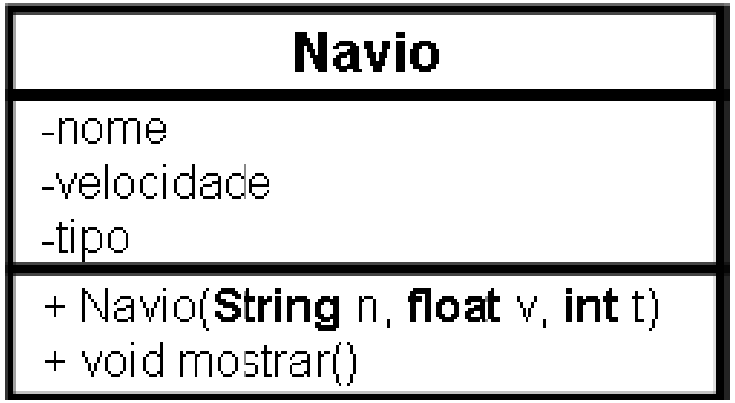


# VAMOS EXERCITAR!! ^^'

# EXERCÍCIO

Crie uma classe Navio cujos campos e métodos estão resumidos na Figura 1. As características de cada campo e cada método são especificadas na Tabelas 1.1.

**Figura 1**




| Método | Descrição |
|---|---|
| Navio | Construtor com 3 parâmetros cujos valores foram fornecidos pelo usuário através de um comando de leitura via teclado. |
| mostrar | Método que mostra os valores contidos em um objeto do tipo **Navio**. |

**Tabela 1.1**

Além disto, implemente um programa que ilustre o funcionamento dos métodos da classe **Navio** (classe teste).

# EXERCÍCIO 2

Criar uma classe **Carro** cujos campos e métodos estão resumidos na Figura 2. As características de cada campo e cada método são especificadas na Tabelas 2.1

**Figura 2**

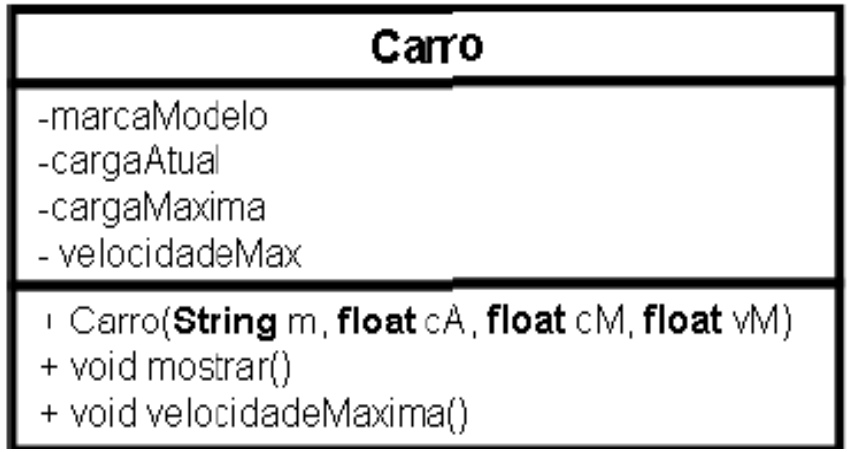
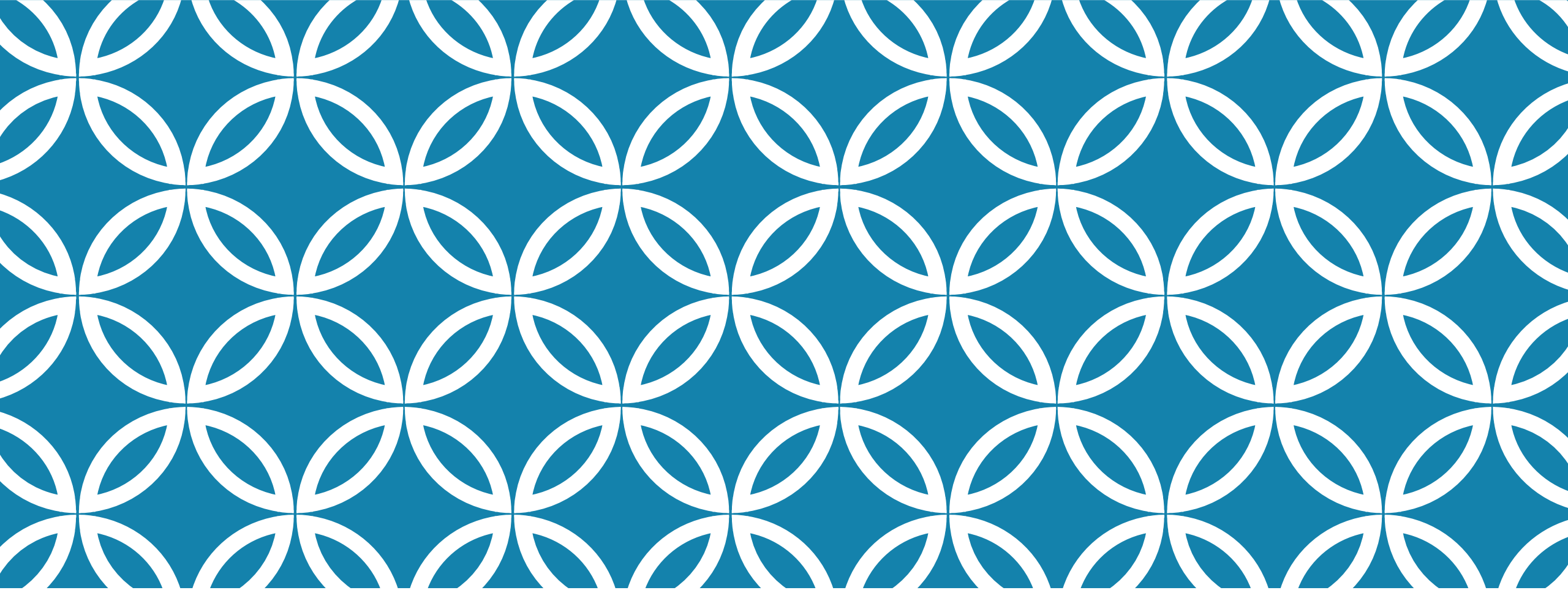



