TEXTO DE APOIO

AULA 4

Desenvolvimento de Sistemas I

Professora Cláudia Rossi

EaD | Universidade Presbiteriana Mackenzie

Sumário

DIAGRAMA DE CLASSES – HERANÇA E POLIMORFISMO

INTRODUÇÃO

Vamos, neste segmento do material de apoio, conhecer e aprender um outro tipo de associação entre classes que é a associação do tipo Generalização/Especialização e outros importantes conceitos associados a ela. Além disso, entenderemos importantes conceitos ligados a esse tipo de associação, que são:

- Herança
- Polimorfismo
- Sobrescrita
- Sobrecarga

Também estudaremos uma técnica para elaborar diagrama de classes.

Vamos começar!

HERANÇA

Uma hierarquia é a classificação ou ordenação de abstrações. Existem duas importantes hierarquias em sistemas complexos, que são:

- sua estrutura de classes → hierarquia “é uma”
- sua estrutura de objetos → hierarquia “parte de”

A **Herança** é a mais importante hierarquia “é um”, e é considerada um elemento essencial em sistemas orientados a objetos segundo Booch (2007). A Herança define um relacionamento entre classes em que uma classe compartilha estrutura e/ou comportamento definido em uma ou mais classes.

Assim, você pode perceber que a herança representa uma hierarquia de abstrações, em que uma subclasse herda de uma ou mais superclasses, ou seja, a subclasse aumenta ou redefine a estrutura e o comportamento existentes de suas superclasses.

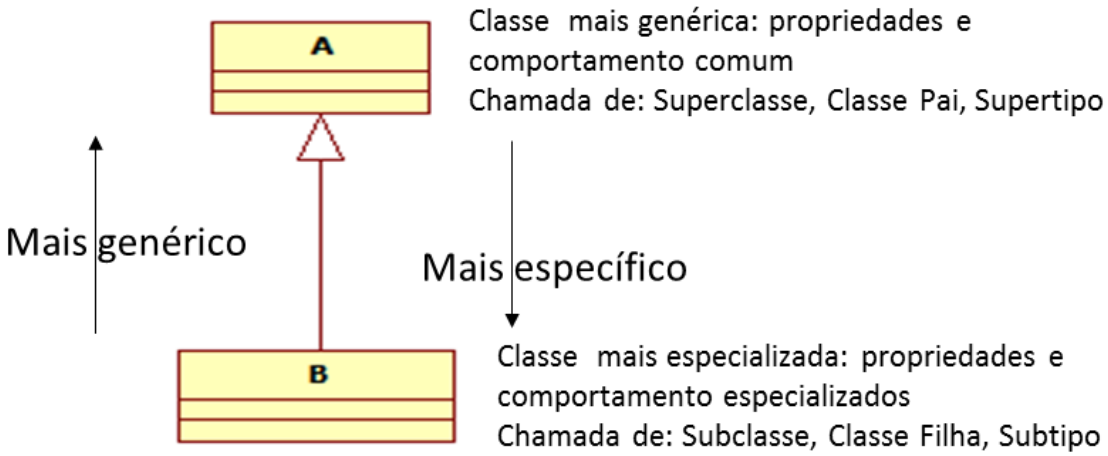
Semanticamente, herança denota um relacionamento “é um”. Por exemplo:

- um Homem “é uma” espécie de mamífero;
- um cachorro “é uma” espécie de mamífero;
- um gato “é uma” espécie de mamífero;
- uma bicicleta “é um” meio de locomoção;
- um carro “é um” meio de locomoção.

A herança implica em uma hierarquia de **generalização / especialização**, sendo que fica localizada na subclasse a especialização, ou seja, o comportamento especializado. Já o comportamento comum deve ficar localizado na superclasse, ou seja:

- Generalização é o relacionamento entre uma classe (**a superclasse**) e uma ou mais variações da classe (**as subclasses)**.
- A generalização e a especialização são dois pontos de vista do mesmo relacionamento, ou seja, dado duas classes **A** e **B**, se **A** é uma generalização de **B**, então **B** é uma especialização de **A**.

