

# Diagrama de Classe UML usado na documentação de Projetos Orientado a Objetos

Prof. Dirson Santos de Campos  
dirson\_campos@ufg.br

**INF**

INSTITUTO DE  
INFORMÁTICA

27/11/2023



# Tópicos

1. O que é a Linguagem UML?
2. Diagrama de Classe na UML
  - Exemplos
3. Diagrama de Classe na notação da ferramenta UMLet
  - Instalação no IDE Eclipse
  - Instalação no Visual Studio Code
  - Exemplos



# Introdução a UML

# Página oficial da Linguagem UML



<https://www.uml.org/>

# Especificação da última versão da Linguagem UML

<https://www.omg.org/spec/UML>

INF

INSTITUTO DE  
INFORMÁTICA



Standards  
Development  
Organization

ABOUT US ▾

GROUPS ▾

CERTIFICATIONS ▾

RESOURCES ▾

SPECIFICATIONS ▾

## ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1

2.5.1 • UML

UML®

Unified Modeling Language

A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.

Title: Unified Modeling Language

Acronym: UML®

Version: 2.5.1

Document Status: formal ⓘ

Publication Date: dezembro 2017

Categories:

Modeling

Software Engineering

Platform

IPR Mode ⓘ

RF-Limited ⓘ

# O que é a Linguagem UML?

- A UML, que significa **Unified Modeling Language** (Linguagem de Modelagem Unificada), é uma linguagem de modelagem visual usada na engenharia de software para representar sistemas de software de forma clara e precisa.
- Ela fornece um conjunto de notações gráficas e semântica para descrever diferentes aspectos de um sistema, como sua estrutura, comportamento, interações e arquitetura.
- Desenvolvida inicialmente pela Rational Software Corporation (agora parteda IBM) na década de 1990, a UML tornou-se um padrão da indústria amplamente adotado.
- Última versão 2.5.1 de dezembro de 2017

# Importância do padrão UML?

- A importância da UML na engenharia de software pode ser destacada da seguinte forma:
  - Comunicação
  - Documentação
  - Projeto e análise
  - Documentação
  - Padronização
  - Ferramentas de Software
  - Melhorias no Processo de Desenvolvimento

## Importância do padrão UML?

- Em resumo, a UML desempenha um papel crucial na engenharia de software, ajudando as equipes a entender, projetar, comunicar e documentar sistemas de software de maneira eficaz.
- Ela promove a padronização, a clareza e a colaboração, tornando o desenvolvimento de software mais eficiente e menos propenso a erros.



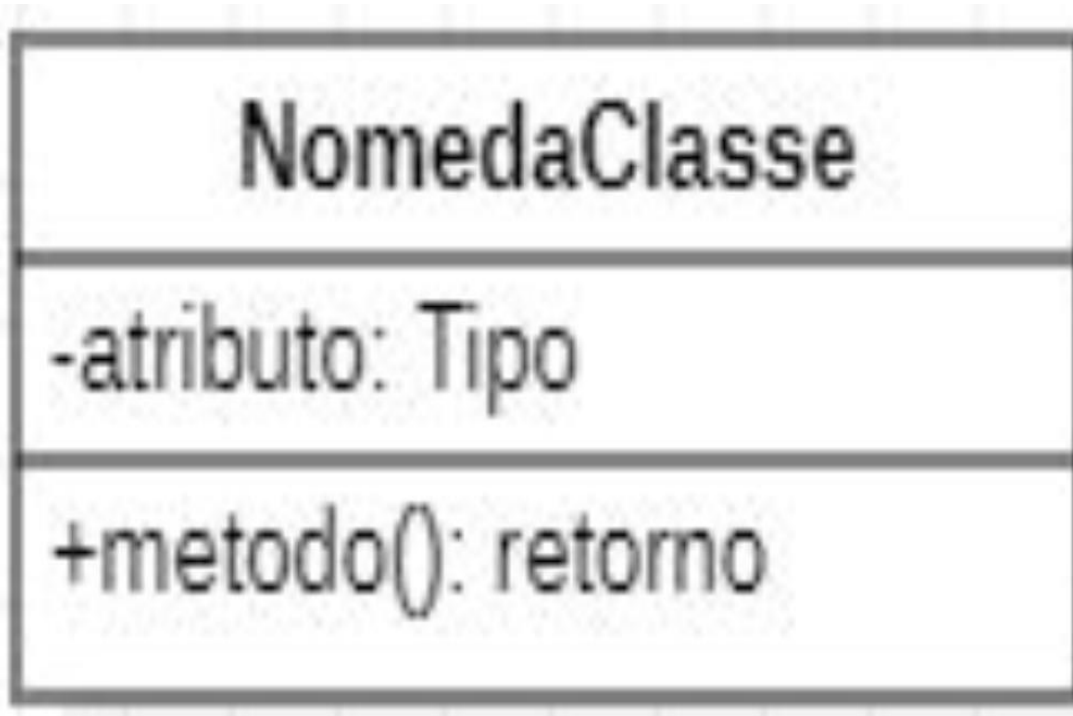


# Diagrama de Classe UML

## Diagrama de Classe na UML

- Os diagramas de classes são usados para modelar a estrutura de um sistema orientado a objetos contendo classes, atributos e métodos.

# Representação de um Diagrama de Classes



# Modificadores de acesso dos atributos e métodos

- (Sinal de adição): **O sinal de adição (+) indica que o atributo/método é público.** Isso significa que o atributo pode ser acessado e modificado por qualquer classe que tenha uma instância desse objeto.
- (Sinal de subtração): **O sinal de subtração (-) indica que o atributo/método é privado.** Isso significa que o atributo só pode ser acessado e modificado pela própria classe à qual ele pertence.

# Modificadores de acesso dos atributos e métodos

- (Sinal de cerquilha): **O sinal de cerquilha (#) indica que o atributo/método é protegido (protected).** Isso significa que o atributo pode ser acessado pela classe à qual ele pertence e também por subclasses dessa classe.

# Modificadores de acesso dos atributos e métodos

- (Sinal de til) : **O sinal de til (~) indica que o atributo/método é default (padrão).** Isso significa que a classe e/ou seus membros são acessíveis somente por classes do mesmo pacote.

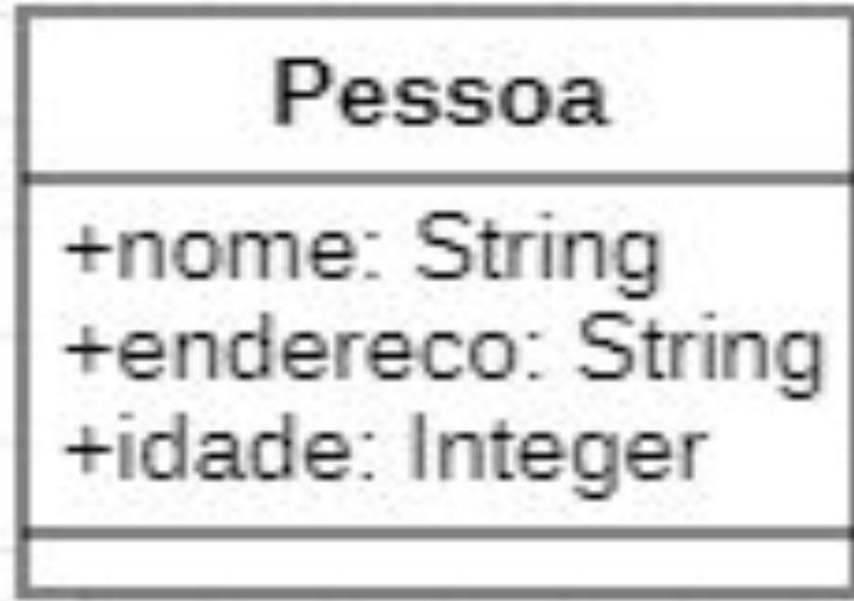
**Observação:** Não há palavra-chave para acesso padrão. Simplesmente omite o modificador de acesso.

# Modificadores de Acesso em Java

Recordando, um modificador de acesso determina como será a visibilidade de uma classe, atributo ou método a partir de outras classes, métodos ou pacotes.

