INSTITUTO FEDERAL DE MATO GROSSO DO SUL
CAMPUS NAVIRAÍ

NOTAS DE AULA DE

# ANÁLISE E PROJETO ORIENTADO A OBJETO II

**Prof. MSc. Luiz F. Picolo**

NAVIRAÍ - MS

**Atualizado em 8 de setembro de 2020**

# 1 Introdução

*Persistência é a irmã gêmea da excelência.*
*Uma é a mãe da qualidade, a outra é a mãe do tempo.*
**Marabel Morgan**

Para que haja um entendimento melhor sobre alguns conceitos que serão utilizados no decorrer dessa disciplina, devemos, neste momento, recapitular alguns conhecimentos obtidos em semestres anteriores. Tais conhecimento se referem ao **paradigma da orientação a objetos**, ou, **POO**.

Portanto, esse capítulo introdutório, terá como objetivo introduzir/recapitular os conceitos sobre o paradigma da orientação a objeto. Contudo, apenas será revisto tais conceitos, os quais, serão utilizados em aula e não retomado todo o conteúdo referente ao tema.

## 1.1 Orientação a objeto

O Paradigma da Orientada a Objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de **objetos**. O POO é um dos 4 principais paradigmas, os outras são o imperativa, funcional e lógica). Já a programação orientada a objetos se ocupa de realizar um projeto de software utilizando uma linguagem de programação orientada a objetos, por exemplo, a linguagem Java, que aceita a implementação direta de objetos e fornece recursos para definir as classes de objetos [Sommerville 2003].

### 1.1.1 Projeto orientado e objeto

O projeto orientado a objetos é uma estratégia em que os projetistas de sistema pensam em termos de "coisas", em vez de operações ou funções. O sistema em funcionamento é constituído de objetos que interagem entre si mantendo seu próprio estado local e fornecem operações com base nessas informações de estado [Sommerville 2003].

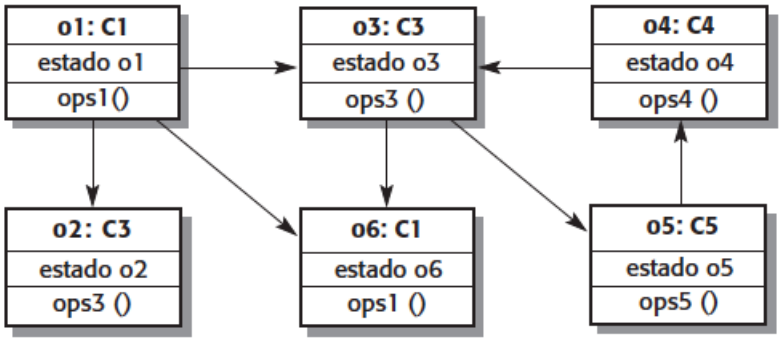



**Figura 1** – Um sistema constituído de objetos que interagem entre si.

Fonte: [Sommerville 2003]

Como afirma Sommerville 2003, projeto orientado a objetos é parte do desenvolvimento orientado a objetos, em que uma estratégia orientada a objetos é utilizada em todo o processo de desenvolvimento:

- A **análise orientada a objetos** se dedica a desenvolver um modelo orientado a objetos do domínio de aplicação. Os objetos identificados refletem entidades e operações que estão associadas com o problema a ser resolvido.
- O **projeto orientado a objetos** se dedica a desenvolver um modelo orientado a objetos de um sistema de software para implementar os requisitos identificados. Os objetos em um projeto orientado a objetos estão relacionados à solução do problema que está sendo resolvido. É possível que haj relacionamentos próximos entre alguns objetos do problema e alguns objetos da solução, mas o projetista, inevitavelmente, precisa adicionar novos objetos e transformar objetos do problema, a fim de implementar a solução.
- A **programação orientada a objetos** se ocupa de realizar um projeto de software utilizando uma linguagem de programação orientada a objetos, por exemplo, a linguagem Java, que aceita a implementação direta de objetos e fornece recursos para definir as classes de objetos.

A transição entre esses estágios de desenvolvimento deve ser contínua e direta, com a mesma notação utilizada em cada um. Mover para o próximo envolve aprimorar o anterior, adicionando detalhes às classes existentes de objetos e inventando novas a fim de oferecer funcionalidades adicionais.

## 1.2 Classes e objeto

Para [Sommerville 2003] "objeto" e "orientado a objetos" são amplamente utilizados e aplicados a diferentes tipos de entidades, métodos de projeto, sistemas e linguagens de programação. Contudo, existe uma aceitação geral de que um objeto é um encapsulamento de informações, e isso se reflete na definição de um objeto e de uma classe de objeto a seguir:

- Um objeto é uma entidade que possui um estado e um conjunto definido de operações que operam nesse estado. O estado é representado por um conjunto de atributos de objeto. As operações associadas com o objeto fornecem serviços para outros objetos (clientes), que requisitam esses serviços quando alguma computação é necessária.
- Os objetos são criados de acordo com uma definição de classe de objetos que serve como um template para criar objetos. Essa classe apresenta declarações de todos os atributos e operações que devem ser associados a um objeto dessa classe.