Como isso é representado em UML?

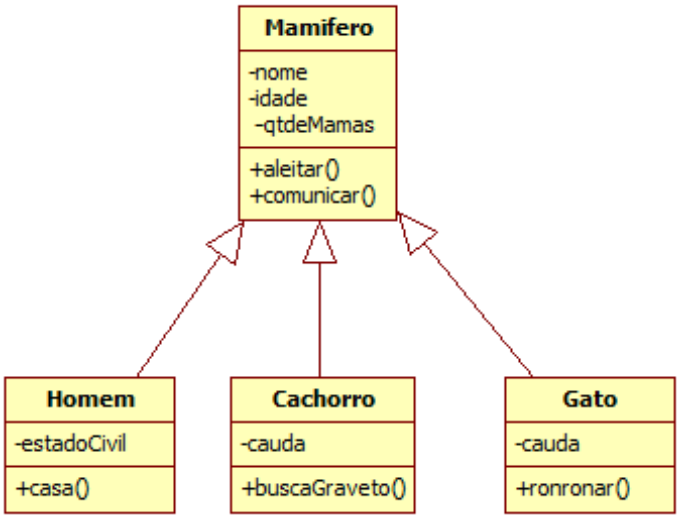


Segundo Bezerra (2015), o termo subclasse é utilizado para denotar que a classe herda as propriedades de outra classe (superclasse) por meio de uma generalização, ou ainda, que a classe que possui as propriedades herdadas por outras é a superclasse. Também podem ser usados os seguintes nomes:

- Supertipo (superclasse) e subtipo (subclasse);
- Classe base (superclasse) e classe herdeira (subclasse);
- subclasse é uma especialização da sua superclasse, e a superclasse é uma generalização de suas subclasses;
- o termo ancestral e descendente também são usados para fazer referência ao uso do relacionamento de generalização em vários níveis.

Semanticamente, a generalização/especialização denota um relacionamento “é um”.

A superclasse mantém atributos, operações e associações comuns; as subclasses acrescentam atributos, operações e associações específicas. Assim, pode-se afirmar que a subclasse herda as características de sua superclasse. Veja o exemplo dos animais mamíferos:



Nesse exemplo, você pode notar que um **Homem** "é uma" espécie de **Mamífero**, um **Cachorro** "é uma" espécie de **Mamífero**, um **Gato** "é um" tipo de **Mamífero**.

A hierarquia de generalização/ especialização, é a estrutura ou o comportamento mais especializado de uma subclasse em relação as suas superclasse. A herança é o compartilhamento de atributos e operações (recursos) entre classes com base em um relacionamento hierárquico.

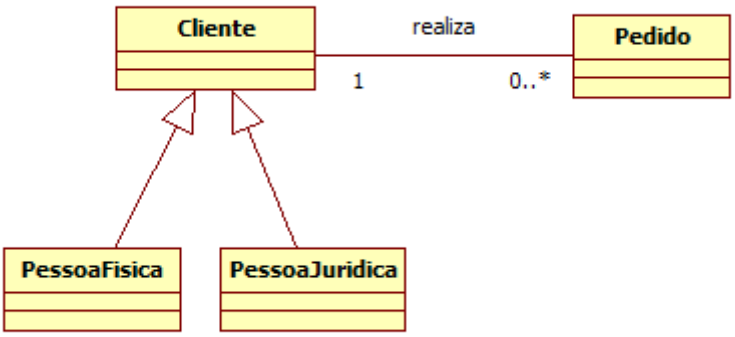
Todos os Mamíferos compartilham os atributos nome, idade e quantidade de mamas, entretanto, apenas alguns certos tipos de mamíferos que possuem estado civil e podem casar como os Homens. Os mamíferos do tipo Gato e Cachorro, por sua vez, possuem cauda; apenas os cachorros sabem buscar gravetos, e apenas os gatos sabem ronronar.