	<b>private</b>	<b>default</b>	<b>protected</b>	<b>public</b>
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasses)	não	não	sim	sim
pacotes diferentes (sem subclasses)	não	não	não	sim

## Representação de um Diagrama de Classes (atributos)



A classe Pessoa contém 3 atributos públicos e sem métodos



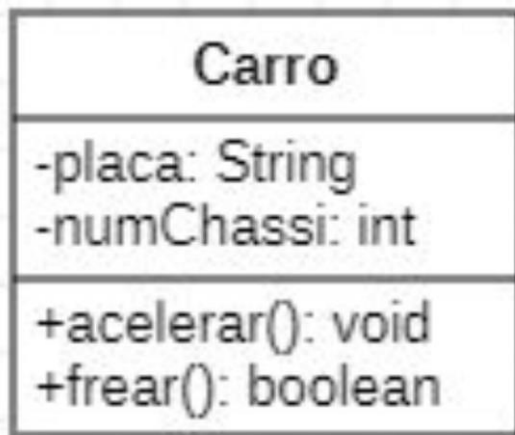
## • Métodos com retorno

- Métodos com retorno são na verdade funções dentro de uma classe que executam uma determinada operação e retornam um valor de um determinado tipo de dado.

## • Método sem retorno

- Métodos sem retorno são na verdade funções dentro de uma classe que executam uma determinada operação e não retornam um valor de um determinado tipo de dado. Chamados de void.
- A palavra void significa vazio em inglês, então no Diagrama de Classe pode não se escrever void e deixar simplesmente vazio.

## Representação de um diagrama de classes (atributos e métodos)



A classe Carro contém 2 atributos privados e com dois métodos públicos.

## Relacionamento entre classes: associação

- Um Diagrama de Classes com relacionamento de associação é uma representação visual de como as classes em um sistema orientado a objetos estão relacionadas entre si.
- Ele descreve como objetos de diferentes classes interagem e se comunicam por meio dessas associações.

## Relacionamento entre classes: associação

- As associações são usadas para modelar a estrutura estática de um sistema, mostrando como as classes estão conectadas umas às outras.
  - As classes são representadas como retângulos.
  - Cada retângulo representa uma classe e contém o nome da classe no topo do retângulo.

## Relacionamento entre classes: associação

- As associações são linhas sólidas ou setas que conectam as classes no diagrama. Elas representam os relacionamentos entre as classes.
- Uma associação indica que objetos de uma classe estão de alguma forma ligados a objetos de outra classe. As associações podem ser unidirecionais ou bidirecionais, dependendo da natureza do relacionamento (navegabilidade).

## Relacionamento entre classes: associação

- As associações contêm multiplicidades que é um número colocado perto de uma extremidade da linha de associação para indicar quantos objetos de uma classe estão associados a quantos objetos da outra classe.
- As associações podem ser enumeradas:

**0..1**

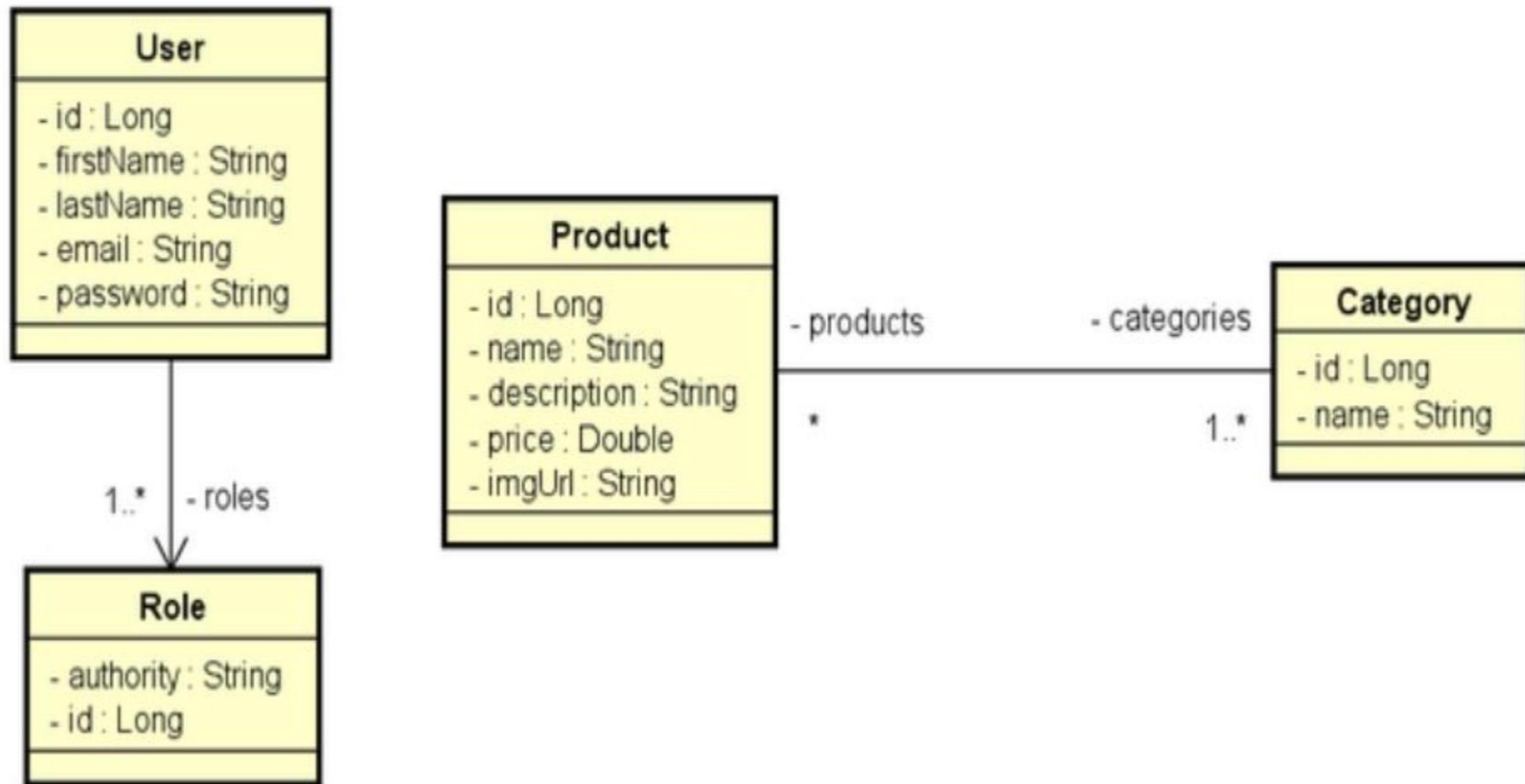
**1**

**0..\***

**1..\***

**\***

# Exemplo de associação (exemplo real)



## Relacionamento entre classes: herança

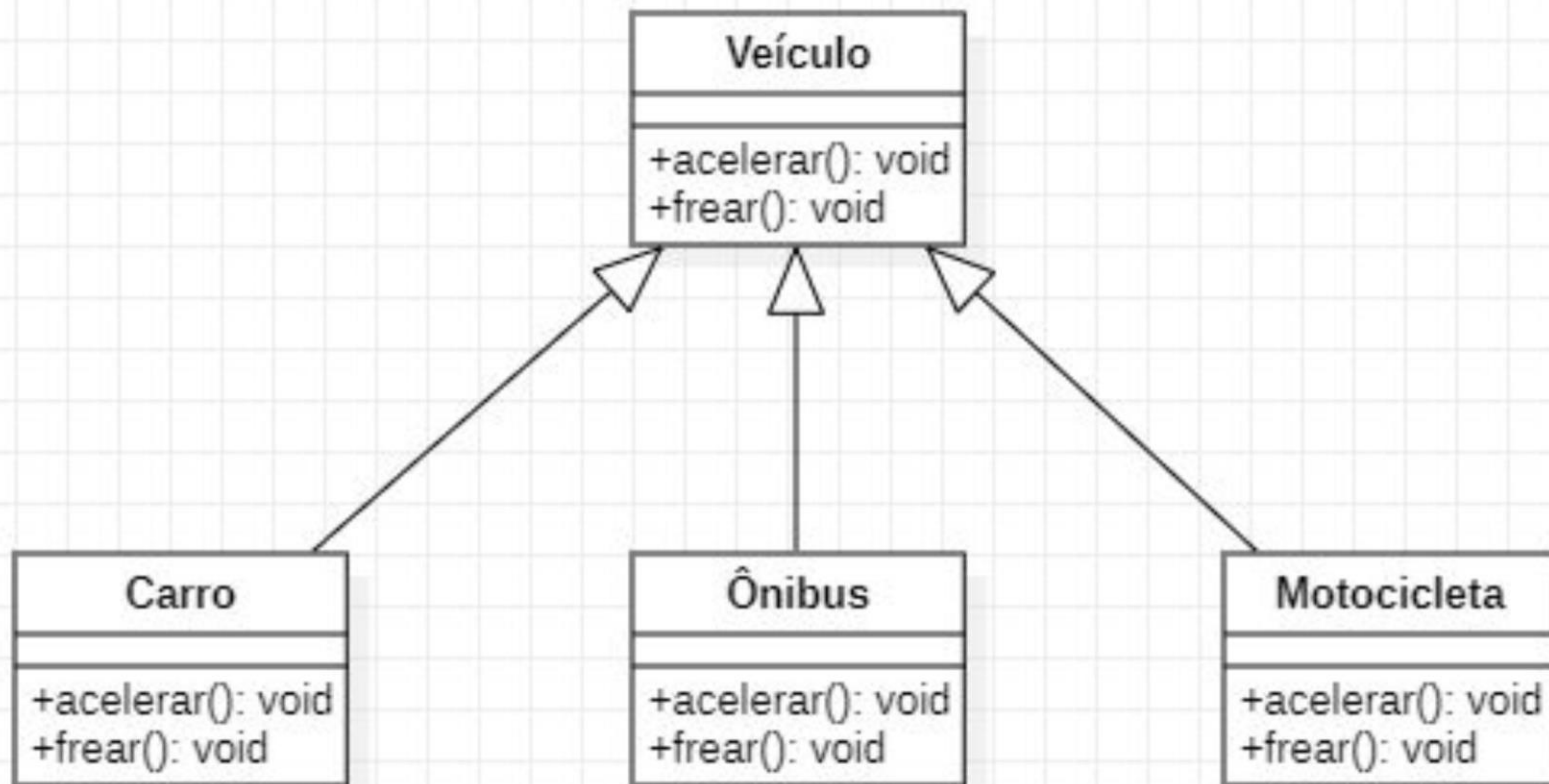
- Em um diagrama de classes, o relacionamento de herança é representado por uma linha sólida com uma seta que se estende de uma classe pai (ou superclasse) para uma classe filha (ou subclasse).