**Tabela 2.1**

| Método | Descrição |
|---|---|
| Carro | Construtor com 3 parâmetros cujos valores foram fornecidos pelo usuário através de um comando de leitura via teclado. |
| mostrar | Método que mostra os valores contidos em um objeto do tipo Carro. |
| velocidadeMaxima | Método que fornece a velocidade máxima para uma determinada marca e modelo de Carro de acordo com a seguinte fórmula: (cargaAtual/cargaMaxima)*velocidadeMax |

Construir um programa que ilustre o funcionamento dos métodos da classe **Carro**, de acordo com as descrições.



# HERANÇA X COMPOSIÇÃO

# HERANÇA X COMPOSIÇÃO

Composição e herança são dois mecanismos para **reutilizar funcionalidade**

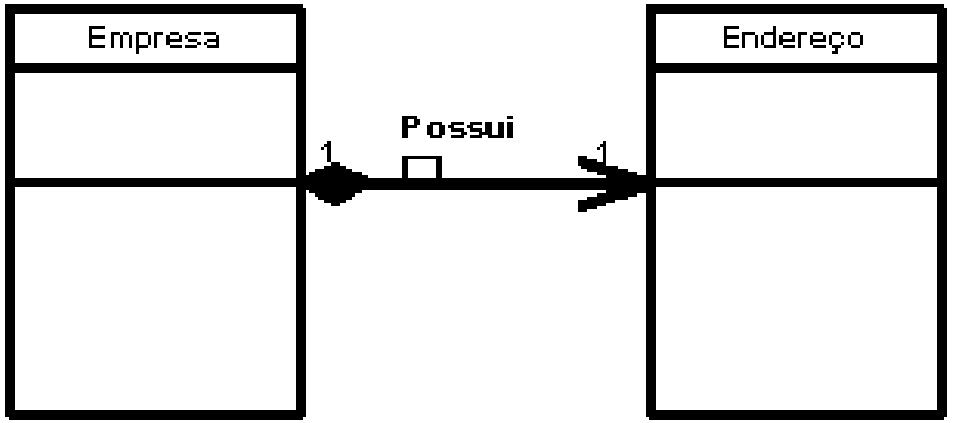
A composição estende uma classe pela delegação de trabalho para outro objeto enquanto a herança estende atributos e métodos de uma classe

Hoje, considera-se que a composição é muito superior à herança na maioria dos casos

Quer dizer que iremos baixar um pouco a bola da herança??

Calma! A herança continuará sendo importante, mas aprenderemos que para contextos mais específicos.

# COMPOSIÇÃO




Use composição para estender as responsabilidades pela delegação de trabalho a outros objetos

Um exemplo no domínio de endereços

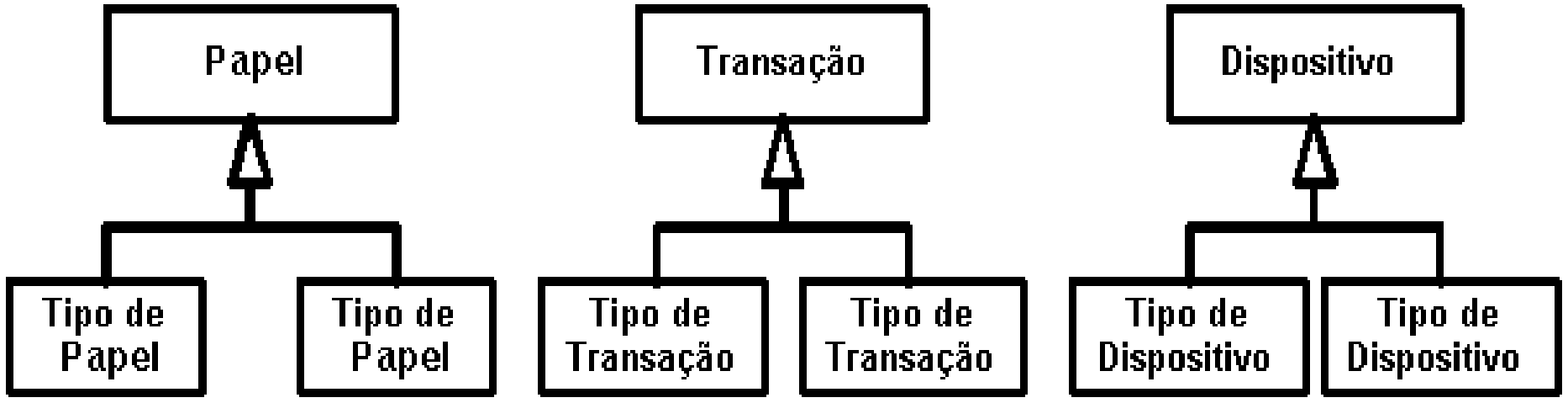
- Uma empresa tem um endereço (digamos só um)
- Uma empresa "tem" um endereço
- Podemos deixar o objeto empresa responsável pelo objeto endereço e temos agregação composta (composição)