Os cachorros pertencem à classe Mamífero, mas apenas os animais que têm as características e os comportamentos de cachorros pertencem à classe Cachorro, portanto, os cachorros possuem todas as características inerentes a Mamífero, além de possuir as suas próprias características, dessa forma, a classe Cachorro herda as propriedades da classe Mamífero.

Assim, para entender a semântica de um relacionamento de herança, é preciso lembrar que uma classe representa um conjunto de objetos que compartilham um conjunto comum de propriedades (atributos, operações e associações). Entretanto, alguns objetos, embora bastante semelhantes a outros, podem possuir um conjunto de propriedades que outros não possuem.

Outro aspecto importante a ser destacado é que: não somente atributos e operações são herdados pelas subclasses, mas também as associações que estão definidas na superclasse.

Veja o exemplo a seguir:



Nesse exemplo, existe uma associação entre Cliente e Pedido, não há necessidade de criar uma associação Pedido e PessoaFisica ou entre Pedido e PessoaJuridica, pois as subclasses herdam a associação de sua superclasse.

Entretanto, se existe alguma associação que não é comum a todas as subclasses, mas sim a alguma subclasse particular, então essa associação deve ser representada em separado, envolvendo somente as subclasses em questão.

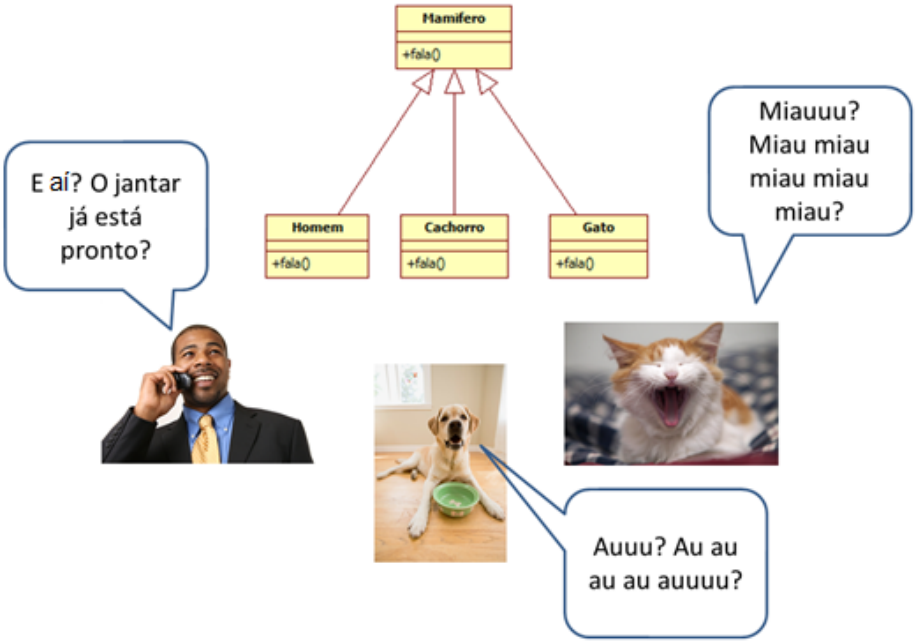
SOBRESCREVENDO CARACTERÍSTICAS

Uma subclasse pode sobrescrever uma característica da subclasse definindo uma característica com mesmo nome, ou seja, a característica que sobrescrever (a característica da subclasse) refina e substitui a característica sobrescrita (a característica da superclasse).

Motivos para você querer sobrescrever uma característica:

- Para especificar o comportamento que depende da subclasse.
- Para refinar a especificação de uma característica.
- Para melhorar o desempenho.