## Relacionamento entre classes: herança

- Esse relacionamento indica que a classe filha herda atributos e métodos da classe pai. A herança é uma parte fundamental da programação orientada a objetos e é usada para criar uma hierarquia de classes, onde as classes filhas herdam as características da classe pai e podem adicionar novos atributos e métodos ou substituir os métodos existentes.

# Exemplo de herança (exemplo real)



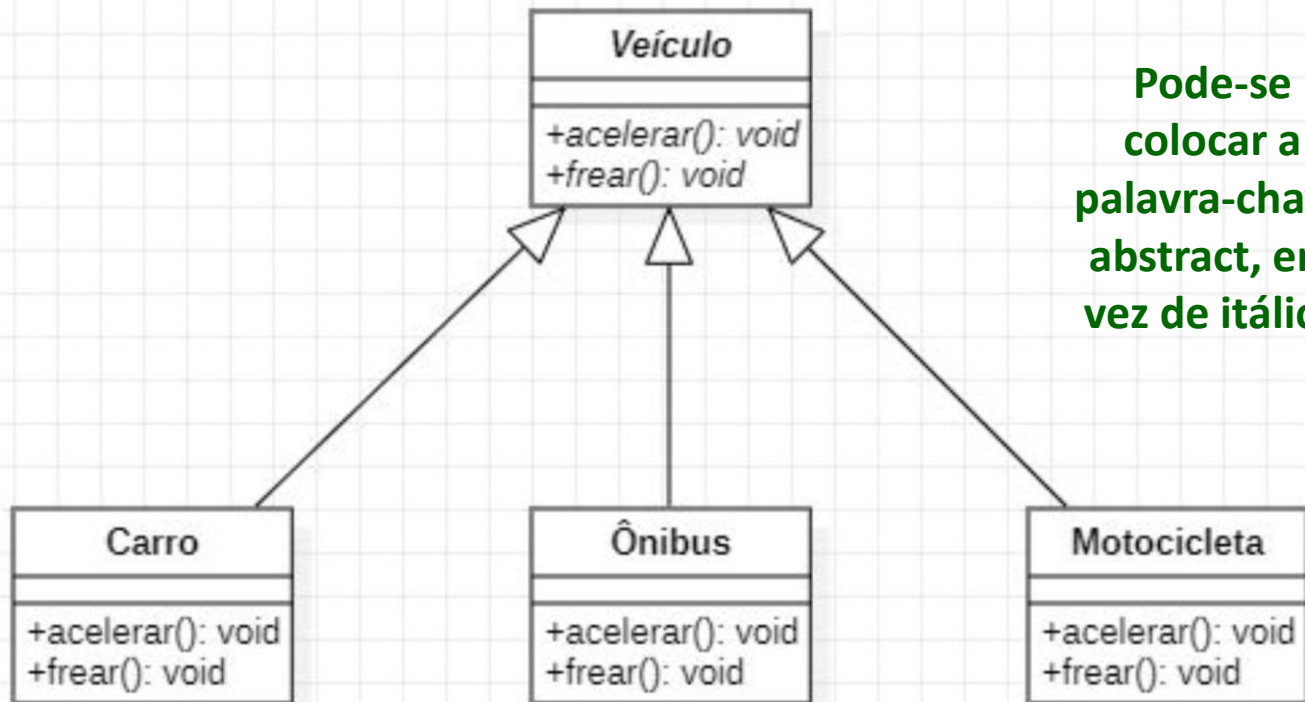
# Exemplo de herança com abstração

- Dependendo dos requisitos e/ou regras de negócio ou ainda da arquitetura do sistema, é viável que as classes pai sejam abstratas. Isso permite com que não seja necessário instanciar objetos da classe pai.

# Exemplo de herança com abstração

- Dessa forma se você tiver uma classe pai que não faz sentido ser instanciada por si só e desejar forçar a implementação de determinados métodos nas classes filhas, é apropriado tornar essa classe pai abstrata. Portanto, a escolha entre tornar uma classe pai abstrata ou concreta depende da semântica do seu design e das necessidades do sistema.
- O nome da classe e dos métodos abstrato esteticamente aparecem como em *itálico*.

# Exemplo de herança com abstração



Pode-se  
colocar a  
palavra-chave  
abstract, em  
vez de *itálico*

## Relacionamento entre classes: agregação/composição

- Em um diagrama de classes, a agregação é um relacionamento que representa uma associação entre classes em que uma classe (a classe agregadora) contém ou "agrega" outras classes (as classes agregadas) como parte de sua estrutura.

## Relacionamento entre classes: agregação/composição

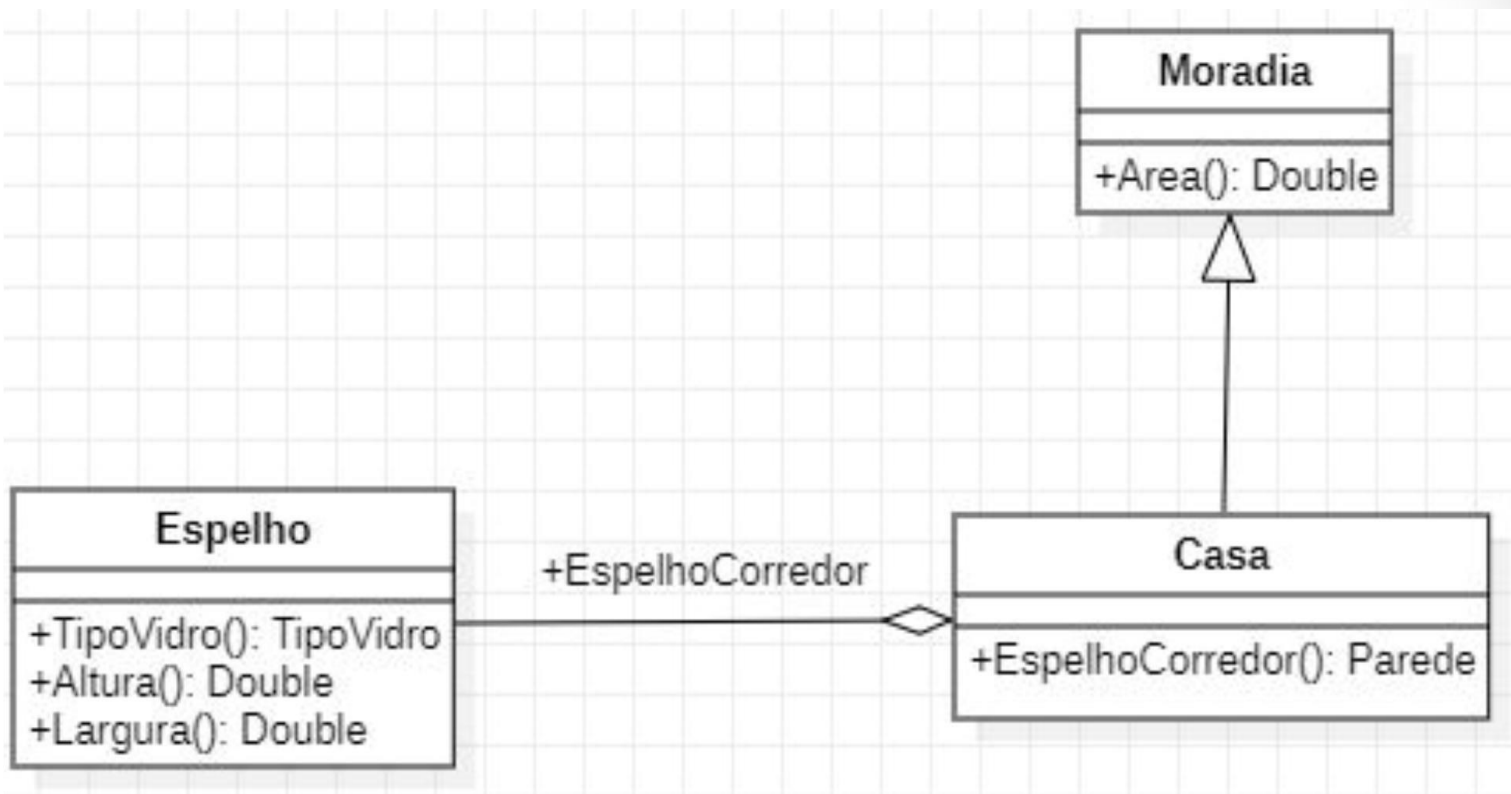
- A agregação é um tipo de relacionamento mais fraco do que a composição.
- Ela indica que as classes agregadas podem existir de forma independente da classe agregadora e não têm uma relação de vida tão forte com a classe agregadora.

