Design Patterns

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

Ementa

- Conceitos fundamentais de design patterns;
- Padrões de criação, estruturais e comportamentais;
- Aplicação de design patterns na arquitetura de software;
- Princípios SOLID e boas práticas de design orientado a objetos;
- Refatoração e melhoria de código com o uso de design patterns

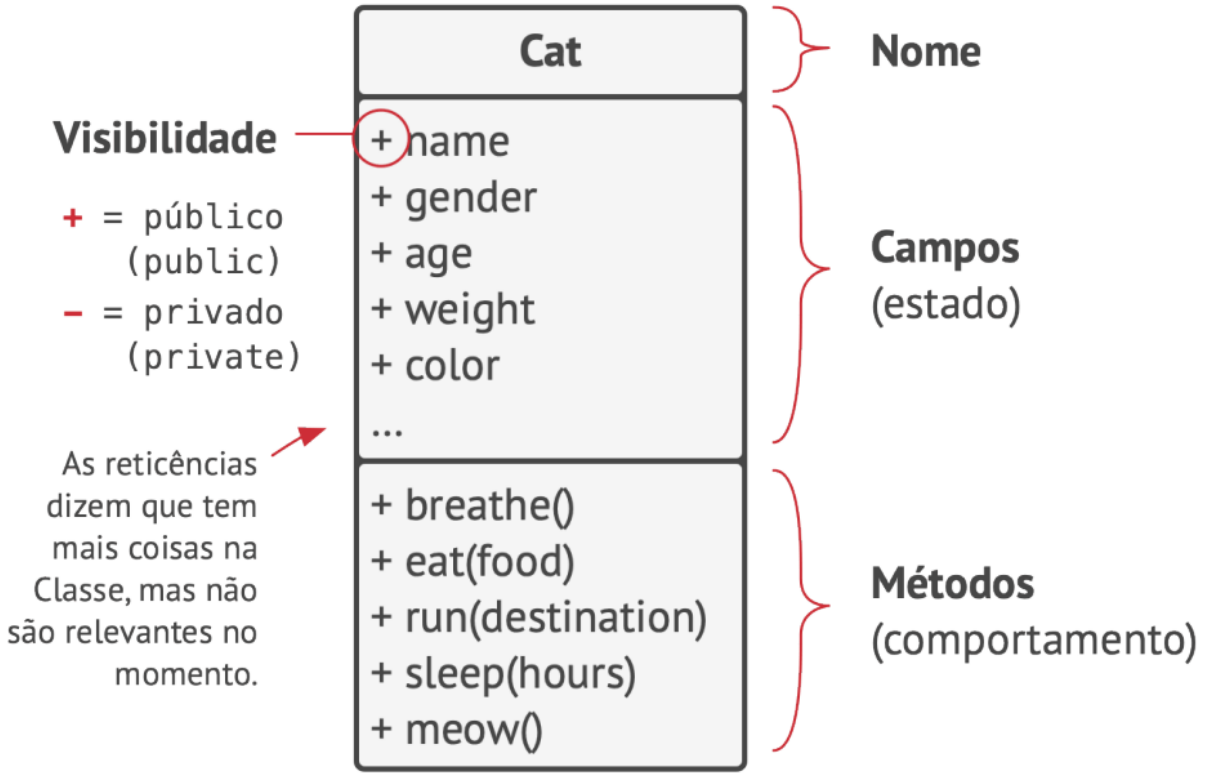
Revisão a Programação Orientada a Objetos

Definição

A Programação Orienteda à Objetos (POO) é um paradigma baseado no conceito de envolver pedaços de dados, e comportamentos relacionados aqueles dados, em uma coleção chamada objetos, que são construídos de um conjunto de “planos de construção”, definidos por um programador, chamados de classes.