# HERANÇA

Atributos, conexões a objetos e métodos comuns vão na superclasse (classe de generalização)

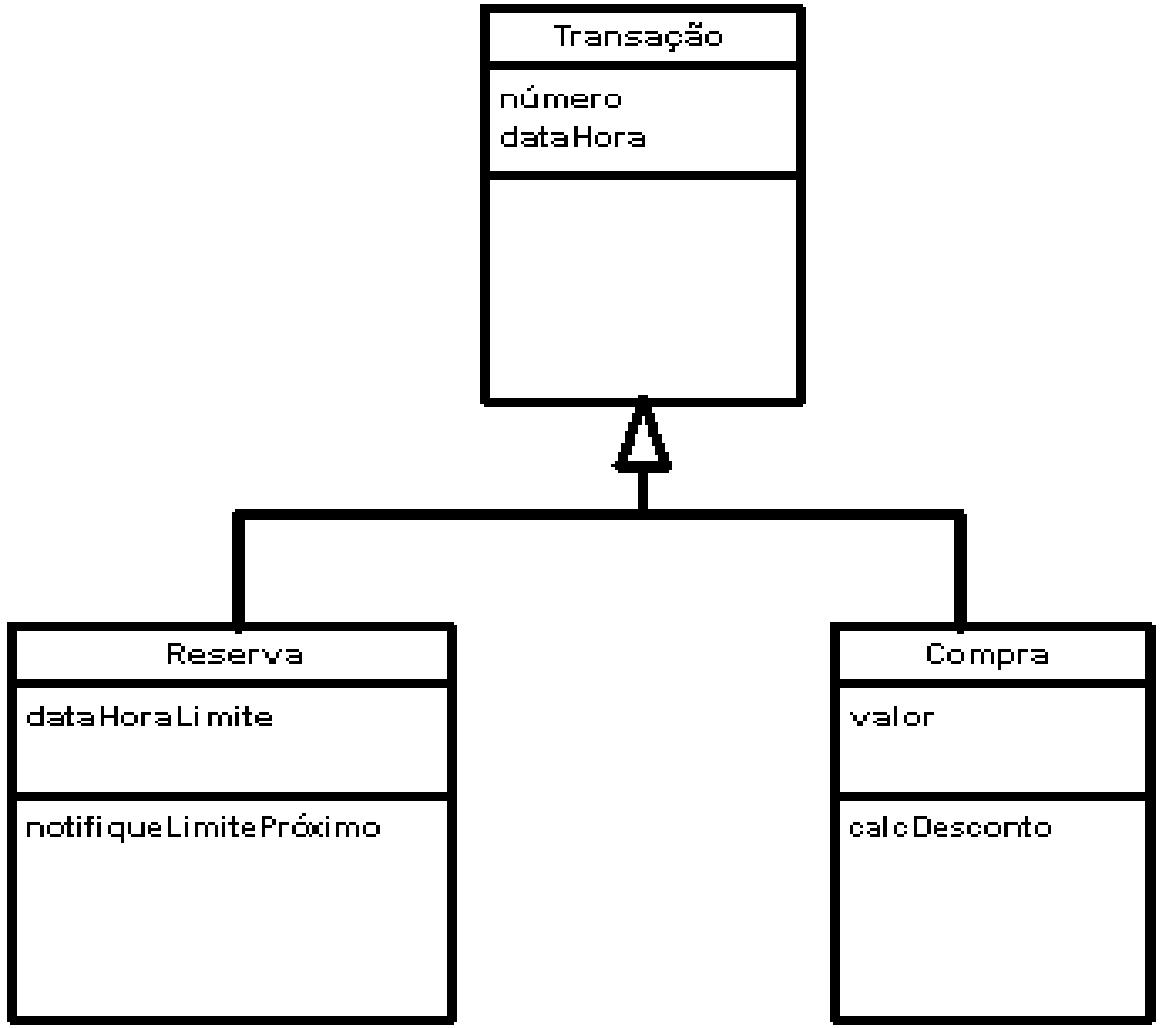
Adicionamos mais características e ações nas subclasses (classes de especialização)

Três situações comuns para a herança:



# HERANÇA

Exemplo no domínio de reserva e compra de passagens de avião



# BENEFÍCIOS DA HERANÇA

Captura o que é comum e o isola daquilo que é diferente

A herança é vista diretamente no código

# PROBLEMAS DA HERANÇA

O encapsulamento entre classes e subclasses é fraco (o acoplamento é forte)

Mudar uma superclasse pode afetar todas as subclasses

Isso viola um dos princípios básicos de projeto OO (manter fraco acoplamento)

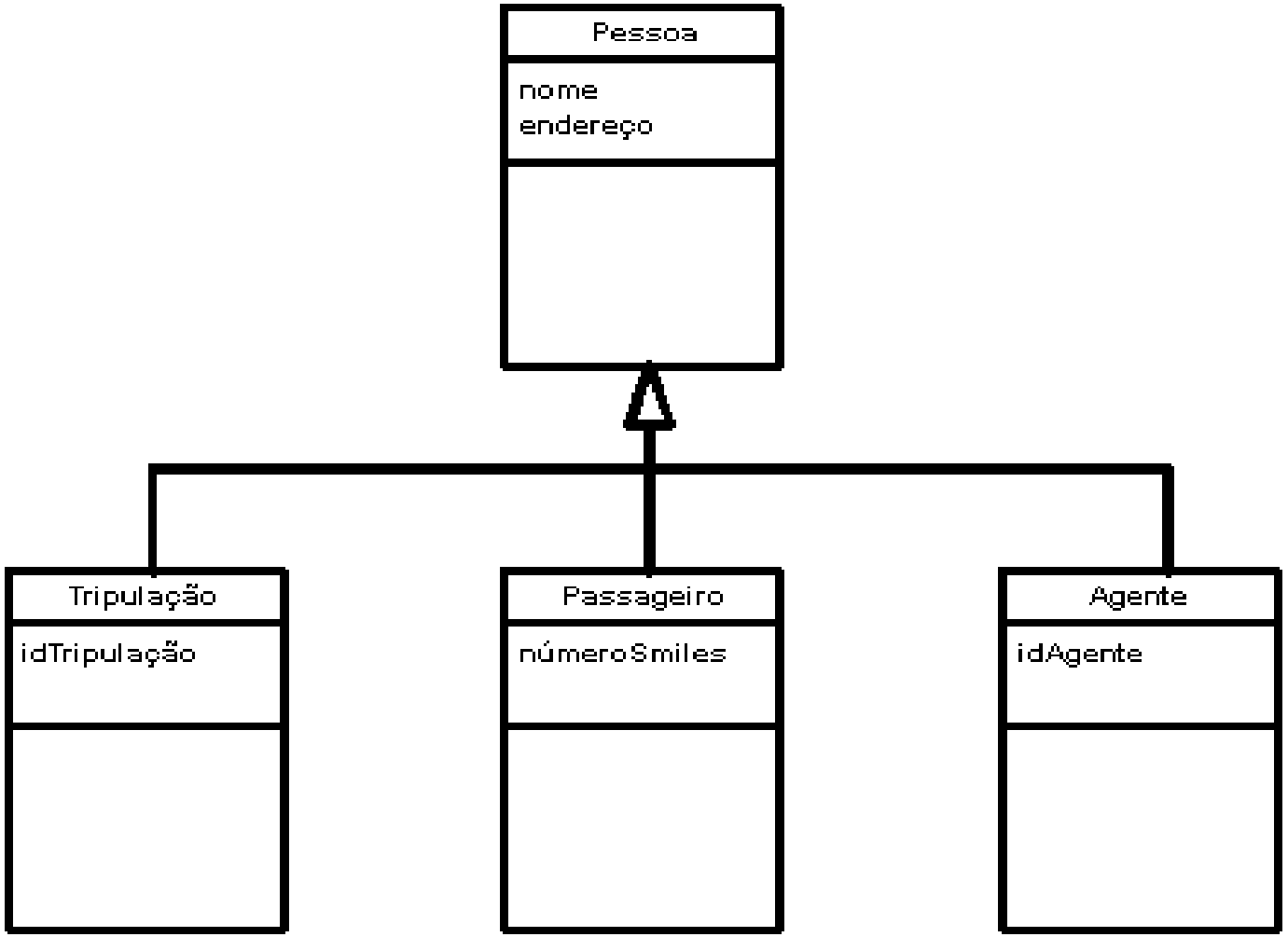
Com herança, a estrutura está parafusada no código e não pode sofrer alterações facilmente em tempo de execução