Observe novamente o exemplo dos mamíferos :



Nesse caso, a subclasse tem de redefinir o comportamento da operação da superclasse. Dessa forma, o comportamento de alguma operação herdada pode ser diferente, como mostra o exemplo da classe de mamíferos apresentado na aula anterior, como relação à forma como homem, o cachorro e gato implementam o método **fala()**.

Ou seja, as **operações polimórficas** são operações de mesma assinatura definidas em diversos níveis de uma hierarquia de herança e que possuem métodos diferentes, sendo que essas operações devem ter a assinatura repedida na(s) subclasse(s) para enfatizar que elas são redefinidas; dessa forma, somente as assinaturas são herdadas, mas não a implementação (BEZERRA, 2015).

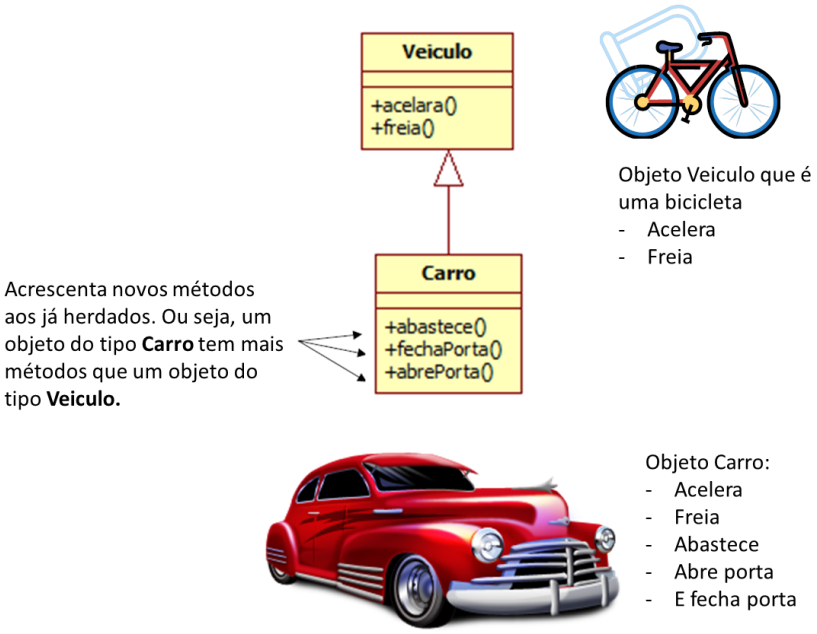
O **Polimorfismo** é o princípio da orientação a objetos em que duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura, mas comportamentos distintos, especializados para cada classe derivada, usando, para tanto, uma referência a um objeto do tipo da superclasse.

Uma subclasse herda todas as propriedades de sua superclasse que tenham **visibilidade pública (+)** ou **protegida (#).**

Pode-se observar que uma subclasse pode receber uma mensagem para que uma operação (pública ou protegida) definida na superclasse seja executada.

A subclasse pode estender e sobrescrever comportamento da superclasse.

Veja este exemplo:



Ou seja, o comportamento de alguma operação da classe herdada é diferente para a algumas subclasses.

Importante :

A decisão sobre qual método deve ser selecionado, de acordo com o tipo da classe derivada, é tomado em tempo de execução, por meio do mecanismo de ligação tardia.

Ligação Tardia: Para a utilização de polimorfismo, a linguagem de programação orientada a objetos deve suportar o conceito de **ligação tardia** *(late binding)*, em que a definição do método que será efetivamente invocado só ocorre durante a execução do programa. Também é conhecido pelos termos *dynamic binding* ou *run-time binding*.

Ligação prematura: Quando o método a ser invocado é definido durante a compilação do programa, o mecanismo de **ligação prematura** *(early binding)*.

Outro conceito importante:

» Sobrescrita dos métodos não é a mesma coisa que sobrecarga (***overloading***) de métodos e, assim, não deve ser confundida com este mecanismo.