Em outras palavras, podemos dizer que classe é uma descrição generalizada que descreve uma coleção de objetos similares, o qual, segundo Pressman e Maxim 2016, é um conceito orientado a objeto que encapsula dados e abstrações procedurais necessárias para descrever o conteúdo e comportamento de alguma entidade do mundo real.

Exemplos de objetos são: os **objetos físicos** (um livro, uma caneta), **funções de pessoas** para os sistemas (funcionário, cliente), eventos (uma compra, um telefonema), **interações** entre outros objetos (um item de uma nota fiscal é uma interação entre uma compra e um produto do estoque) e **lugares** (loja matriz, revenda nordeste).

Para fins de estudo, e com objetivo mais didático, usaremos um cachorro como nosso "objeto":

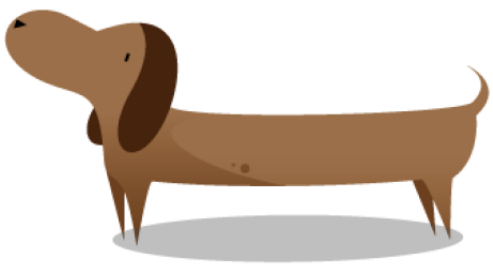


**Figura 2** – Representação de um objeto

Ao analisar o objeto, deduz-se que há características pertencentes somente a ele. Tais como:

- Nome;

- Idade;
- Comprimento de pelos;
- Cor dos pelos;
- Cor dos olhos;
- Peso, entre outros;

Tais características que descrevem um objeto são chamadas na orientação a objeto de atributos.

## 1.3 Atributos

Os objetos do mundo real tem propriedades que por sua vez possuem valores. Estes valores determinam o estado do objeto. Assim, na orientação a objeto, essas propriedades são chamadas de atributos. Logo, podemos dizer que esses atributos são semelhantes a variáveis ou campos que guardam os variados valores que os objetos podem receber como características.

**O estado de um objeto é um grupo de valores que estão em seus atributos em um certo momento**.

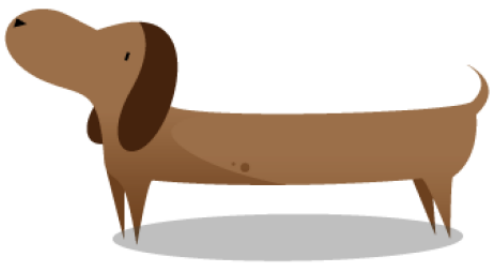


**Figura 3** – Representação de um objeto

| Cachorro | |
|---|---|
| Nome: | Hubert |
| Idade: | 2 anos |
| Tipo Pelo: | Curtos |
| Cor dos pelos: | Marrom |
| Cor dos olhos: | Castanhos |
| Peso | 5kg |

**Tabela 1** – Atributos e valores

Outro objeto cachorro teria valores diferentes para estes mesmos atributos, como exemplo disto temos:

**Figura 4** – Representação de um objeto

| Cachorro | |
|---|---|
| Nome: | Floks |
| Idade: | 4 anos |
| Tipo pelo: | Curtos |
| Cor dos pelos: | Branco |
| Cor dos olhos: | Castanhos |
| Peso | 5kg |

**Tabela 2** – Atributos e valores

Para que os atributos de um objeto mudem de valor isso deve ser feito exclusivamente por estímulos externos ou internos. Assim, a única maneira de alterar os atributos dos objetos é disparando eventos que geram a mudança desses estados no objeto.

## 1.4 Métodos

São procedimentos ou funções que executam as ações específicas do objeto. Dessa forma, os métodos são as ações que o objeto é capaz de realizar. É através dos métodos que o objetos faz tudo, inclusive se manifesta e interage com outros objetos.

Um objetos expõe algum comportamento (ou seja, executa uma operação) ao receber um estímulo de outro objeto. É através de mensagens que um objeto requisita uma ação a outro objeto. Sendo esta mensagem uma solicitação a um objeto para que sejam executadas as rotinas a qual são nomeadas de Método da classe. Então os métodos assumem a responsabilidade de acessar ou alterar os atributos de um objeto.

Anteriormente no estudo do objeto cachorro foi enumerado uma série de métodos (ações), tais como: latir, babar, comer, sentar, etc.

## 1.5 Herança

O conceito de herança é um dos conceitos fundamentais de POO. Herança, na prática, significa a possibilidade de construir objetos especializados que herdam as características de objetos mais generalistas, ou ainda, a herança uma maneira de reutilizar código a medida que podemos aproveitar os atributos e métodos de classes já existentes para gerar novas classes mais específicas que aproveitarão os recursos da classe hierarquicamente superior [Ruiz 2008].