## Relacionamento entre classes: agregação/composição

- A agregação é representada por uma linha sólida com um losango vazio no lado da classe agregadora e uma seta que se estende até a classe agregada. A agregação é usada para mostrar que uma classe contém outras classes, mas as classes contidas podem ser compartilhadas entre várias instâncias da classe agregadora.



## Exemplo de agregação (Classe Espelho)



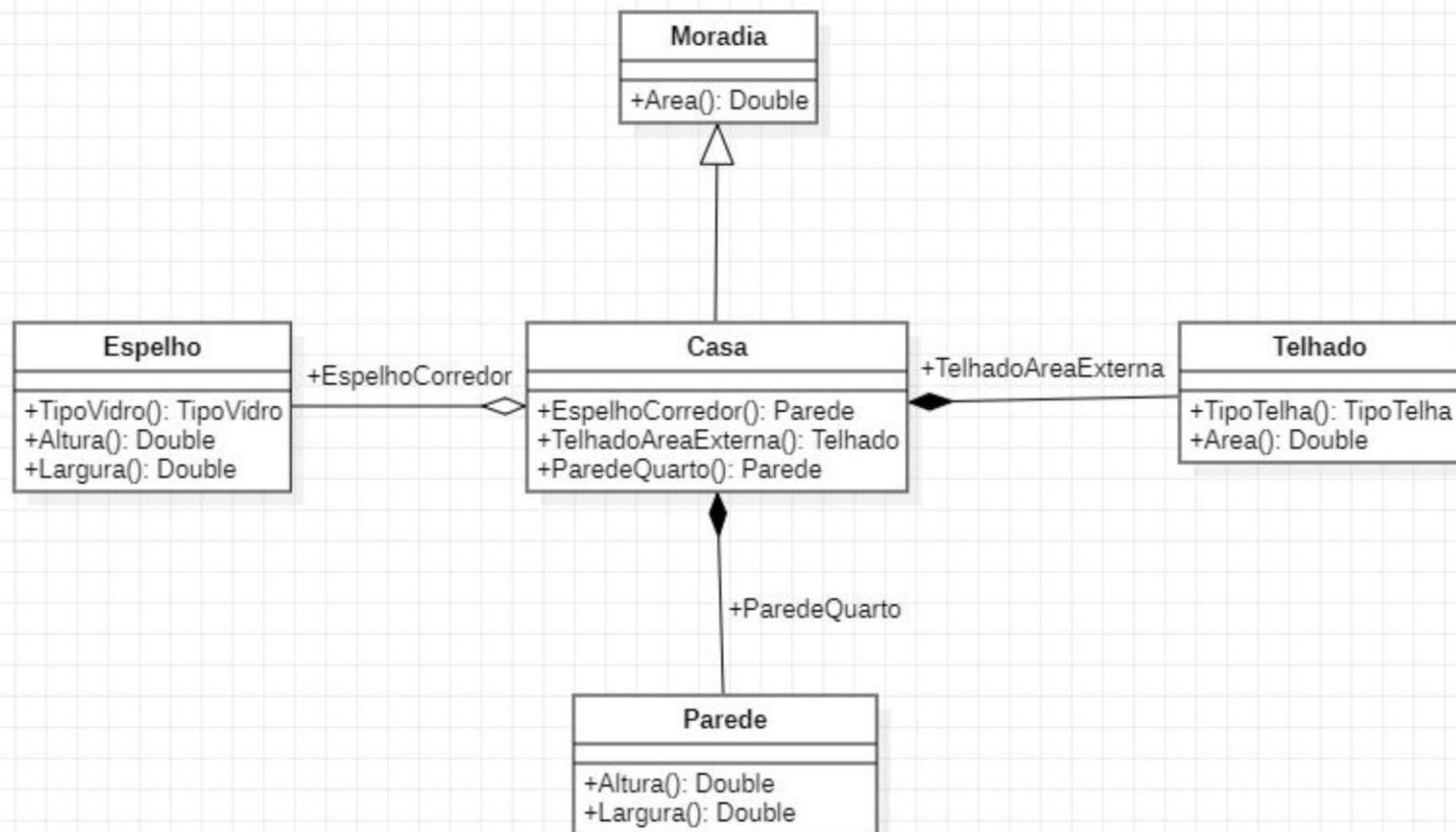
## Relacionamento entre classes: agregação/composição

- Por outro lado, a composição é um relacionamento entre classes que indica que uma classe (à parte) é composta por outras classes (os componentes).
- Esse relacionamento é representado por uma linha sólida com um losango preenchido no lado da parte e uma seta que se estende até a classe componente.

## Relacionamento entre classes: agregação/composição

- A composição indica que a parte não pode existir sem o componente e que a parte possui uma relação forte com o componente.
- A composição é usada quando um objeto de uma classe é composto por outros objetos e a vida útil dos objetos componentes está intimamente ligada à vida útil do objeto da classe principal. Quando o objeto da classe principal é destruído, os objetos componentes também são destruídos.