A herança é um relacionamento estático que não muda com tempo

# PROBLEMAS DA HERANÇA

Às vezes um objeto precisa ser de uma classe diferente em momentos diferentes

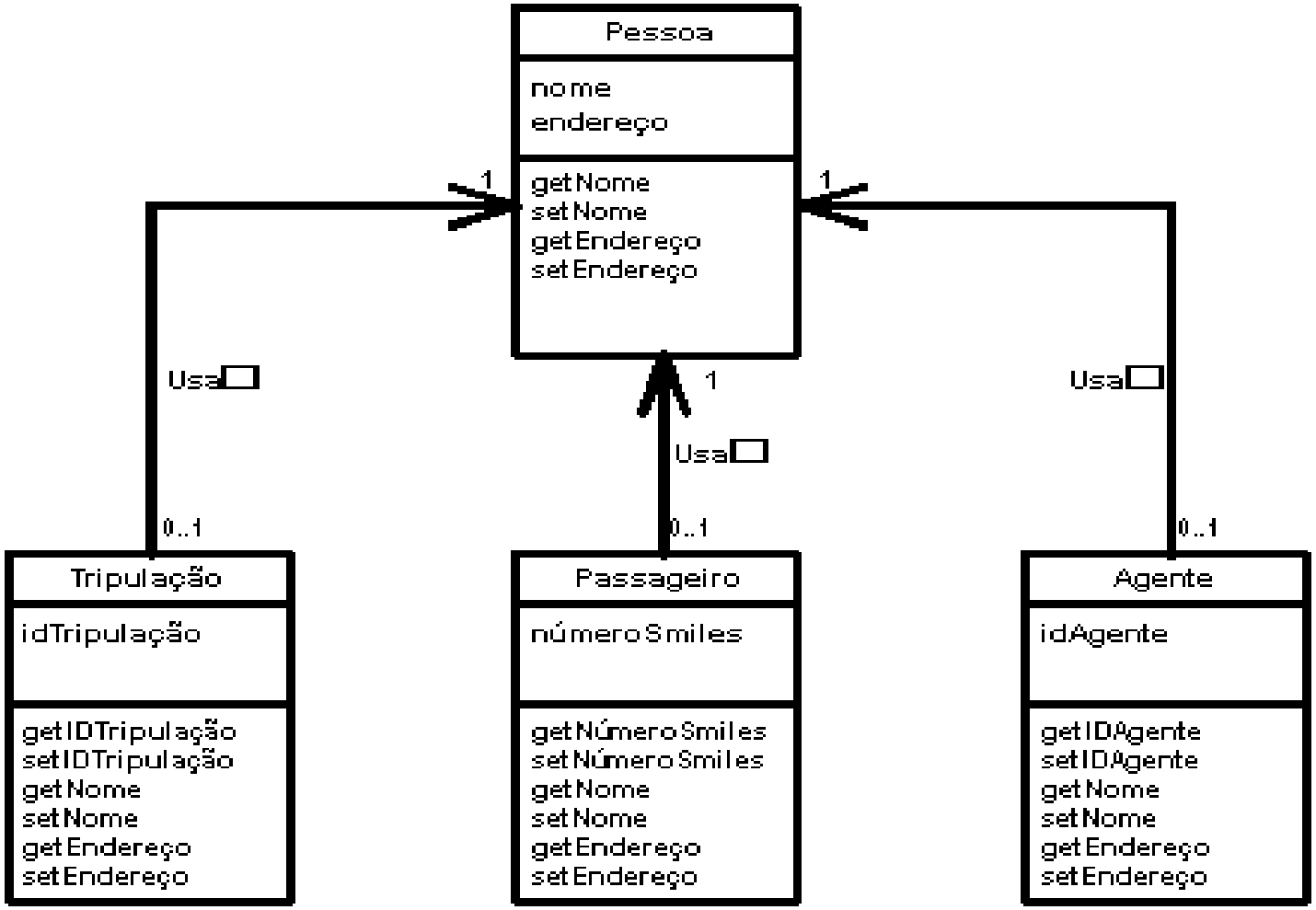
# CENÁRIO: PESSOAS ENVOLVIDAS NA AVIAÇÃO




Problema: uma pessoa pode mudar de papel ao assumir combinações de papéis

Fazer papéis múltiplos requer 7 combinações (subclasses)