O conceito de herança mimetiza as características hierárquicas de vários sistemas reais, como por exemplo, os sistemas de classificação em biologia que, pode determinar como uma hierarquia o seguinte:

- animais;
- vertebrados e invertebrados;
- mamíferos e aves;
- entre outras características mais específicas

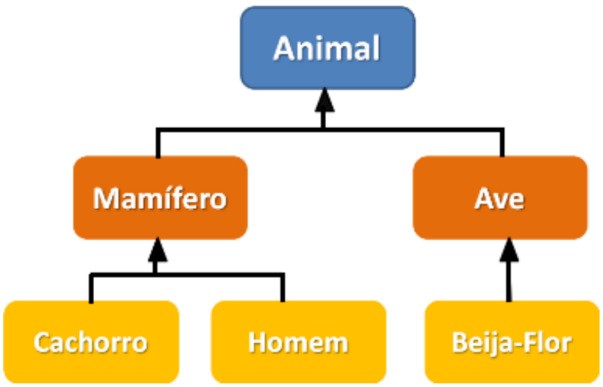



**Figura 5** – Diagrama representando a herança entre as classes

## 1.6 Classes abstratas

Podemos dizer que as classes abstratas servem como modelo para outras classes que dela herdem (veremos herança nas próximas sessões), não podendo ser instanciada por si só. Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe. Os métodos da classe abstrata devem então serem sobrescritos nas classes filhas.

### 1.6.1 Superclasses e subclasses

Em POO todo objeto de uma classe construída pelo usuário da linguagem é também um objeto de outra classe.Por exemplo, na hierarquia da área da saúde, podemos dizer que pessoa é uma superclasse e que empregados é uma subclasse de pessoa.

Outra nomenclatura que é utilizada para especificar superclasses ou subclasses é a Generalização ou Especialização. No exemplo acima, pessoa é a generalização de empregado, e empregado é a especialização de pessoa conforme representado na Figura 6

Uma máxima que podemos guardar é:

***Uma subclasse guarda a relação é um com a superclasse.***


**Figura 6** – Superclassses e Subclasses

Fonte: [Silva et al. 2009]

## 1.7 Exercícios de fixação

1. Para satisfazer as necessidades de informatização de uma biblioteca universitária um sistemas foi proposto para satisfazer algumas características:

   - Cadastro dos usuários da biblioteca com endereço completo. Usuário são classificados em três grupos: professores, alunos e funcionário.
   - Cadastro das obras da biblioteca são classificados em: livros científicos, periódicos científicos, periódicos informativos, periódicos diversos, entretenimento, etc.
   - Linguagem usada no exemplar da obra.
   - Mídia que armazena o exemplar da obra.
   - Autores da obra com o controle da nacionalidade dos mesmos.
   - Editoras dos exemplares com ano de edição referente a cada exemplar.

   Identifique os possíveis objetos com seus atributos e métodos respectivos.

2. **Desafio:** Pesquise sobre os pontos negativos da orientação a objeto, principalmente sobre os conceitos de Coesão e Acoplamento.

## 1.8 Mensagem

Mensagens são requisições enviadas de um objeto para outro, para que o objeto "receptor" forneça algum resultado desejado por meio da execução de uma operação. As

trocas de mensagem funcionam como uma fábrica que recebe uma ordem de produção (mensagem de solicitação), processa essa ordem (operações) utilizando matéria-prima (atributos) e gera um produto final (mensagem de resposta).

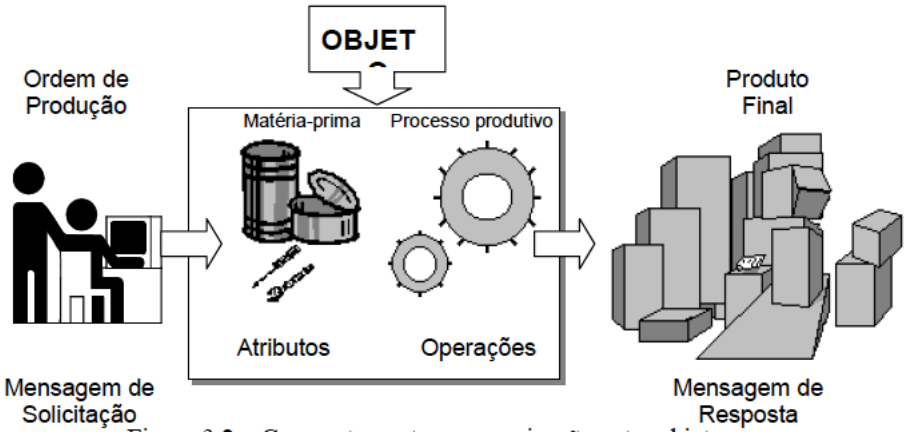



**Figura 7** – Comportamento e comunicação entre objetos

## 1.9 Encapsulamento

Cada objeto é visto como o encapsulamento do seu estado interno, suas mensagens e seus métodos. A estrutura do estado e dos pares "mensagem - método"são todas definidas através da classe à qual o objeto pertence.

- A encapsulação de dados com o código que os manipula em classes é a principal vantagem da Orientação a Objeto
- No sentido de não quebrar a encapsulação, é muito importante que os membros de uma classe (atributos e métodos) sejam visíveis apenas onde estritamente necessário (A lei é: "Não posso quebrar o que não posso acessar").

### 1.9.1 Especificadores de controle de acesso

Juntamente com a ideia de encapsulamento nós também possuimos os especificadores de controle de acesso ou visibilidades. A visibilidade é a maneira com a qual o desenvolvedor proibe ou permite acesso a determinados métodos ou atributos de uma classe, como pode ser visto na Figura 22. Neste sentido, o objeto formado possuirá as mesmas definições declaradas pela classe.

#### 1.9.1.1 A visibilidade public

- Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public;

- O alvo aqui é o programador cliente que usa suas classes;
- É raro ter atributos públicos mas é comum ter métodos públicos.

