Advanced Institute for Artificial Intelligence

Programação em Python

https://advancedinstitute.ai

Programação Python

Projeto de Software

Referências e Fontes das Imagens

- UML2: Uma Abordagem Prática

Classificação, Abstração e Instanciação

> *No início da infância, o ser humano aprende e pensa de maneira bastante semelhante à filosofia da orientação a objetos, representando seu conhecimento por meio de abstrações e classificações.*

- Intuição de classes como **grupos de objetos com as mesmas características e comportamentos**
  - E.g.: qualquer peça de metal com quatro rodas que se locomova de um lugar para outro transportando pessoas recebe a *denominação de carro*.
    - Esforço de abstração: carros apresentarem diferentes formatos, cores e estilos;

UML2: Uma Abordagem Prática

Classes, Atributos e Objetos

- Quando objeto é instanciando a partir de uma classe, estamos **criando um novo item do conjunto representado por essa classe**;
- Possuem as **mesmas características e comportamentos** de todos os outros objetos já instanciados;
  - Possuem mesmos atributos porém **não são exatamente iguais**: cada objeto armazena valores diferentes em seus atributos:
  - Pessoas possuem valores diferentes para `CPF`, `nome`, `telefone`, etc.
- Uma classe representa uma **categoria** e os objetos são os **membros ou exemplos** dessa categoria

Projeto de Software

- **Análise Orientada a Objetos**
  - Identificar os **objetos e as interações** entre esses objetos
  - Sobretudo, sobre o **que precisa ser feito**.
    - Entrevistar clientes, estudar processos, eliminar possibilidades;
    - **Requisitos!**
- **Projeto Orientado a Objetos**
  - Processo de conversão de **requisitos em uma especificação de implementação**
  - Descrever uma coleção de objetos que interagem por meio de seus **dados e comportamento**.
  - Output: **especificação de implementação**
    - **Classes e Interfaces** independente de linguagem de programação.

Projeto de Software

- **Programação Orientada a Objetos**
  - Converter Projeto Orientado a Objetos em um **software funcional**.
  - Nem sempre é possível um **mapeamento 1-para-1 entre design e implementação**;
- No mundo real as **coisas são mais complicadas**:
  - Sempre encontraremos **coisas que precisam de mais análise durante o projeto**;
  - Descobrimos p**artes do projeto que precisam ser mais claras, durante a implementação**;
- **Desenvolvimento iterativo**:
  - **Pequena parte da tarefa** é modelada, projetada e programada
  - Incluir novos recursos em uma **série de curtos ciclos de desenvolvimento**;

Linguagem de Modelagem Unificada - UML

Linguagem de Modelagem Unificada - UML

- União de três métodos orientados a objeto: o método de Booch, o método OMT (Object Modeling Technique) o método OOSE (Object-Oriented Software Engineering);
- Lançamento, em 1996, da primeira versão da UML;
- Versão 2.0 lançada em julho de 2005;
- Atualmente a UML encontra-se na versão 2.5 (lançada em 2015)
- 23 Diagramas agrupados em **Estruturais** e **Comportamentais**
  - **Nem todos diagramas são necessários no projeto de um dado software**;
  - **Múltiplas visões** do sistema a ser modelado, analisando-o e modelando-o sob **diversos aspectos**;

Diagrama de Classes

- Um dos mais importantes e utilizados da UML;
- **Visualização das classes** que fazem parte do sistema com seus **atributos e métodos**;
- Mostrar **relacionamento entre classes**;
- **Visão estática**;
- **Construído durante a análise** e refinado durante o projeto;
  - **Modelo Conceitual** - informações necessárias ao software;
  - **Modelo de Domínio** - focado na solução do problema;
- **Vários modelos de classe** em um mesmo projeto de software com **enfoques diferentes**;

Diagrama de Classes

- Notação:

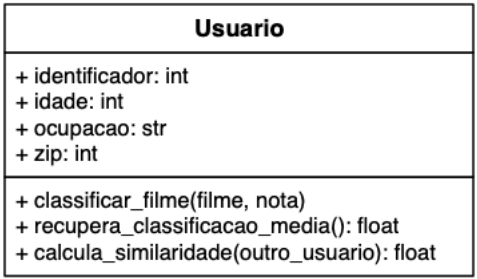
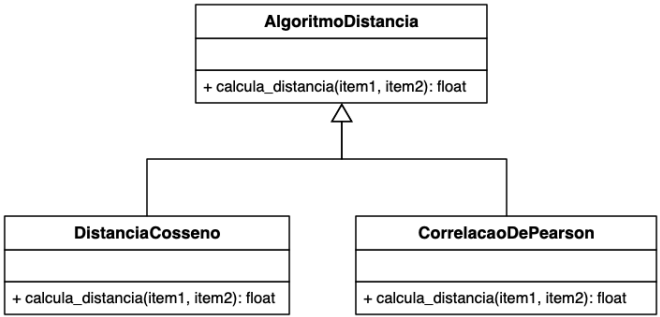