» Sobrecarga de métodos: Métodos podem compartilhar o mesmo nome, mas possuir diferentes parâmetros, que variam em número ou tipo.

Um método aplicado a um objeto é selecionado para execução por meio de sua assinatura e da verificação a qual classe o objeto pertence.

O mecanismo de sobrecarga (**overloading**) – dois métodos de uma mesma classe podem ter o mesmo nome, desde que suas listas de parâmetros sejam diferentes, constituindo assim uma assinatura diferente.

Essa situação não gera conflito, uma vez que o compilador é capaz de detectar qual método deve ser escolhido a partir da análise dos tipos de argumentos do método.

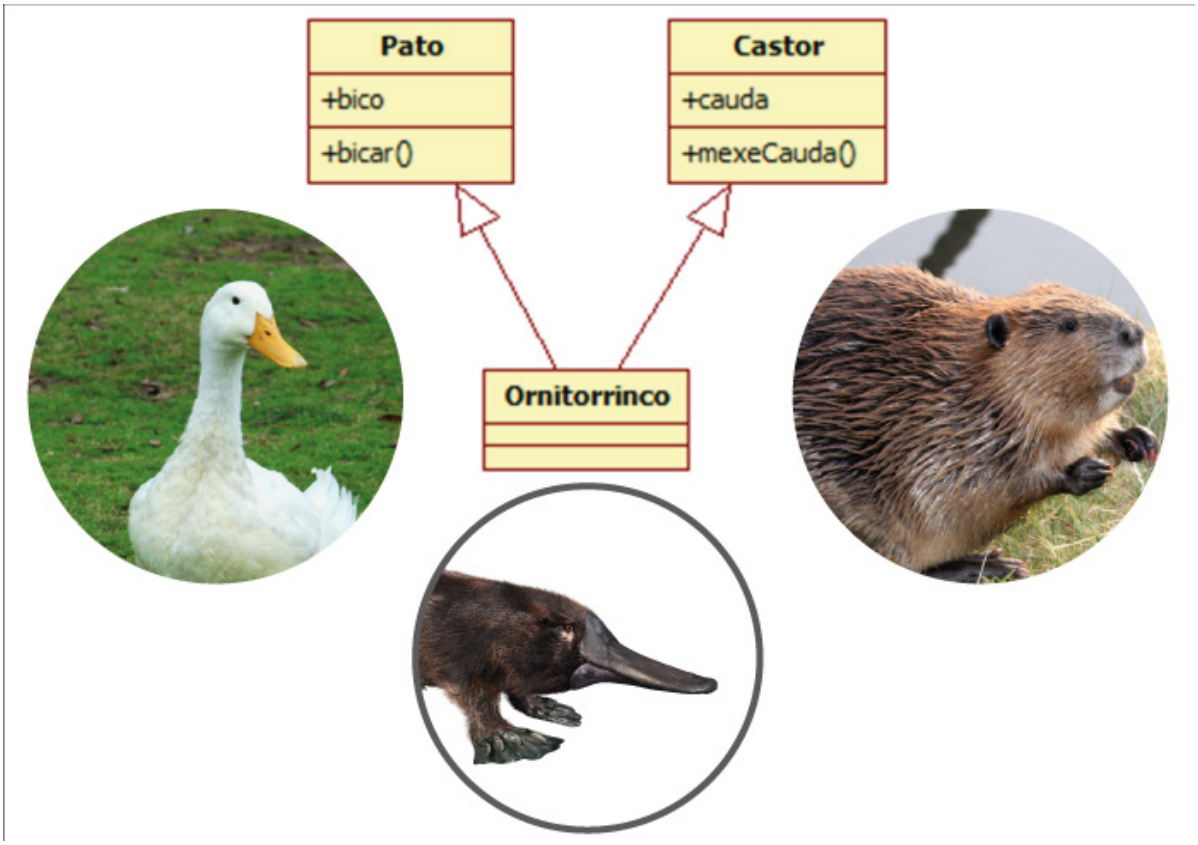
Podem ser citados como exemplo em Java os métodos abs(), max() e min() da classe Math, que têm implementações alternativas para quatro tipos de argumentos distintos.