#### 1.9.1.2 A visibilidade private

- O membro private não é acessível fora da classe;
- A intenção aqui é permitir que apenas você que escreve a classe possa usar esse membro.

#### 1.9.1.3 A visibilidade protected

- O membro protected é acessível à classe e a suas subclasses;
- A intenção é dar acesso ao programadores que estenderão sua classe.

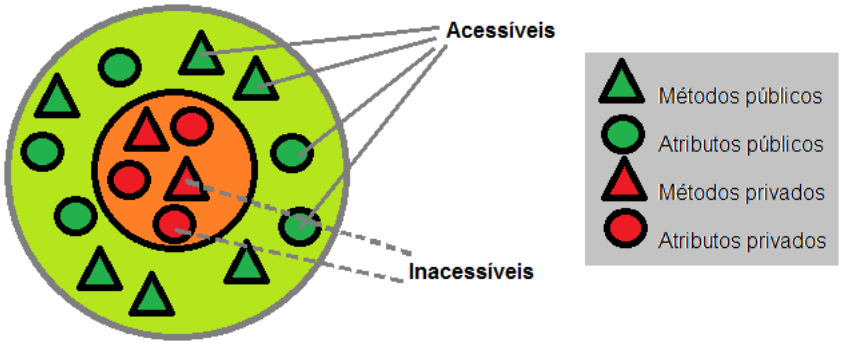



**Figura 8** – Encapsulamento

## 1.10 Polimorfismo

É a propriedade que permite que a mesma mensagem seja enviada a diferentes objetos e que cada objeto execute a operação que é apropriada à sua classe. No caso de polimorfismo, é necessário que os métodos tenham exatamente a mesma identificação, sendo utilizado o mecanismo de redefinição de métodos.

No exemplo utilizado na Figura 9, podemos perceber que diferentes objetos, quando é solicitado a mesmo ação, se comportam de maneira diferente. Similiar a objetos do mundo real, na Orientação a Objetos, o comportamento por meio do polimorfismo acontece da mesma forma.

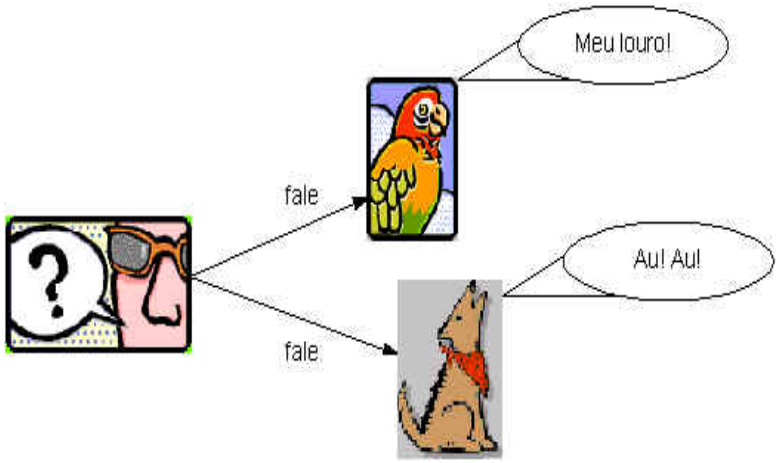



**Figura 9** – Polimorfismo

## 1.11 Conceitos estudados até este momento

Para ilustrar, a tabela abaixo contém um resumo e um exemplo de todos estes conceitos.

| Palavra-Chave | Breve Definição | Exemplo |
|---|---|---|
| Classe | Agrupamento de objetos similares que apresentam os mesmos atributos e operações | Indivíduo, caracterizando as pessoas do mundo |
| Atributo | Característica particular de uma ocorrência da classe | Indivíduo possui nome, sexo, data de nascimento |
| Operações | Lógica contida em uma classe para designar-lhe um comportamento | Cálculo da idade de uma pessoa em uma classe (Indivíduo) |
| Encapsulamento | Combinação de atributos e operações de uma classe | Atributo: data de nascimento<br>Operação: cálculo da idade |
| Herança | Compartilhamento pela subclasse dos atributos e operações da classe pai | Subclasse (Eucalipto) compartilha atributos e operações da classe (Árvore) |
| Subclasse | Característica particular de uma classe | Classe (Árvore),Subclasses (Ipê, Eucalipto, Jacarandá, etc.) |
| Instância de Classe | Uma ocorrência específica de uma classe. É o mesmo que objeto | Uma pessoa, uma organização ou um equipamento |
| Objeto | Elemento do mundo real (natureza). Sinônimo de instância de classe | Pessoa “Fulano de Tal”, Organização “ACM”, Equipamento “Extintor” |
| Mensagem | Uma solicitação entre objetos para invocar certa operação | Informar idade da pessoa “Fulano de Tal” |
| Polimorfismo | Habilidade para usar a mesma mensagem para invocar comportamentos diferentes do objeto | Chamada da operação: “Calcular Saldo” de correntista. Invoca as derivações correspondentes para cálculo de saldo de poupança, renda fixa, etc. |

## 1.12 Exercícios de fixação

Baseando-se nas explicações em aula e em suas anotações, responda as seguintes questões:

1. O que é a Programação Orientada a Objetos e qual a sua relação com o Projeto orientado e objeto. Aponte as diferenças e similitudes em ambos os conceitos.