# Exemplo de composição (Classes Telhado e Parede)



## Relacionamento entre classes: Dependência/Notas

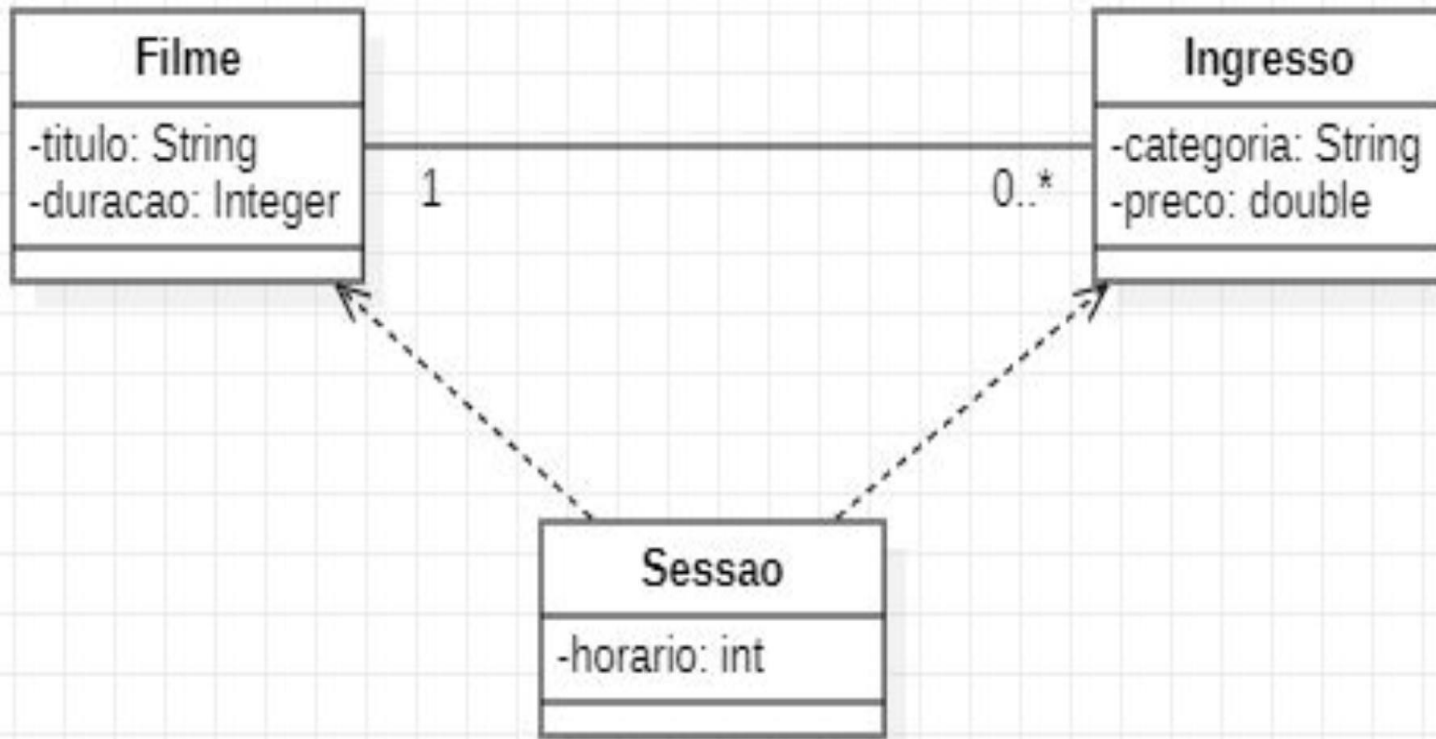
- Em um diagrama de classes, quando vemos uma seta pontilhada com uma linha tracejada ligando uma classe a outra, isso representa uma dependência ou associação. Uma dependência indica que uma classe depende de outra classe de alguma forma, mas essa dependência não implica em um relacionamento direto ou forte entre as classes.

- A dependência pode ocorrer quando uma classe usa ou chama métodos ou atributos de outra classe, mas não possui uma relação de composição, agregação ou herança com essa classe. A dependência é mais fraca que os relacionamentos de composição ou agregação e geralmente ocorre quando uma classe precisa interagir com outra de alguma forma, mas não mantém uma referência direta aos objetos da outra classe.

## Relacionamento entre classes: Dependência/Notas

- A dependência é frequentemente usada para representar colaborações entre classes em um diagrama de classes e pode ser útil para documentar como as classes se relacionam umas com as outras em um sistema.

# Exemplo de dependência entre classes





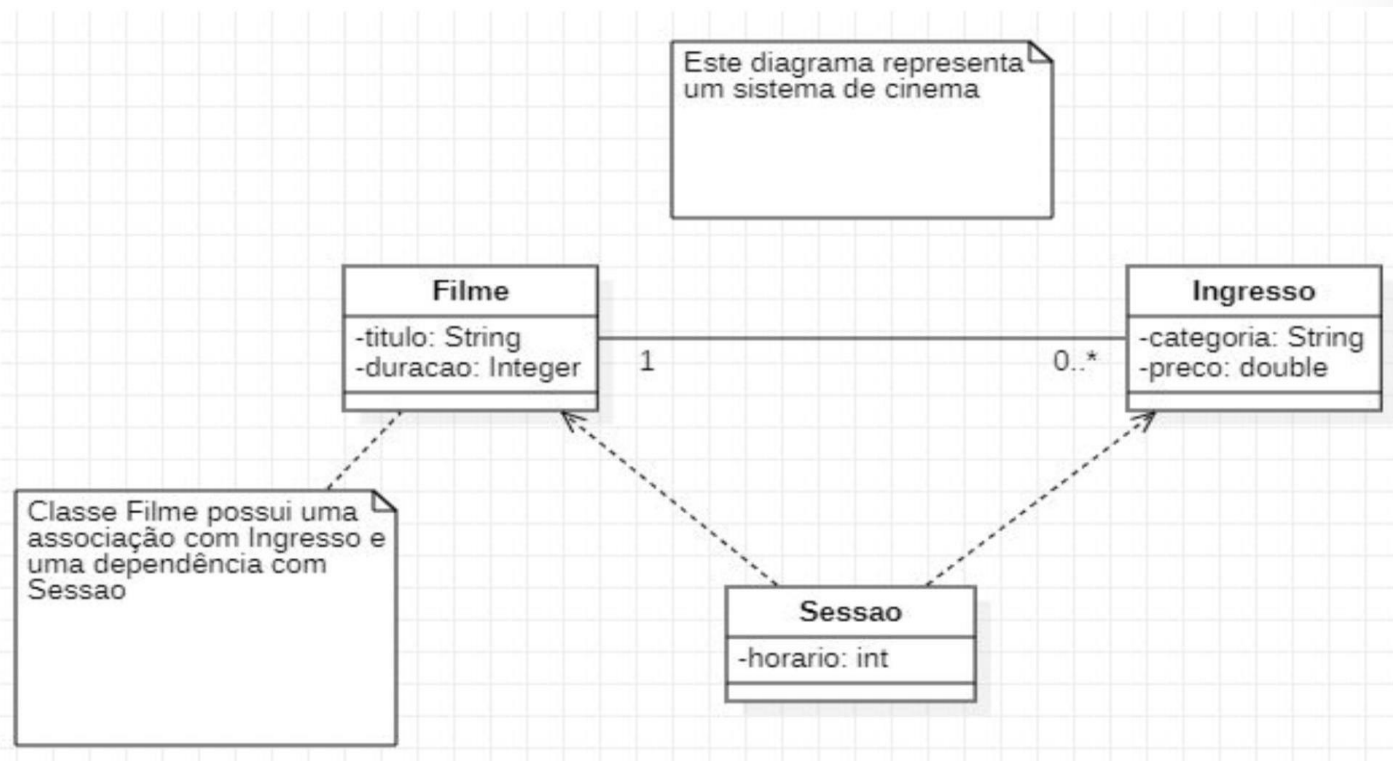
# Notas ou comentários em UML

- As notas são caixas de texto ou comentários que podem ser adicionados para fornecer informações adicionais sobre as classes, relacionamentos, atributos ou métodos representados no diagrama. As notas são usadas para documentar e esclarecer aspectos do diagrama que podem não ser óbvios apenas com os símbolos e linhas usados no diagrama de classes.

# Notas ou comentários em UML

- As notas geralmente são representadas como caixas de texto retangulares conectadas a um elemento do diagrama, como uma classe, relacionamento, atributo ou método, por uma linha tracejada com uma seta. O conteúdo da nota é colocado dentro dessa caixa de texto.

# Exemplo de notas entre as classes



# Tipos enumerados

- Os "tipos enumerados" referem-se a classes que representam tipos de dados que consistem em um conjunto fixo de valores possíveis. Esses valores são geralmente chamados de "enumerações" ou "enum" e são usados para representar categorias discretas de informações.

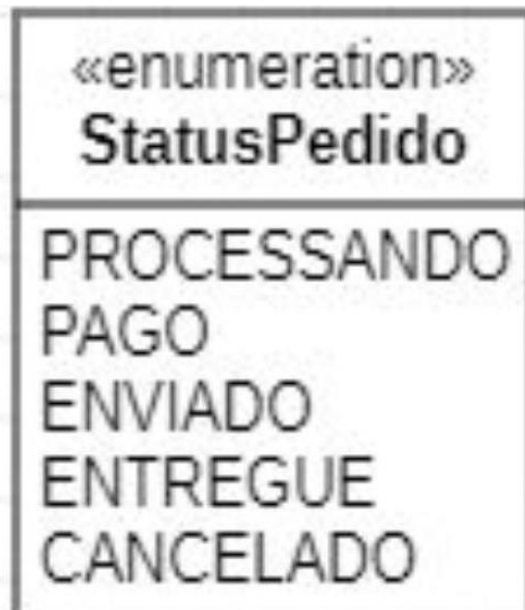
# Tipos enumerados

- Os tipos enumerados são úteis quando você deseja modelar situações em que um atributo, propriedade ou estado pode ter um conjunto específico de valores conhecidos e limitados.

# Tipos enumerados

- A notação para representar um tipo enumerado em um diagrama de classes UML é um retângulo dividido em seções que listam os valores possíveis da enumeração (comum utilizar-se de letras maiúsculas). Cada valor é listado em uma linha separada dentro do retângulo.
- Contém estereótipo de identificação.