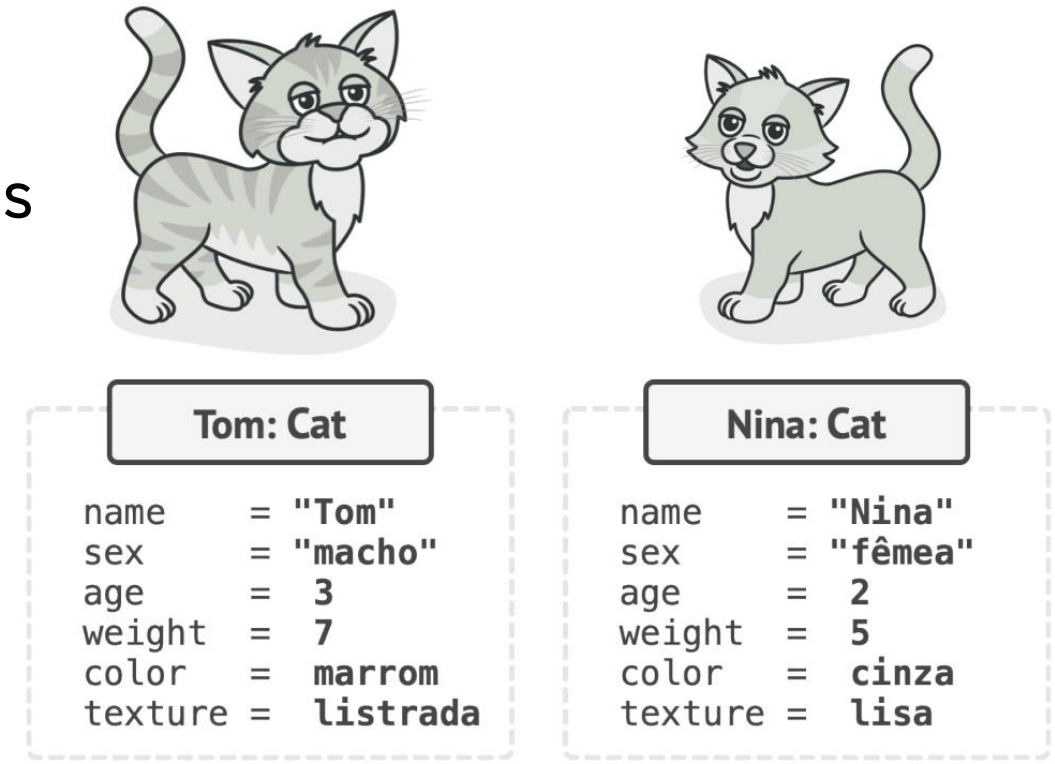
Objetos, classes

Todos os gatos também se comportam de forma semelhante: eles **respiram, comem, correm, dormem, e miam**. Estes sãos os métodos da classe. Coletivamente, os campos e os métodos podem ser referenciados como membros de suas classes.