2. Um ponto importante que deve ser claro para que possamos atingir o objetivo de nossa disciplina é o conceito sobre classes e objetos. Explique o que são e de exemplos do seu cotidiano para reforçar a ideia.

3. O que são atributos e métodos? Defina cada um e de exemplos para afirmar sua resposta. Obs: Caso possível, use o exemplo anterior.

4. Um dos pilares da orientação a objeto é a Herança. Desenho um gráfico, semelhante ao da Figura 5, representado a sua Árvore Genealógica. Caso consiga, vá té seus bisavós.

5. Utilizando o exercício anterior. Represente, por meio de setas, as possíveis trocas de mensagem entre os objetos que você pode identificar.

# 2 Diagrama de classes

*Dedique-se aos estudos para que eles o façam*
*melhor para a sociedade e para si mesmo.*
**Alvaro Granha Loregian**

O modelo de caso de uso é formado por diversos elementos que tem como propósito o fornecimento de uma visão do ponto de vista externo ao sistema. Essa funcionalidade, externamente observada, é fornecida por meio da colaboração entre diversos elementos, os quais são chamados de **objetos**. A colaboração entre os objetos podem ser vista em dois aspectos, os **dinâmicos** e **os estruturais estáticos**.

O aspecto dinâmico diz respeito a forma com que os objetos realizam as trocas de mensagens e as reações a eventos que ocorrem no sistemas. Já os estruturais estáticos permite visualizar como o sistema está estruturado internamente para que as funcionalidades visíveis sem produzidas [Bezerra 2016].

Assim, realizaremos uma introdução aos aspectos estruturais estáticos, demostrando os elementos que o diagrama de classes utiliza para a representar um sistema orientado a objeto. O diagrama de classe, segundo Guedes 2018, é um dos mais importante e mais utilizados diagramas da UML, o qual permite a visualização da **classes** que comporão o sistemas com seus respectivos **atributos**, **métodos** e pelo relacionamento entre estes.

## 2.1 Modelo de classes

A medida que o sistemas é desenvolvido, o modelo de classe pode evoluir para demostrar o sistema com maior riqueza de detalhes. Assim, podemos estudar três níveis de detalhamento: **domínio**, **especificação** é **implementação** [Bezerra 2016].

- **Modelo de domínio:** Nesse modelo não é levado em consideração como o sistemas será implementado mas apenas as classes que o comporão.
- **Modelo de especificação:** Já no modelo de especificação são adicionados detalhes específicos relacionados a solução do software.
- **Modelo de implementação:** Esse modelo corresponde a implementação das classes em alguma linguagem de programação.

## 2.2 Diagrama de classes

O diagrama de classes, como dito anteriormente, é um dos mais importantes para o desenvolvimento de um sistema orientado a objeto.

Como todos os diagramas da UML, o diagrama de classe é composto de elementos que, consequentemente, possuem também suas responsabilidades. Assim, veremos todos este elementos, essenciais para o modelo de domíno, e como eles interagem entre si.

Lembre-se que, o conceito relacionado a cada um já foi apresentado no Capítulo 3.

### 2.2.1 Classes, atributos e métodos

Uma classe é representada por meio de uma **caixa** com, no máximo, três compartimentos. No compartimento superior é exibido o nome da classe, o qual é o único elemento obrigatório.

No segundo compartimento são declarados os atributos. Este, como visto no 3, armazenam os dados que um objeto armazena. Por fim, no último compartimento são declarados os métodos.

Assim, segundo Bezerra 2016, as possíveis anotações para que possamos representar uma classe utilizando as notações UML são representadas na Figura 10.

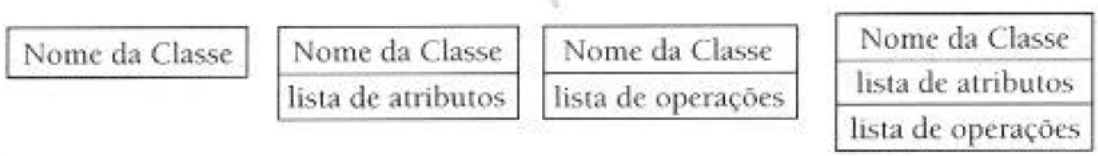



**Figura 10** – Possíveis notações para uma classe em UML.

Fonte: [Bezerra 2016]

Por convenção o nome das classe sempre será no singular e iniciará com letra maiúscula usando sempre o a forma de escrita conhecida como *CamenCase*. Já os atributos e métodos sempre iniciarão com letra minúscula, mas também seguirão o padrão *CamenCase*. Os métodos devem expressar uma ação que o objeto da classe sabe realizar. Vejamos um exemplo da classe ***ContaBancaria***.


**Figura 11** – Diferentes graus de detalhe para *ContaBancaria*.

Fonte: [Bezerra 2016]

## 2.3 Exercícios de fixação

Utilizando às notações aprendidas, identifique as possíveis classes, atributos e métodos para o problema abaixo.

1. O professor **Picolo** precisa de um software para lhe ajudar a lembrar os aniversários de seus alunos. O software deve ser capaz de armazenar o nome do aluno, sua data de nascimento, seu e-mail, seu sexo e qualquer outro dado relevante. O software deve permitir o cadastro dos alunos, avisar quem faz aniversário no dia, calcular e mostrar a idade do aluno, exibir um relatório de aniversariantes no mês e outras funcionalidades que se julgar necessário.