# SOLUCIONANDO O PROBLEMA COM COMPOSIÇÃO: UMA PESSOA E VÁRIOS PAPEIS POSSÍVEIS

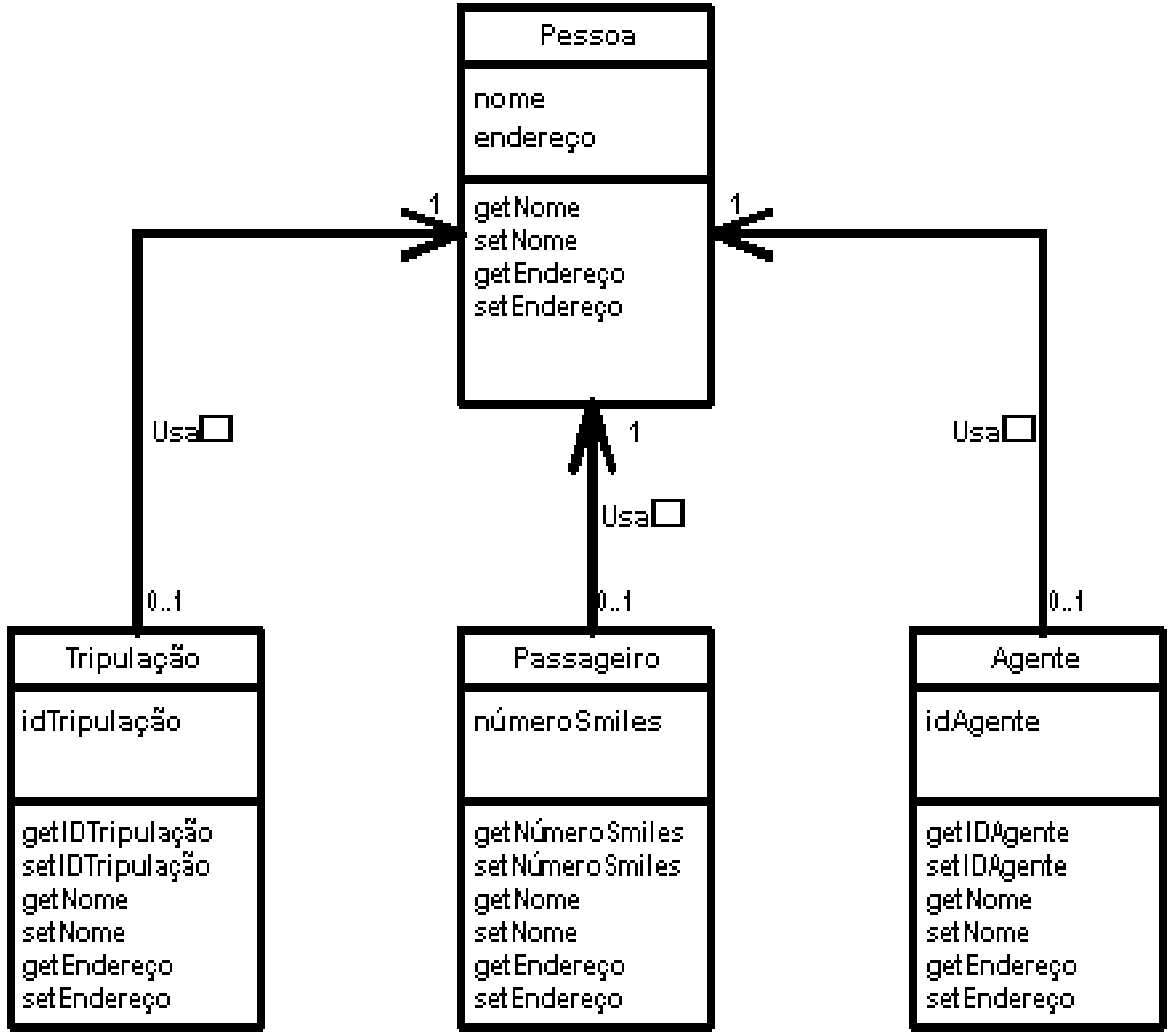



Estamos estendendo a funcionalidade de Pessoa de várias formas, mas sem usar herança

Observe que também podemos inverter a composição (uma pessoa tem um ou mais papeis)

- Pense na implicação para a interface de "pessoa"

# SOLUCIONANDO O PROBLEMA COM COMPOSIÇÃO: UMA PESSOA E VÁRIOS PAPEIS POSSÍVEIS




Estamos usando delegação: dois objetos estão envolvidos em atender um pedido (digamos setNome)

O objeto tripulação (digamos) delega setNome para o objeto pessoa que ele tem por composição

- Técnica também chamada de *forwarding*
- É semelhante a uma subclasse delegar uma operação para a superclasse (herdando a operação)
  - Delegação sempre pode ser usada para substituir a herança
- Se usássemos herança, o objeto tripulação poderia referenciar a pessoa com *this*
- Com o uso de delegação, tripulação pode passar this para Pessoa e o objeto Pessoa pode referenciar o objeto original se quiser
- Em vez de tripulação *ser* uma pessoa, ele *tem* uma pessoa
- A grande vantagem da delegação é que o comportamento pode ser escolhido em tempo de execução e vez de estar amarrado em tempo de compilação
- A grande desvantagem é que um software muito dinâmico e parametrizado é mais difícil de entender do que software mais estático

# O RESULTADO DE USAR COMPOSIÇÃO

Em vez de codificar um comportamento estaticamente, definimos pequenos comportamentos padrão e usamos composição para definir comportamentos mais complexos