Diagrama de Classes

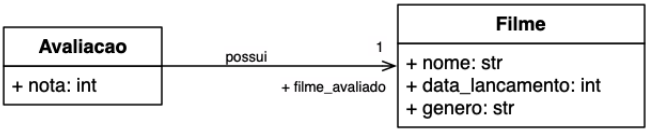
- **Atributos**:
  - Peculiaridades que costumam **variar de um objeto para outro**
  - Segunda divisão da classe e contêm **Nome e Tipo de dado**
  - Visibilidade **+** (**pública**), **#** (**protegida**) e **–** (**privada**)
- **Métodos**:
  - Atividade que um **objeto de uma classe pode executar**
  - Terceira divisão da classe e contêm assinatura do método (**visibilidade; nome; parâmetros** - contendo tipo ou não; e **tipo de retorno**
- Podem ser simplificada a **uma única divisão** contendo somente o nome da classe;

Diagrama de Classes - Herança



- Relação de **Generalização/Especialização**

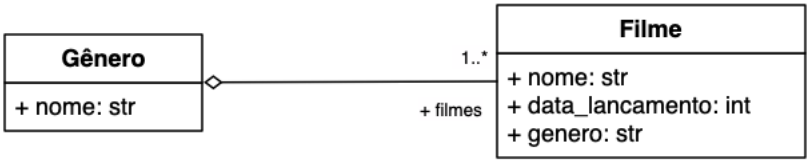
Diagrama de Classes - Associações



- **Navegabilidade**: determina que os objetos da classe para onde a seta aponta não têm conhecimento dos objetos aos quais estão associados na outra extremidade da associação;
- **Multiplicidade**: quantidade de instâncias presentes na classe oposta a seta;
  - 1, *, 1..*, 0..1, m..n
- **Papéis**: nome e visibilidade do atributo na classe oposta a seta;

Diagrama de Classes - Relacionamento *Todo-Parte*

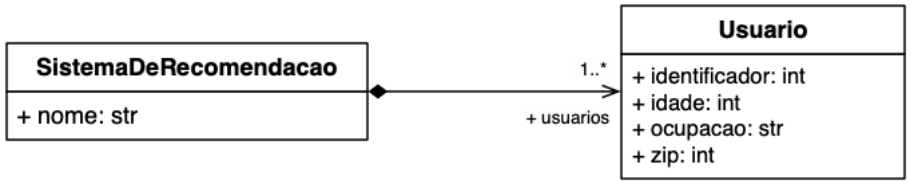
- **Agregação**:



- *Objetos-todo* que precisam ter suas informações complementadas pelos *objetos-parte*
- *Objetos-parte* continuam fazendo sentido mesmo após a destruição do *objeto-todo*

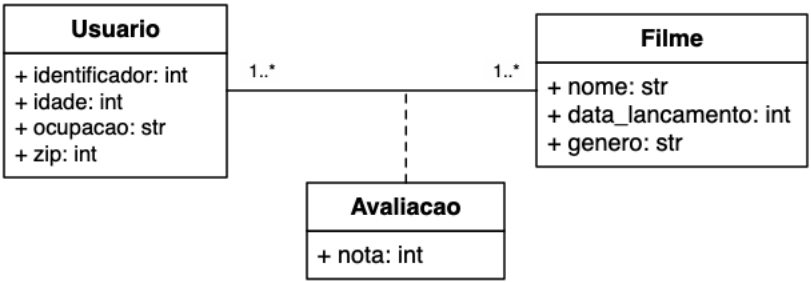
Diagrama de Classes - Relacionamento *Todo-Parte*

- **Composição**:



- *Objetos-parte* têm de estar **associados a um único *objeto-todo***
- **Não podem ser destruídos por um objeto diferente do *objeto-todo***;
- *Objetos-parte* **não fazem sentido após a destruição** do *objeto-todo*

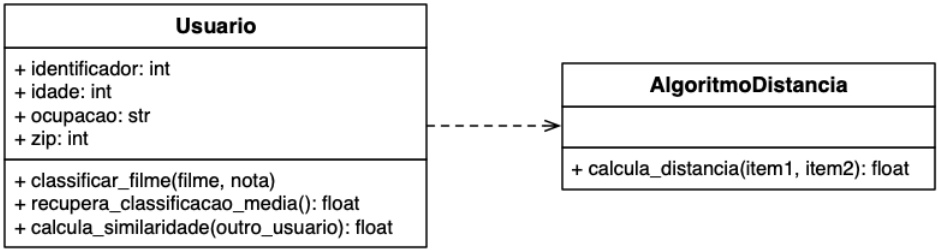
Diagrama de Classes - Classes Associativas



- Atributos que **pertencem a associação**;
  - Não podem ser armazenados em **nenhuma das classes envolvidas**;

Diagrama de Classes - Dependência

- Identifica certo grau de dependência de um elemento em relação à outro
- Relacionamento fornecedor/cliente entre elementos do modelo



Dúvidas?