HERANÇA MÚLTIPLA

> Existem diversas categorizações que podem ser feitas a respeito do conceito de herança, uma delas é com relação à quantidade de superclasses que uma certa classe pode ter. Assim, uma classe pode ter mais de uma superclasse, esse é o conceito de **herança múltipla**.

Dessa forma, a herança múltipla permite que uma classe possua mais de uma superclasse e herde características de todos os seus ancestrais. O que possibilita a mesclagem de informações de duas ou mais origens.

Exemplo:



A herança múltipla é uma forma mais complicada de generalização do que a herança simples, que restringe a hierarquia de classes a uma árvore.

- A vantagem da herança múltipla: maior capacidade de especificação de classes e a maior oportunidade de reutilização. A herança múltipla habilita a modelagem de objetos para mais próximo da maneira como se costuma pensar.

- A desvantagem da Herança Múltipla: perda em simplicidade conceitual e de implementação.

Nem todas as linguagens de programação orientadas a objetos implementam a herança múltipla. Algumas linguagens que implementam herança múltipla, como C++ e Phyton. As linguagens Java, C# e Smalltalk não implementam herança múltipla.

Agora que conhecemos muitos tipos de associações entre classes, que tal elaborarmos diagramas de classes?

PROCEDIMENTO PARA A CONSTRUÇÃO DO DIAGRAMA DE CLASSES

Para elaborar um diagrama de classes na fase de análise, você pode aplicar o seguinte roteiro, que foi adaptado de Blaha e Rumbaugh (2006):

1. Identificar classes a partir dos casos de uso.

    1.1. Identificar os candidatos a classes:

    - Identificar os substantivos na descrição dos casos de uso.
    - Os substantivos podem ser, geralmente, candidatos a classes.
    - Nesse momento, não é necessário se preocupar inicialmente com as estruturas de herança e agregação.

    1.2. Manter as classes adequadas:

    - Eliminar classes redundantes.
    - Eliminar classes não relevantes (por exemplo, substantivos que aparecem no texto, mas não estão no escopo do sistema).
    - Especificar melhor as classes vagas.
    - Eliminar as construções não relacionadas com o mundo real; as construções relacionadas com a implementação devem ser utilizadas na fase de projeto (por exemplo, processador, subrotina, algoritmo, interrupção).

2. Preparar o dicionário do modelo, incluindo os elementos dos modelos com descrição de duas a três linhas.
3. Identificar e nomear as associações entre as classes.
4. Identificar as operações mais significativas das classes.
5. Identificar os atributos das classes e das associações.

6. Aplicar os conceitos de generalização e especialização para organizar as classes e reduzir as redundâncias.

7. Identificar os relacionamentos de agregação.

8. Verificar se existem todas as informações necessárias para a execução das operações do sistema (que podem ser descritas nos cenários dos casos de uso).

9. Refinar o modelo.

REFERÊNCIAS

BEZERRA, E. *Princípios de análise e projeto de sistemas com UML*. 3. ed. Rio de Janeiro: Elsevier-Campus, 2015.

BLAHA, M.; RUMBAUGH, J. *Modelagem e projetos baseados em objetos com UML 2*. Rio de Janeiro: Elsevier-Campus, 2006.

BOOCH, G. *Object-oriented analysis and design with applications*. 3. ed. Massachusetts: Addison-Wesley, 2007.

FOWLER, M. *UML essencial:* um breve guia para linguagem padrão. 3. ed. Porto Alegre: Bookman, 2005.

PAGE-JONES, M. Fundamentos do desenho orientado a objeto com UML. São Paulo: Pearson, 2001.