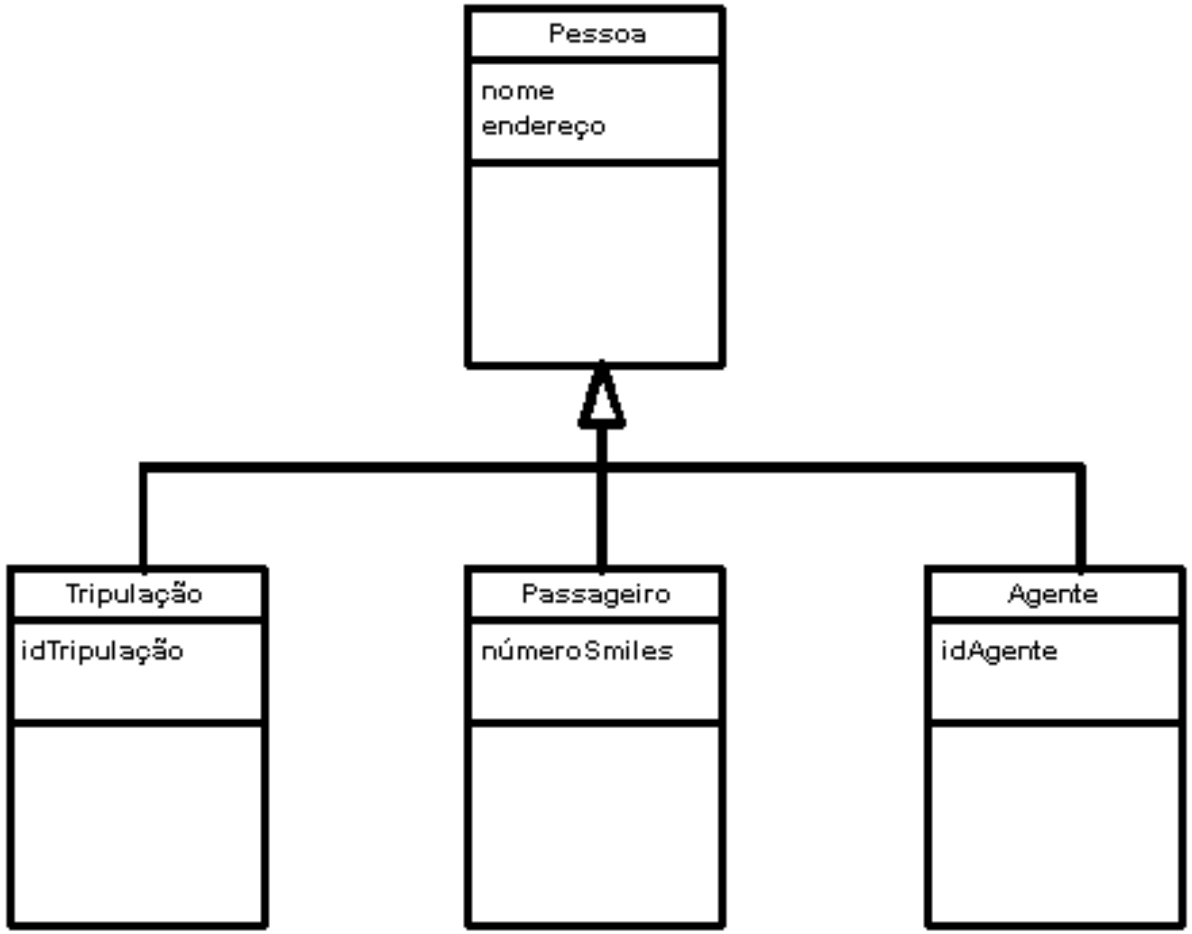
De forma geral, a composição é melhor do que herança normalmente, pois:

- Permite mudar a associação entre classes em tempo de execução;
- Permite que um objeto assuma mais de um comportamento (ex. papel);
- Herança acopla as classes demais e engessa o programa

# 5 REGRAS PARA O USO DE HERANÇA

- O objeto "é um tipo especial de" e não "um papel assumido por"
- O objeto nunca tem que mudar para outra classe
- A subclasse estende a superclasse mas não faz override ou anulação de variáveis e/ou métodos
- Não é uma subclasse de uma classe "utilitária"
- Para classes do domínio do problema, a subclasse expressa tipos especiais de papéis, transações ou dispositivos

# VAMOS APLICAR AS REGRAS NO CASO ANTERIOR




**Regra 1 (tipo especial):** não passa. Um Passageiro não é um tipo especial de Pessoa: é um papel assumido por uma Pessoa