2. Faça o mesmo para o exercício 1 do exercício de fixação 1.7

### 2.3.1 Relacionamentos

As classes costumam ter relacionamentos entre si, com o objetivo de compartilhar informações e colaborar uma com as outras para permitir a execução dos diversos processos executados em um sistema [Guedes 2018]. Assim, vermos as diversas formas de relacionamentos presentes na UML e relacionadas ao diagrama de classes.

#### 2.3.1.1 Associação

Uma associação descreve um vínculo que ocorre normalmente entre duas classes. Esse ligação da-se o nome de **associação binária**. Contudo, há casos que a classe pode se associar a si mesmo, dando origem a um tipo de associação **chamada de unária**, ou associar-se a outras classes, a qual é chamada de **associação N-ária**. As associa-

ções representam a maneira como as classes estão unidas e interagem para realizar o compartilhamento de informações.

Muito similar ao diagrama de Casos de Uso, as associações em um diagrama de classes é representada por um linha sólida ligando as classes envolvidas, podendo também conter setas que representam a **navegabilidade**, ou seja, o sentido que as informações são transmitidas.

As associações também podem conter **títulos** que, juntamente com a navegabilidade, ajudam a compreender melhor o fluxo das informações e a forma com que ela será transmitida de um objeto para outro.

#### 2.3.1.2 Associação unária

Este tipo de associação acontece quando há o relacionamento de uma classe consigo mesma.

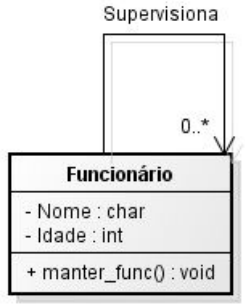



**Figura 12** – Associação unária

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

Percebam que existe uma associação e próximo ao final da seta temos **0..***. Esta informação representa a multiplicidade. A **multiplicidade** é extremamente semelhante a **cadinalidade** utilizada no Modelo Entidade-Relacionadal. Ela procura determinar qual das classes envolvidas em uma associação fornece informações para as outras, além de especificar o nível de dependências.

| | |
|---|---|
| 0..1 | Zero ou um. |
| 1..1 | Um e somente um. |
| 0..* | Zero ou muitos. |
| * | Muitos. |
| 1..* | No mínimo um ou muitos. |
| 3..5 | Mínimo de três e máximo de cinco. |

**Figura 13** – Multiplicidades

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

#### 2.3.1.3 Associação binária

Uma associação binária ocorre quando são identificados relacionamentos entre duas classes. Este tipo de associação é o mais comum no diagrama de classe.

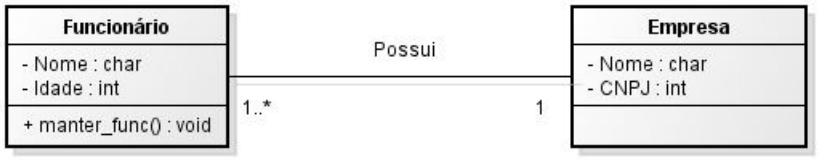



**Figura 14** – Associação Binária

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

#### 2.3.1.4 Associação N-ária

Já, uma associação N-ária, também conhecida como ternária, são aquelas mais de duas classes. Neste, além da linha associando as classes, também possuímos um losango para onde converge todas as ligações.

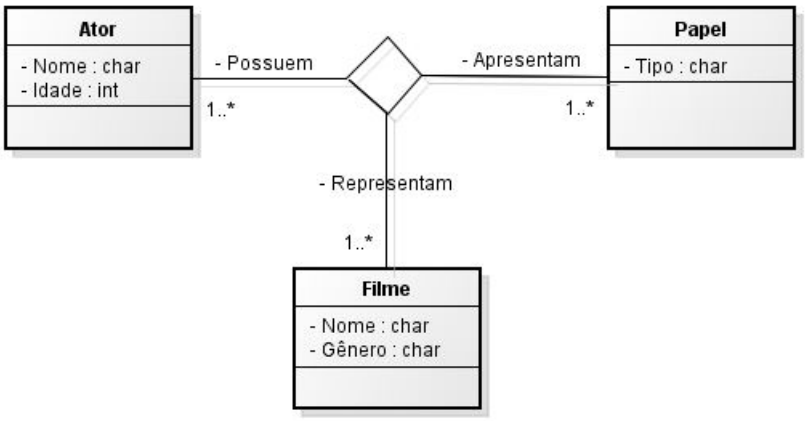



**Figura 15** – Associação N-ária

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

#### 2.3.1.5 Agregação

É um tipo especial de associação em que tenta-se demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte); conhecemos como todo/parte.

Esse tipo de associação é representada por um losango sem preenchimento no final da associação, sempre do lado do objeto **todo**.