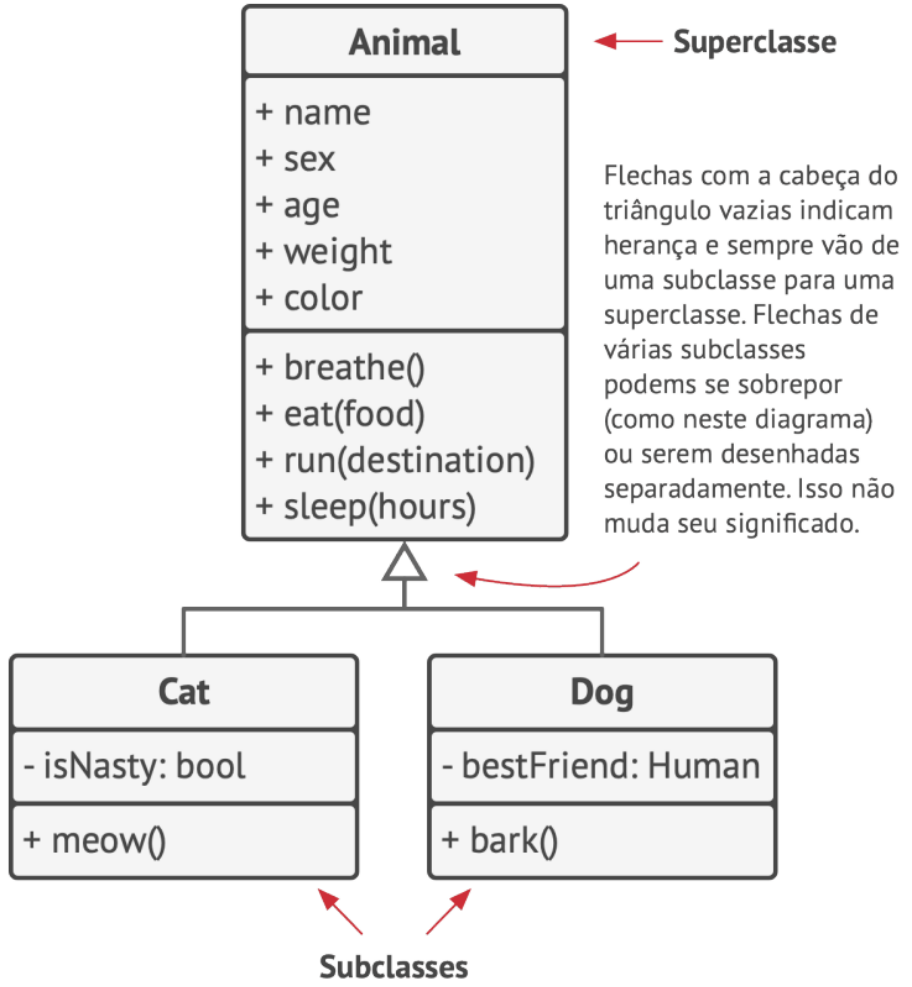
Objetos, classes

- Dados armazenados dentro dos campos do objeto são referenciados como estados, e todos os métodos de um objeto definem seu comportamento.
- Então uma classe é como uma planta de construção que define a estrutura para objetos, que são instâncias concretas daquela classe.



Hierarquias de classe

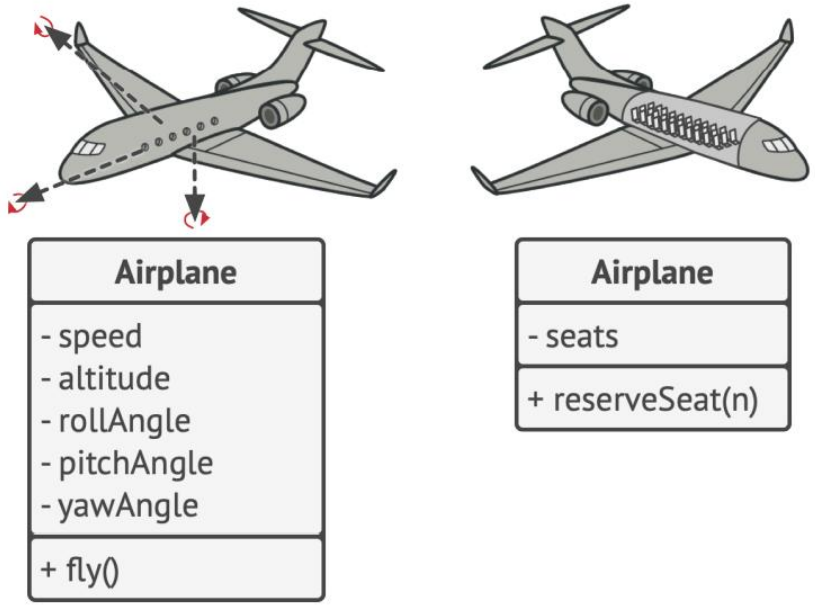
- Uma classe mãe é chamada de uma superclasse. Suas filhas são as subclasses. Subclasses herdam estado e comportamento de sua mãe, definindo apenas atributos e comportamentos que diferem. Portanto, a classe Gato teria o método miado e a classe Cão o método latido.