**Regra 2 (mutação): nã**o passa. Um Agente pode se transformar em Passageiro com tempo

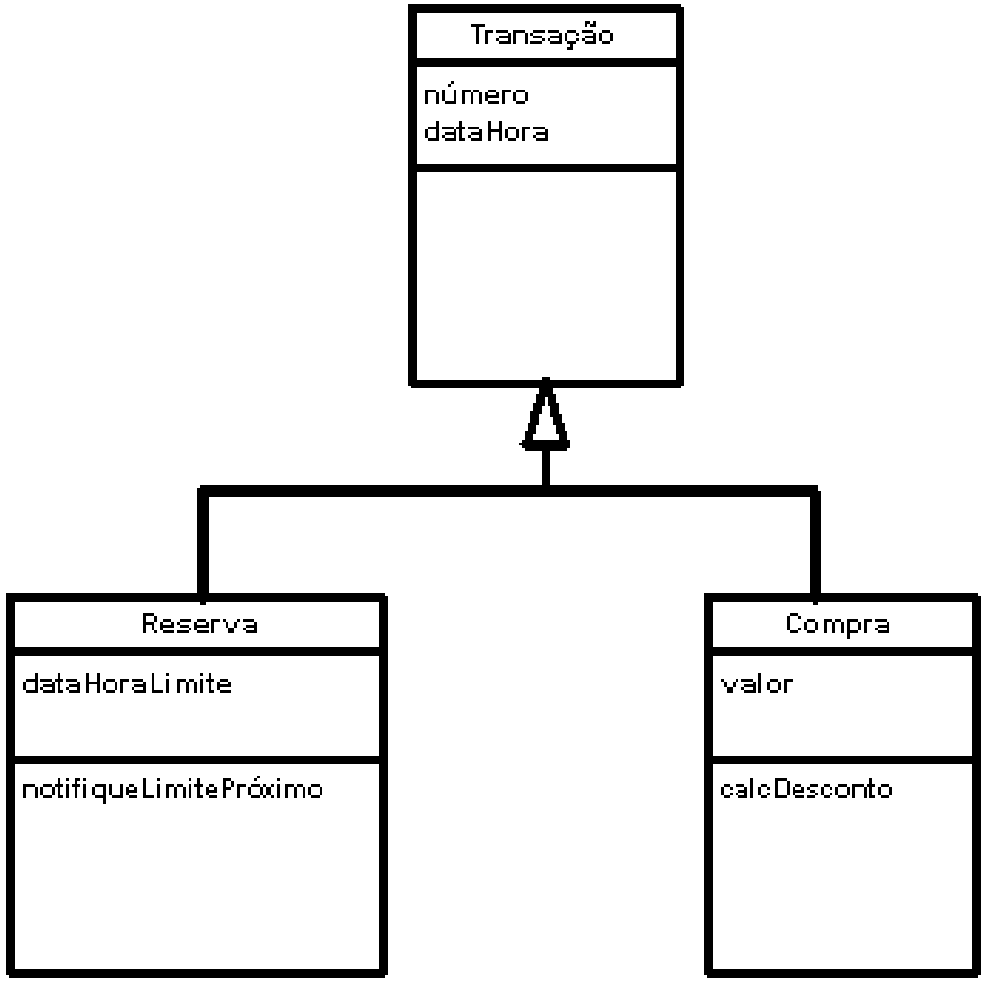
**Regra 3 (só estende):** ok.

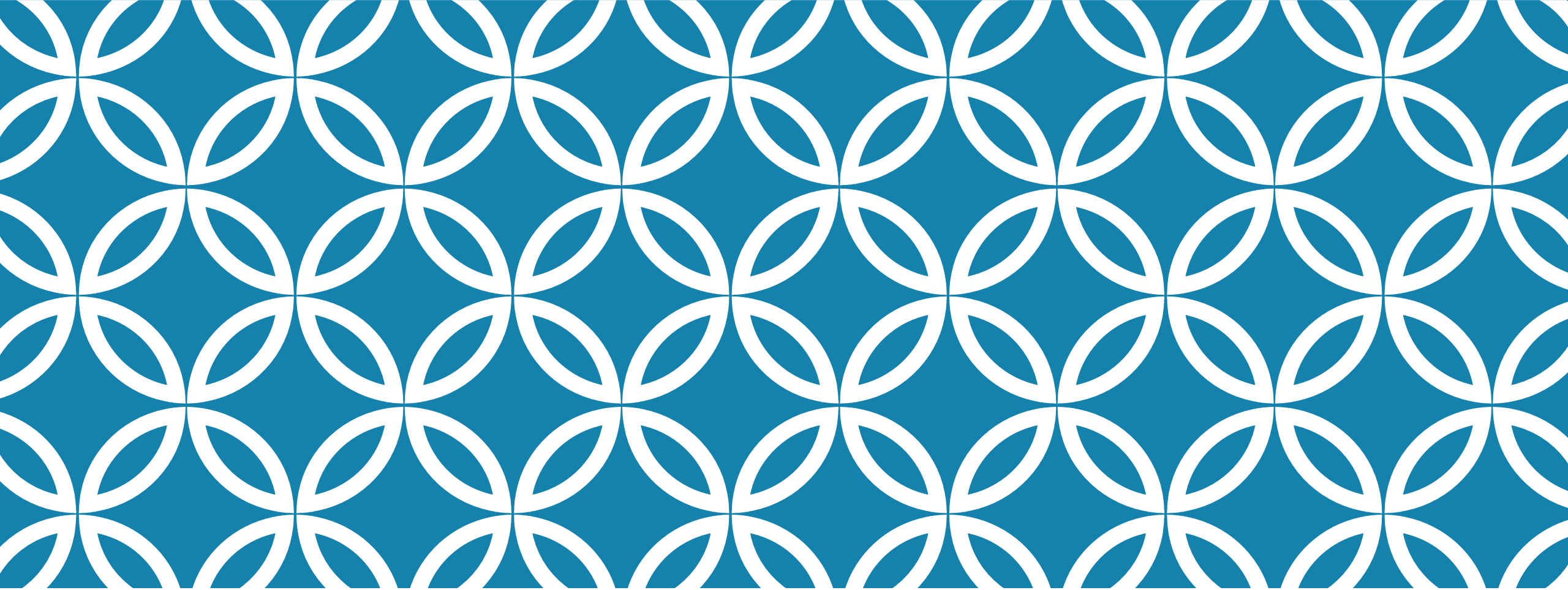
**Regra 4:** ok.

**Regra 5:** não passa. Passageiro está sendo modelado como tipo especial de Pessoa e não como tipo especial de papel

# SUA VEZ PENSE
# A HERANÇA É IDEAL NESTE CASO?

Reserva e Compra podem herdar de Transação?





# OBRIGADA