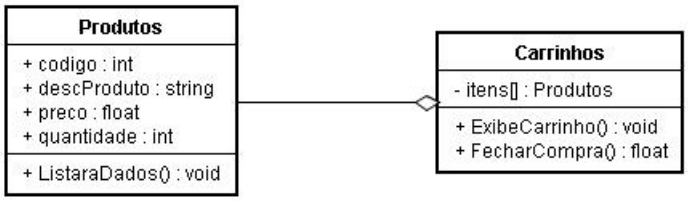



**Figura 16** – Agregação

Fonte: http://www.cpscetec.com.br/adistancia/aula5.html

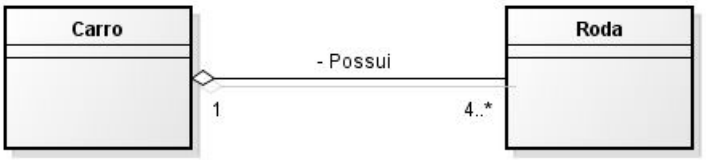



**Figura 17** – Agregação

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

#### 2.3.1.6 Composição

Pode-se dizer que composição é uma variação da agregação. Uma composição tenta representar também uma relação todo - parte. No entanto, na composição o objeto-pai (todo) é responsável por criar e destruir suas partes. Em uma composição um mesmo objeto-parte não pode se associar a mais de um objeto-pai.

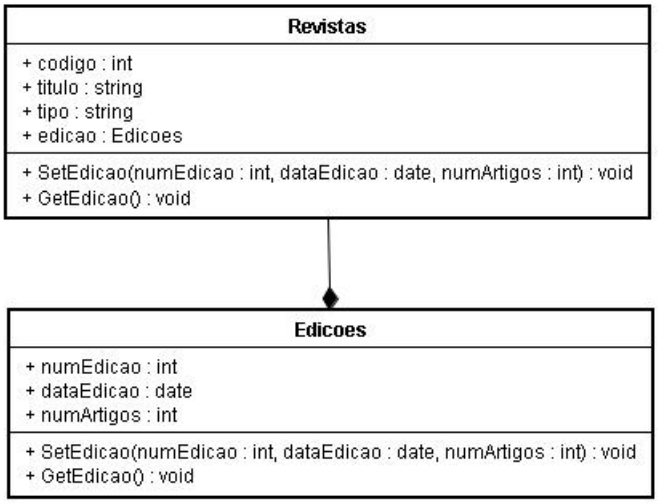



**Figura 18** – Composição

Fonte: http://www.cpscetec.com.br/adistancia/aula5.html

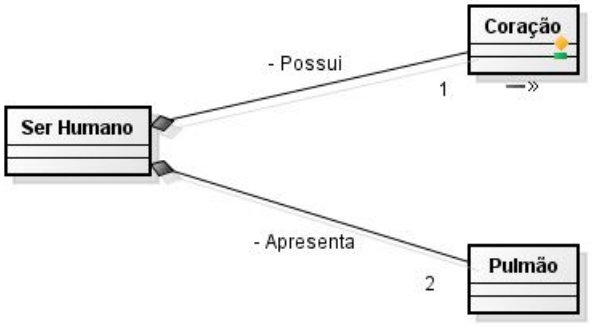



**Figura 19** – Composição

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

#### 2.3.1.7 Especialização/Generalização

Especialização/Generalização ou também chamada de Herança. Este tipo de associação ocorre quando uma classe gera novas classes que sejam suas cópias perfeitas e a partir destas classes é possível adaptá-las ao meio em que sejam necessárias. A classe que é usada como referência é conhecida por **superclasse / generalização** ou classe mãe, a classe criada com base nessa superclasse é conhecida como **sub-classe / especialização** ou classe filha. Na classe filha, podemos redefinir métodos e criar novos campos.

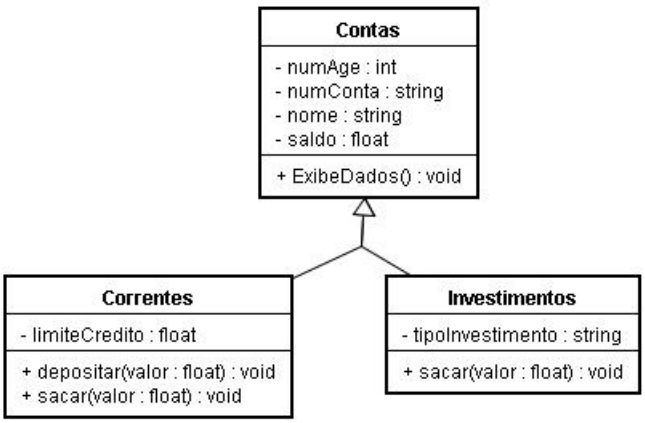



**Figura 20** – Especialização/Generalização

Fonte: http://www.cpscetec.com.br/adistancia/aula5.html

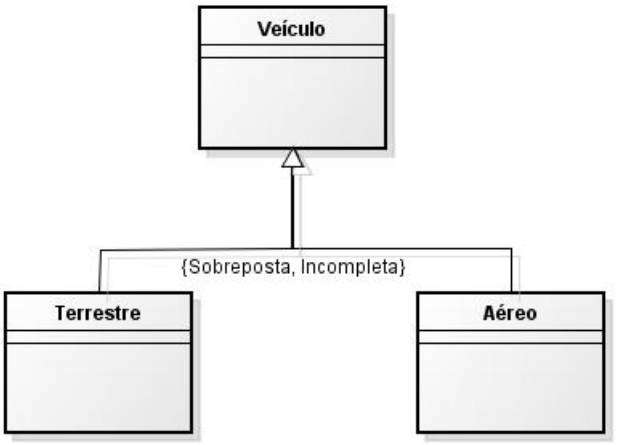



**Figura 21** – Especialização/Generalização

Fonte: http://sgvclin.altervista.org/rea-uml/pop/pop-3/popup-texto3.html

### 2.3.2 Encapsulamento

Conceitua-se encapsulamento como sendo o processo utilizado para proteger os campos e operações de uma classe (atributos e métodos), permitindo que apenas os membros públicos sejam acessados pelos usuários de determinada classe.