Pilares da POO

Abstração

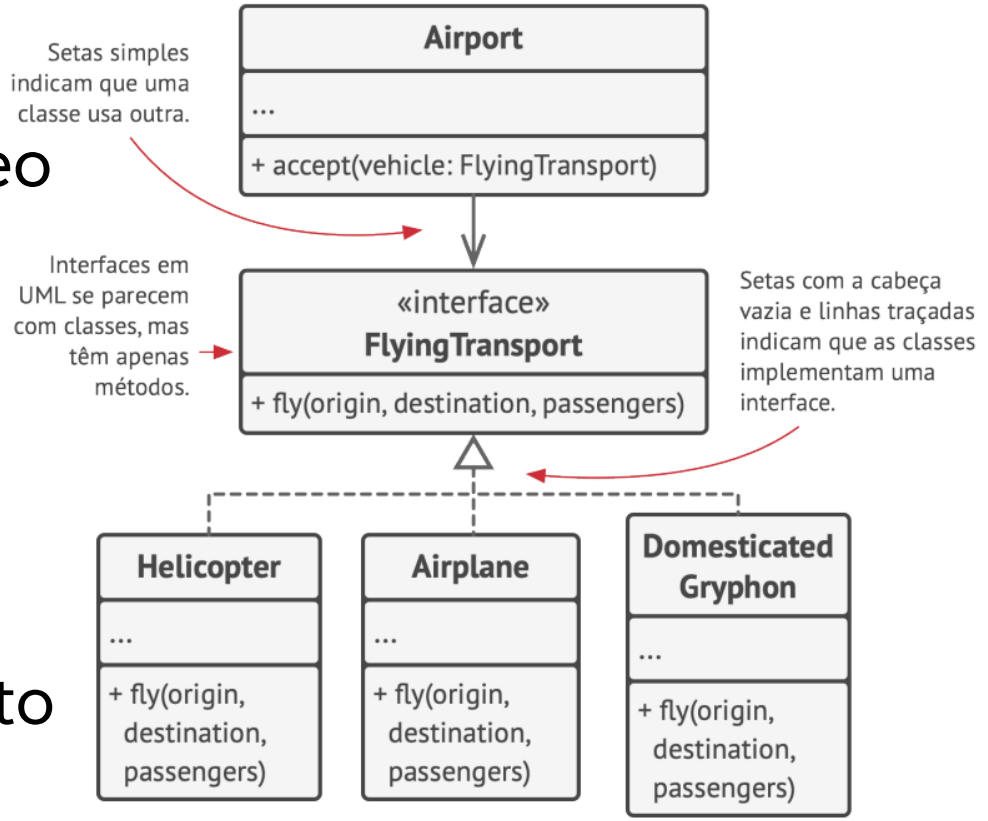
- Uma classe Avião poderia provavelmente existir em um simulador de voo e em uma aplicação de compra de passagens aéreas. Mas no primeiro caso, ele guardaria detalhes relacionados ao próprio voo, enquanto que no segundo caso você se importaria apenas com as poltronas disponíveis e os locais delas.



Diferentes modelos de um mesmo objeto real.

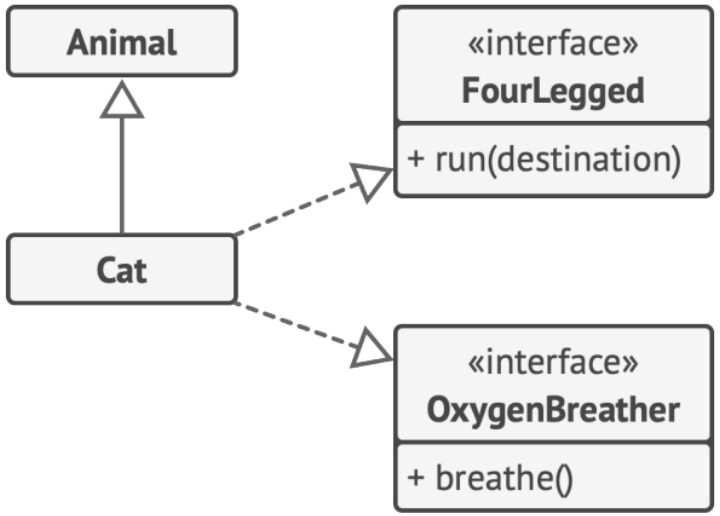
Encapsulamento

- Se em um simulador de transporte aéreo você restringiu a classe Aeroporto para trabalhar apenas com objetos que implementam a interface Transporte Aéreo .
- Após isso, você pode ter certeza que qualquer objeto passado para um objeto aeroporto, seja ela um Avião ou um Helicóptero, todos serão capazes de aterrissar ou decolar deste tipo de aeroporto.