# Exemplo de enumerações entre as classes (Enums)



# Interfaces

- As interfaces são elementos que descrevem um conjunto de operações (métodos) que uma classe ou várias classes relacionadas devem implementar.
- As interfaces são usadas para definir contratos que especificam quais métodos devem estar disponíveis em uma classe que implementa essa interface.



# Interfaces

- Em um diagrama de classes, uma interface é representada por um retângulo com o nome da interface dentro dele.
- O nome da interface é precedido por uma linha tracejada horizontal, indicando que se trata de uma interface.
- As interfaces possuem apenas assinaturas de métodos.

# Interfaces

- Contrato: Quando uma classe implementa uma interface, ela se compromete a fornecer implementações para todos os métodos definidos na interface. Isso cria um contrato que garante que a classe terá os métodos necessários disponíveis.

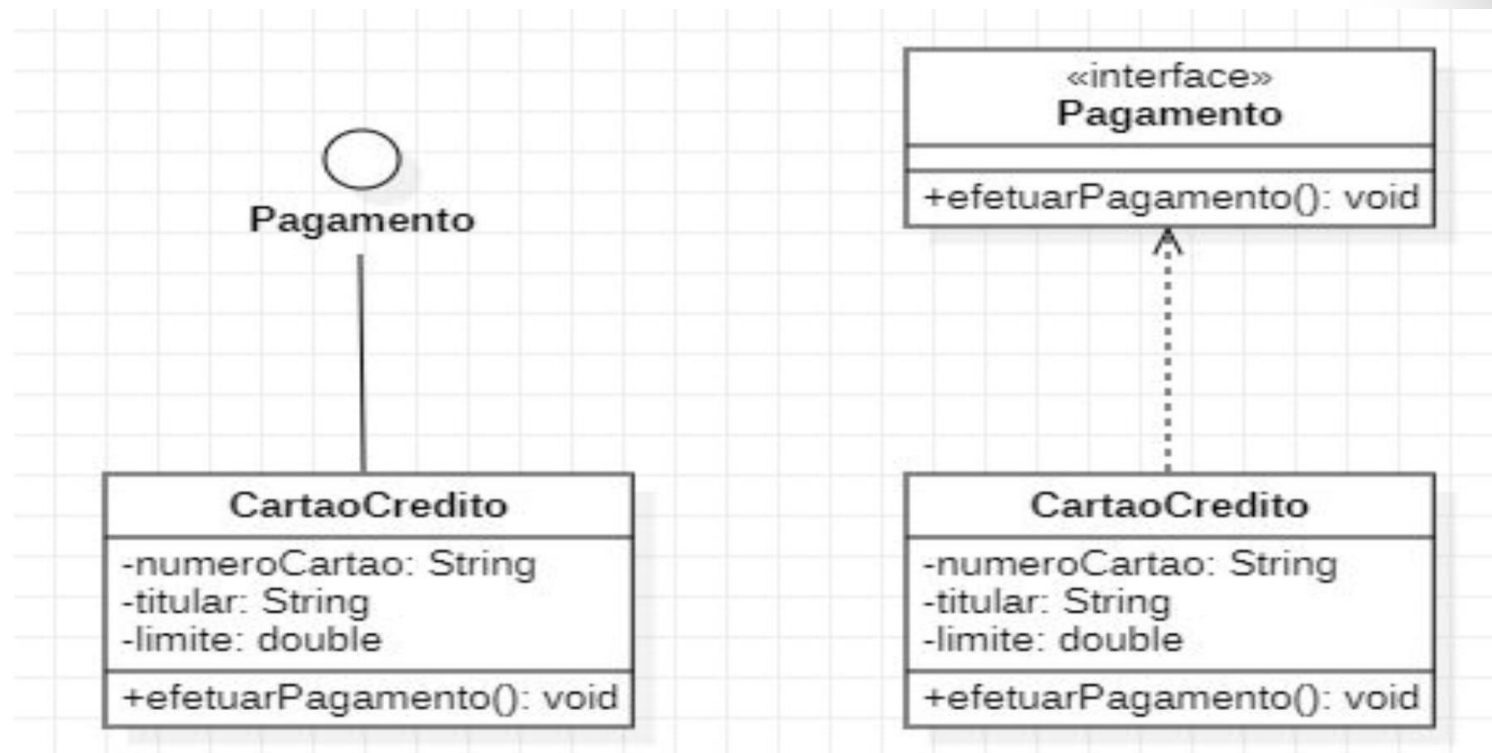
# Interfaces

- Herança Múltipla: Em UML, uma classe pode implementar várias interfaces. Isso permite que a classe herde as operações definidas em várias interfaces diferentes, o que é uma forma de contornar a limitação da herança múltipla, que não é permitida em muitas linguagens de programação, como Java e C#.

# Interfaces

- Uso em Diagramas de Classes: Interfaces são frequentemente usadas em diagramas de classes para modelar relações de implementação entre classes e interfaces. Interfaces são representadas por círculos e/ou retângulo estereotipado.

# Exemplo de interface



# Elementos estáticos

- Os elementos estáticos referem-se a membros de uma classe que pertencem à própria classe em vez de pertencerem a instâncias individuais da classe. Esses elementos não estão associados a objetos específicos da classe, mas sim à classe como um todo. Os elementos estáticos são compartilhados por todas as instâncias da classe e podem ser acessados sem criar uma instância da classe.

# Elementos estáticos

- Os elementos estáticos são representados por texto sublinhado.
- Atributos estáticos (ou variáveis estáticas): São variáveis que mantêm um valor único para toda a classe, independentemente do número de instâncias da classe. Esses atributos são definidos usando a palavra-chave "static" em muitas linguagens de programação, como Java ou C++.

## Exemplo de elementos estáticos

### Biblioteca

-nome: String  
-endereco: String  
-totalLivros: int

+devolverLivro(): boolean



# Constraints/Restrições

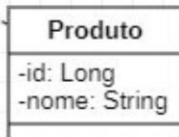
- Uma "constraint" (restrição) é uma especificação adicional que limita ou define as propriedades ou relações de uma classe, associação, operação, entre outros elementos do diagrama. As restrições são usadas para fornecer informações adicionais sobre como as classes e associações devem se comportar ou para impor regras específicas sobre a estrutura ou comportamento dos elementos no diagrama.

# Constraints/Restrições

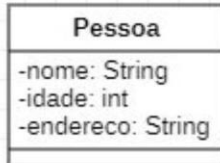
- As constraints são normalmente expressas em linguagens formais, como OCL (Object Constraint Language) ou em texto natural, dependendo do nível de detalhe e da preferência do modelador. Vou fornecer alguns exemplos de implementação de constraints em um diagrama de classes:

# Exemplo de constraints/restrições

{O código do produto deve ser único e auto-incrementável}



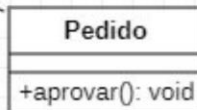
{A idade da pessoa deve estar no intervalo entre 0 e 120 anos}



{nome não deve ser nulo}

{endereco não pode ser nulo}

{O método aprovar pode ser invocado por usuarios com privilégio de admin}



# Constraints/Restrições

- As constraints ajudam a definir as regras e limitações que devem ser seguidas durante a modelagem e implementação do sistema.
- Elas também podem ser usadas para documentar requisitos específicos do sistema, garantindo que o software seja desenvolvido de acordo com as especificações definidas no diagrama de classes.

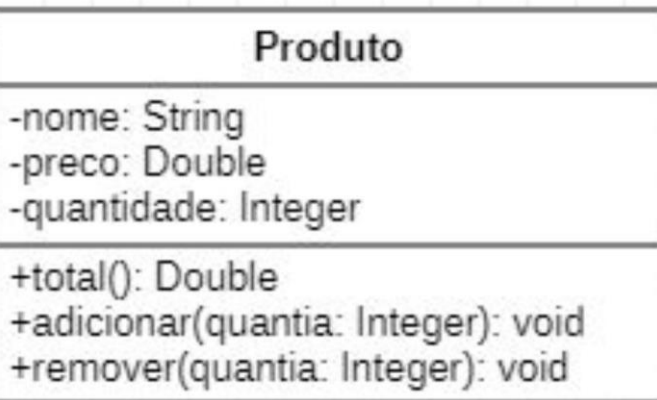


# Exemplo de Diagrama de Classe e a implementação em Java

# Exemplo de Diagrama de Classe

## Nome da Classe: Produto

Atributos



Métodos



**Observação:** no Diagrama de Classe acima não foram colocados o método do tipo construtor para simplificar o diagrama.

# Implementação em Java da Classe Produto

```
package exemplo;

public class Produto{
    private String nome;
    private double preco;
    private int quantidade;

    //Método Construtor

    public Produto(String nome, double preco, int quantidade){
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
    }
}
```

# Implementação em Java da Classe Produto

```
public double total(){
    return this.preco * this.quantidade; }

public void adicionar (int quantia) {
    this.quantidade = this.quantidade + quantia;
    System.out.println(this.quantidade); }

public void remover(int quantia){
    if(this.quantidade >= quantia){
        this.quantidade = this.quantidade - quantia;
        System.out.println(this.quantidade);
    } else
        System.out.println("Quantidade menor que a quantia passada");
} } //fim classe Produto
```



# Implementação da Classe de Teste

```
package aplicacao;  
  
import exemplo.Produto;  
  
public class TestaProduto {  
  
    public static void main(String[] args){  
        Produto p1 = new Produto("PC", 5009.0, 2);  
        Produto p2 = new Produto("Teclado", 100.0, 5);  
        Produto p3 = new Produto("Smartphone", 1500.0, 20);  
        double totalP1 = p1.total();  
        System.out.println(totalP1);  
        p1.adicionar(3);  
        p1.remover(2);  
    }  
}
```

# Implementação da Classe de Teste

```
double totalP2 = p2.total();
```

```
System.out.println(totalP2);
```

```
p2.adicionar(4);
```

```
p2.remover(3);
```

```
double totalP3 = p3.total();
```

```
System.out.println(totalP3);
```

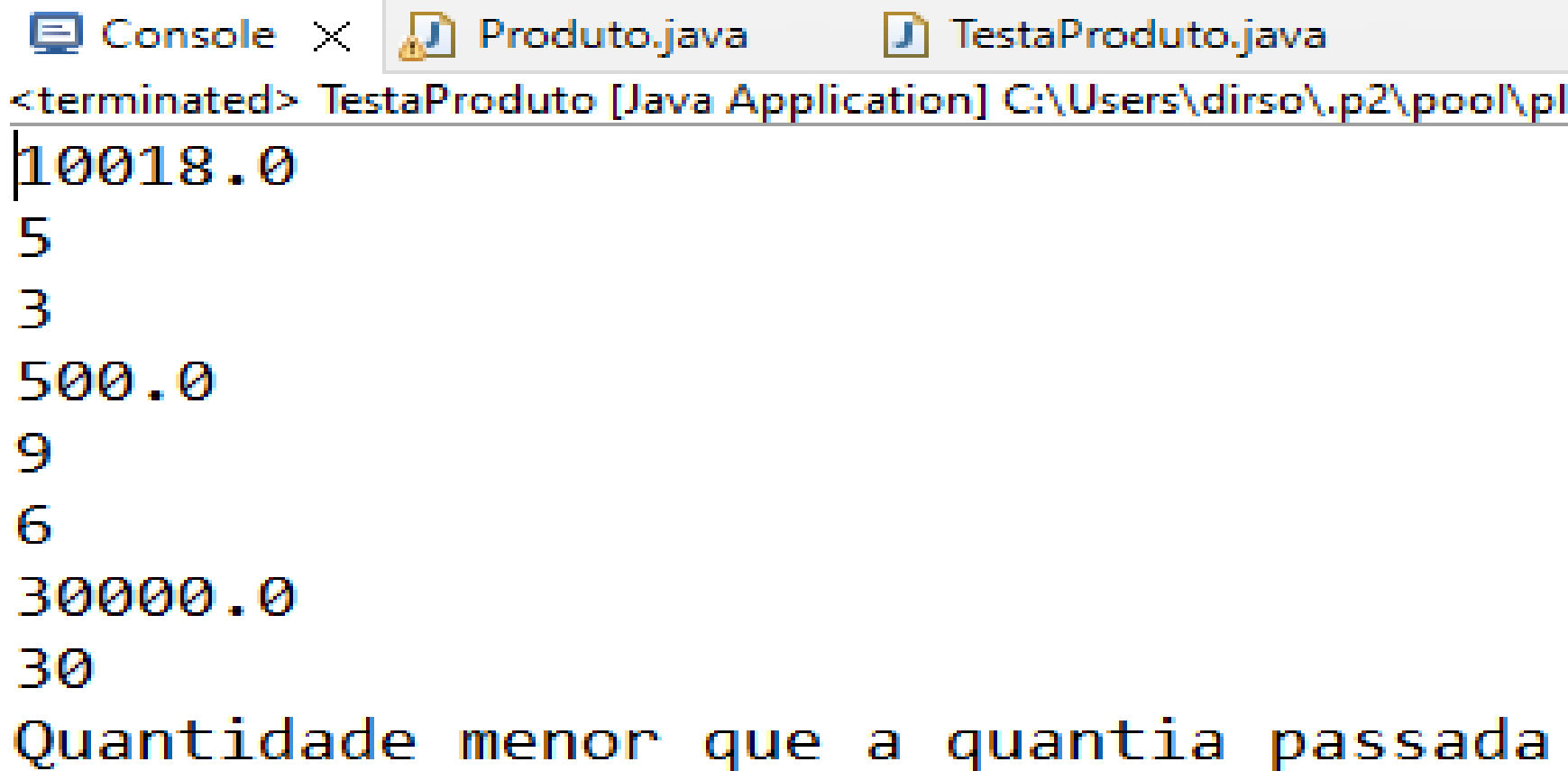
```
p3.adicionar(10);
```

```
p3.remover(40);
```

```
} // fim método main
```

```
} // fim classe TestaProduto
```

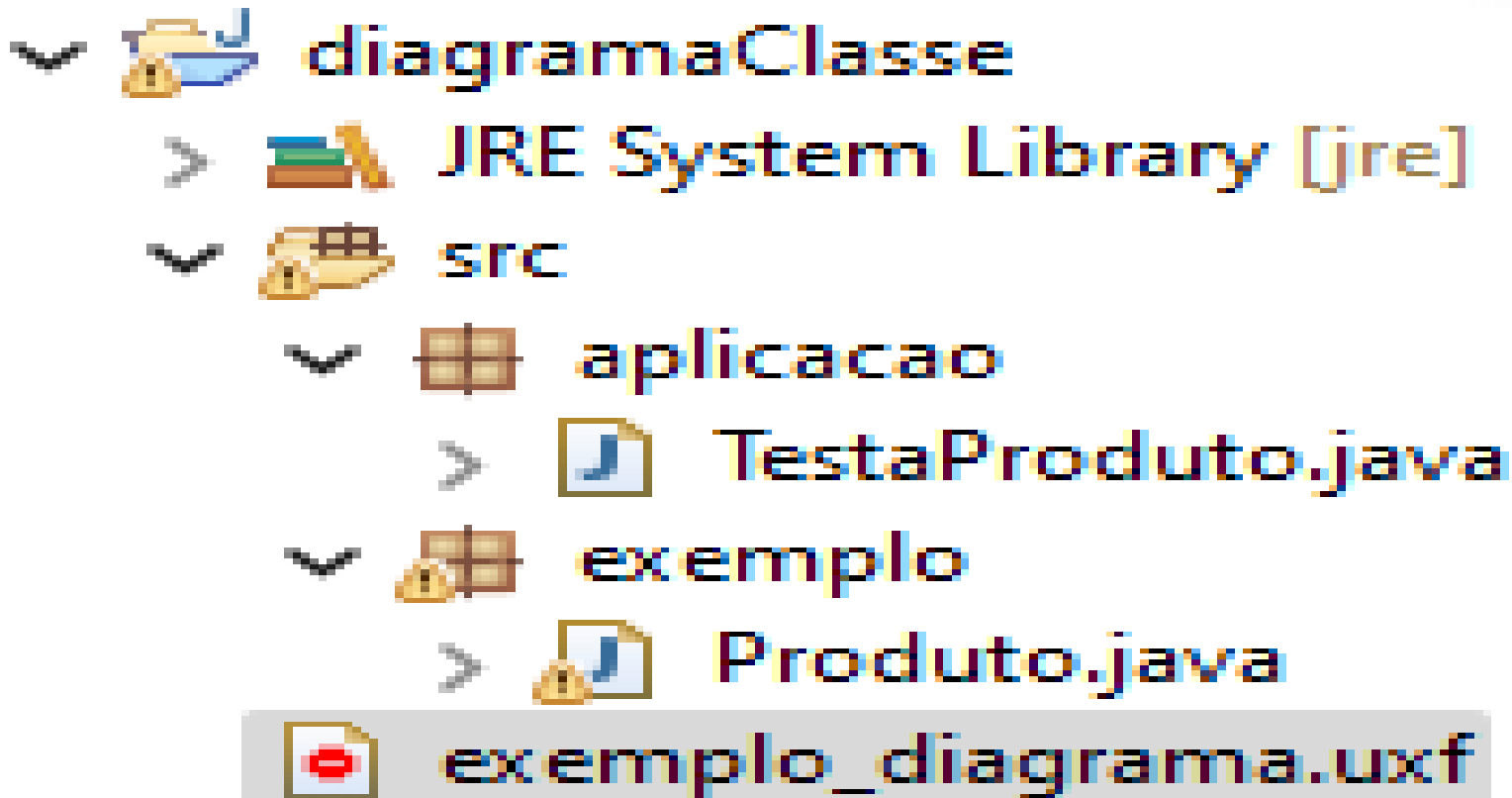
## Executando a Classe de Teste



The screenshot shows a Java IDE with two tabs: 'Console' and 'TestaProduto.java'. The console output displays the results of a test execution, including numerical values and an error message.