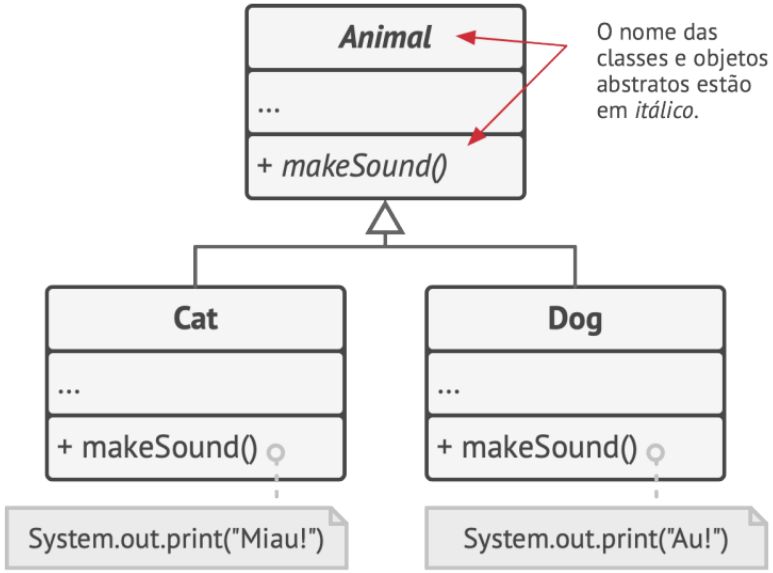
Herança

- A Herança é a habilidade de construir novas classes em cima de classes já existentes. O maior benefício da herança é a reutilização de código. Se você quer criar uma classe que é apenas um pouco diferente de uma já existente, não há necessidade de duplicar o código. Ao invés disso, você estende a classe existente e coloca a funcionalidade adicional dentro de uma subclasse resultante, que herdará todos os campos de métodos da superclasse.

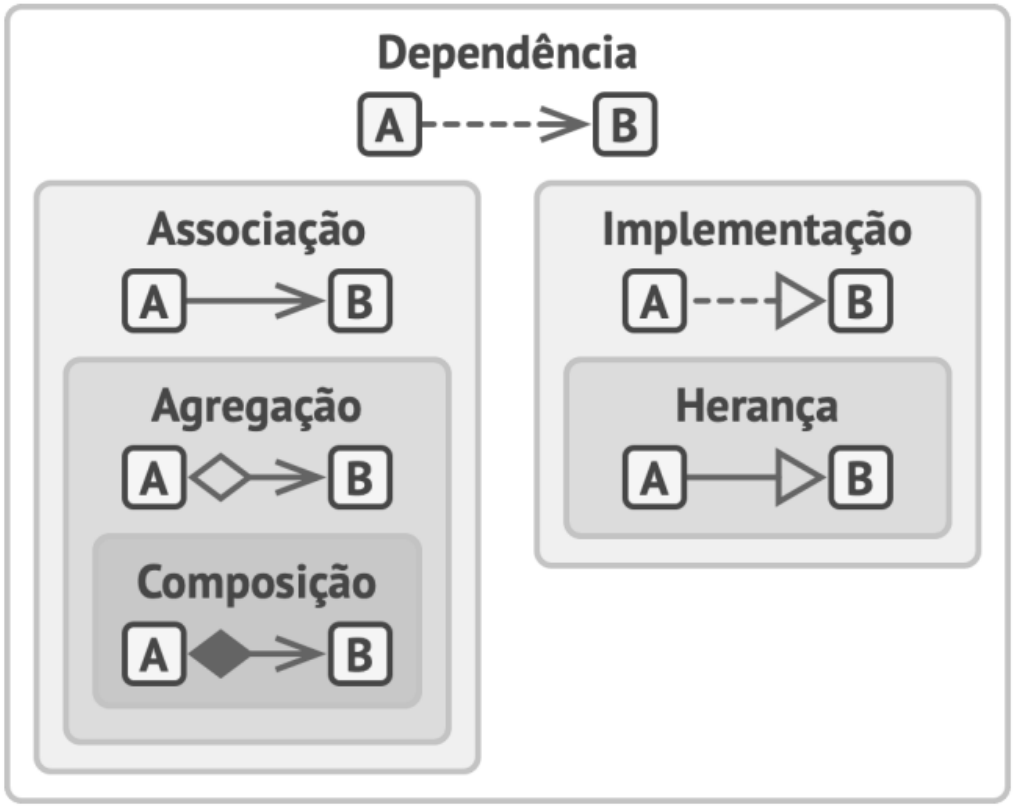


Polimorfismo

- A maioria dos Animais podem produzir sons. Nós podemos antecipar que todas as subclasses terão que sobrescrever o método base produzirSom para que cada subclasse possa emitir o som correto;
- Portanto nós podemos declará-lo abstrato agora mesmo. Isso permite omitir qualquer implementação padrão do método na super classe, mas força todas as subclasses a se virarem com o que têm.