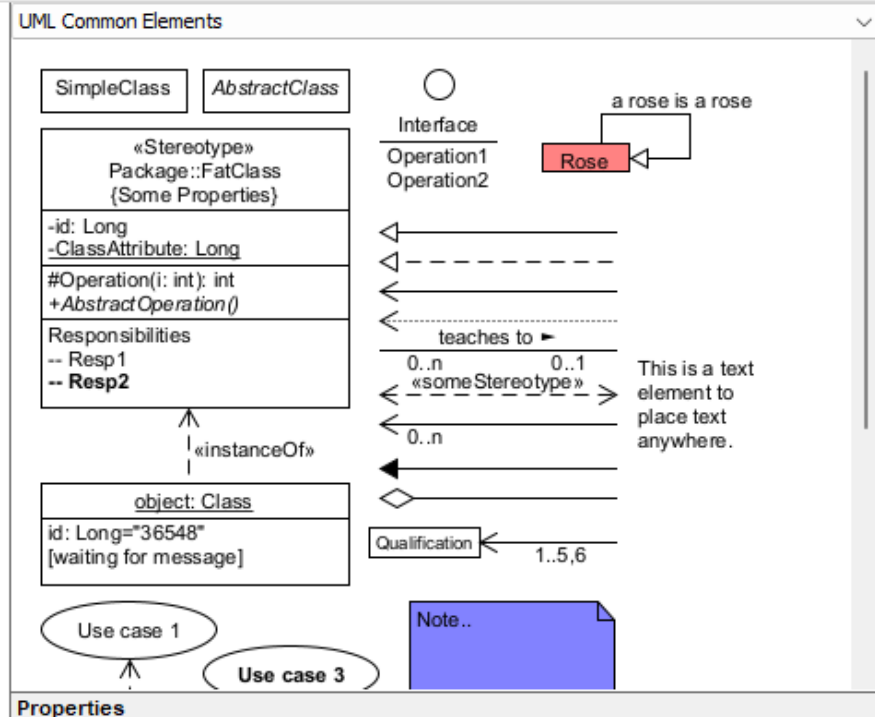
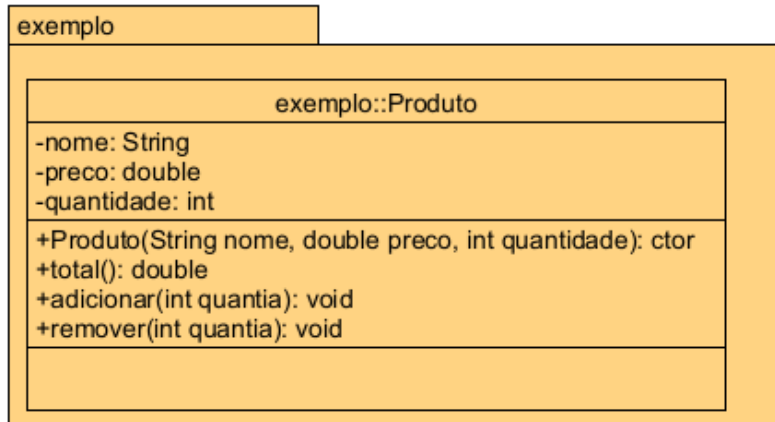
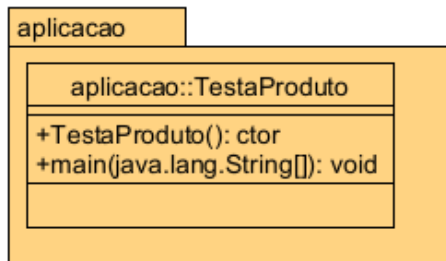
```
<terminated> TestaProduto [Java Application] C:\Users\dirso\.p2\pool\pl  
10018.0  
5  
3  
500.0  
9  
6  
30000.0  
30  
Quantidade menor que a quantia passada
```

# Estrutura do Projeto Java no Package Explorer do IDE Eclipse

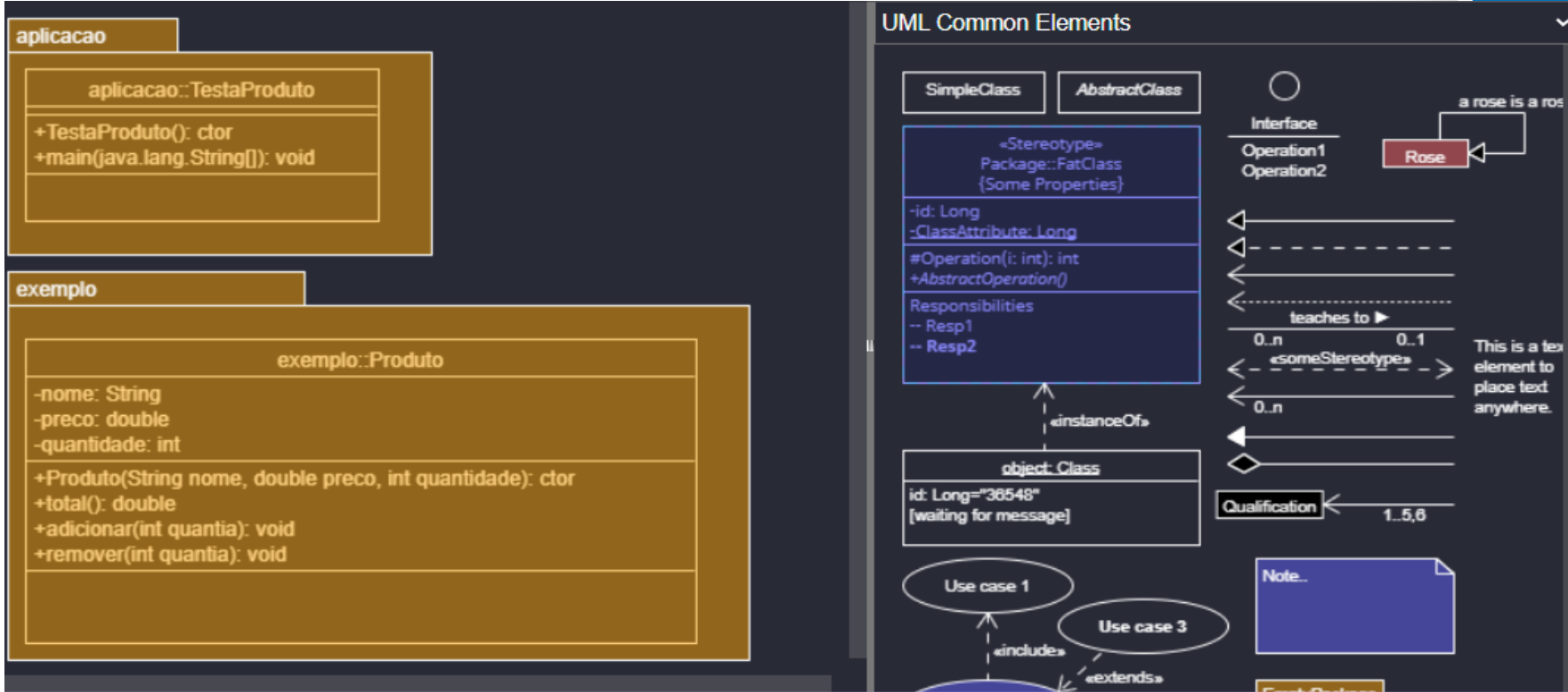


# Diagrama de Classe no UMLet (exemplo\_diagrama.uxf no IDE Eclipse)

Produto.java TestaProduto.java exemplo\_diagrama.uxf X



# Diagrama de Classe no UMLet (exemplo\_diagrama.uxf no IDE Visual Studio Code)



# Pacote do Projeto no Eclipse

- Disponibilizado no arquivo .zip na página da disciplina no SIGAA.