Relações entre objetos



Relações entre objetos e classes: da mais fraca a mais forte.

Relações entre objetos

- A **dependência** é o mais básico e o mais fraco tipo de relações entre classes. Existe uma dependência entre duas classes se algumas mudanças na definição de uma das classes pode resultar em modificações em outra classe. A dependência tipicamente ocorre quando você usa nomes de classes concretas em seu código.
- Geralmente, um diagrama UML não mostra todas as dependências.

UML da Dependência. O professor depende dos materiais do curso.

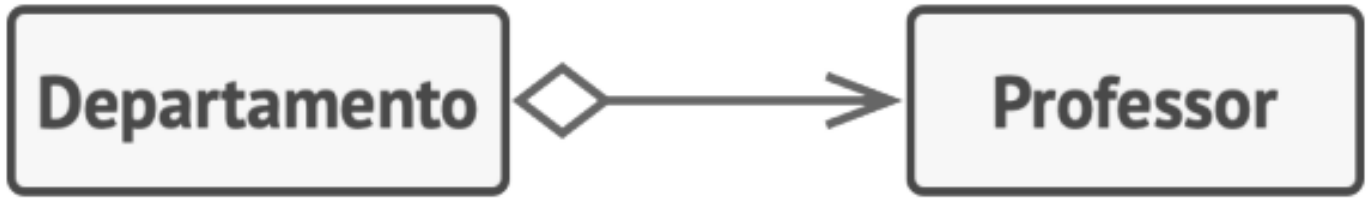
Relações entre objetos

- Em geral, usa uma **associação** para representar algo como um campo em uma classe. A ligação está sempre ali, você sempre pode fazer um pedido para o cliente dela. Mas nem sempre precisa ser um campo. Se você está modelando suas classes de uma perspectiva de interface, ele pode apenas indicar a presença de um método que irá retornar o pedido do cliente.

UML da Associação. O professor se comunica com os alunos.

Relações entre objetos

- A **agregação** é um tipo especializado de associação que representa relações individuais, múltiplas e totais entre múltiplos objetos.



UML da Agregação. O departamento contém professores.

Relações entre objetos

- A **composição** é um tipo específico de agregação, onde um objeto é composto de um ou mais instâncias de outro. A distinção entre esta relação e as outras é que o componente só pode existir como parte de um contêiner.

UML da Composição. A universidade consiste de departamentos.

Panorama geral

- **Dependência**: Classe A pode ser afetada por mudanças na classe B.
- **Associação**: Objeto A sabe sobre objeto B. Classe A depende de B.
- **Agregação**: Objeto A sabe sobre objeto B, e consiste de B. Classe A depende de B.
- **Composição**: Objeto A sabe sobre objeto B, consiste de B, e gerencia o ciclo de vida de B. Classe A depende de B.
- **Implementação**: Classe A define métodos declarados na interface B. objetos de A podem ser tratados como B. Classe A depende de B.
- **Herança**: Classe A herda a interface e implementação da classe B mas pode estendê-la. Objetos de A podem ser tratados como B. Classe A depende de B.

Exercícios

1. **No texto abaixo identifique as relações entre os objetos, na sequencia crie um diagrama UML.**

Uma Pessoa pode ser um Paciente ou um Médico. O Médico pode realizar atendimentos. Cada Paciente possui obrigatoriamente um endereco, o endereço é parte essencial de seu cadastro. Já o Médico pode estar vinculado a uma ou mais Clinicas onde trabalha, a clínica pode existir sem o médico e vice-versa. A Secretaria pode agendar Consultas entre pacientes e médicos. Durante o agendamento, a Secretaria utiliza temporariamente os dados de um Paciente e de um Médico.

Exercícios

2. **No texto abaixo identifique as relações entre os objetos, na sequencia crie um diagrama UML.**

Cada Curso possui um Professor responsável, um conteúdo programático e pode ter vários alunos matriculados. O ConteúdoProgramatico contém vários Módulos, que não podem existir fora do conteúdo. O Professor é um tipo de Pessoa, assim como o Aluno. Cada Pessoa possui um Endereço, que é parte integrante da pessoa. O sistema precisa permitir que Cursos sejam listados, e que os Alunos possam se inscrever. Há uma interface chamada UsuarioSistema, com métodos como fazerLogin() e visualizarPerfil(), que devem ser implementados por Aluno e Professor.

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional.8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.