Para atingir o encapsulamento, uma das formas é definindo a visibilidade das propriedades e dos métodos de um objeto. A visibilidade define a forma como essas propriedades devem ser acessadas.

| | |
|---|---|
| **Público - public** | Nível de acesso livre. Indica que o método ou atributo da classe é público, ou seja, pode-se acessar este atributo, ou executar esse método, por qualquer código de programa. Usamos o sinal (+) na notação UML para simbolizar essa visibilidade. |
| **Privado - private** | Nível mais protegido. Mebros declarados como private, só podem ser acessados dentro da classe em que foram declarados. Usamos o sinal (-) na notação UML para simbolizar essa visibilidade. |
| **Protegido - protected** | Nível de acesso intermediário. Serve para que o método ou o atributo seja público dentro do código da própria classe e de qualquer classe que herde daquela onde está o método ou propriedade protected. É privado e não acessível de qualquer outra parte. Usamos o sinal (#) na notação UML para simbolizar essa visibilidade. |

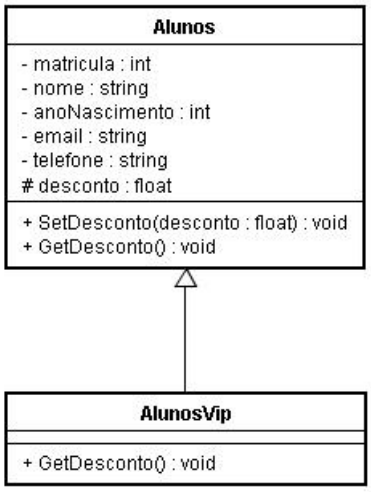



**Figura 22** – Encapsulamento

Fonte: http://www.cpscetec.com.br/adistancia/aula5.html

### 2.3.3 Exercícios de fixação

1. Uma empresa possui fornecedores de peças. Aos fornecedores é atribuído um código que os identifica univocamente. O mesmo procedimento é usado para as peças. Os fornecedores possuem ainda um nome, endereço e telefone. As peças são caracterizadas por designação e cor. As cores podem ser preto, branco, cinzento e vermelho. Os fornecimentos são feitos a um dado preço e numa dada data.

2. Cada escola da comunidade Alfa é dividida em um ou mais departamentos (letras, matemática, etc.). Um departamento é chefiado por um de seus professores, mas há casos em que esse cargo está vago. Não há acúmulo de chefia. Professores podem estar alocados em um ou mais departamentos. Um departamento pode ser criado sem que haja professores a ele alocados. Um aluno pode estar matriculado em mais de uma escola e pode frequentar mais de um curso na mesma escola. Escolas podem não ter alunos matriculados. Cada departamento tem seu conjunto específico de cursos (pelo menos um). Cada curso pode ser ministrado por um ou mais professores. Cada professor pode ministrar qualquer número de cursos.

3. Para cada obra armazena-se o ISBN, o(s) autor(es), o título, a editora, o assunto e a edição. Os exemplares possuem a data de inclusão no acervo e uma marcação de disponibilidade (as bibliotecas só cadastram obras que constam do acervo). As bibliotecas emprestam livros aos alunos e aos membros da comunidade. Alunos podem retirar até 2 títulos; membros da comunidade podem retirar apenas um título por vez.

4. Uma empresa efetua encomendas, numeradas. Cada encomenda possui um número variável de itens, cada um das quais diz respeito a um produto a ser encomendado. Os itens possuem uma ordem numérica dentro de cada encomenda. A encomenda tem uma data e dirige-se a um só fornecedor. Tanto os produtos como os fornecedores são identificados por um código próprio. Os itens incluem o número dom item, o código do produto, o preço unitário e a quantidade encomendada.

# 3 Mapeamento de objetos para o Modelo Relacional

*Um ladrão rouba um tesouro, mas não furta a inteligência.*
*Uma crise destrói um herança, mas não uma profissão.*
*Não importa se você não tem dinheiro, você é uma pessoa rica,*
*pois possui o maior de todos os capitais: a sua inteligência.*
*Invista nela.* ***Estude****!.*
**Augusto Cury**

A tecnologia de orientação a objetos é consolidada como a forma mais usual para o desenvolvimento de softwares. Já, quando pensamos em banco de dados, os bancos relacionais forma os que tiveram maior exito e, sem dúvida, os Sistemas Gerenciadores de Banco de dados Relacionais (SGBDR) dominam o mercado.

No entanto, essas duas tecnologias surgiram por meio de princípios teóricos muito diferentes. Segundo Bezerra 2016 o descasamento de informações *(impedance mismatch)* é um termo utilizado para denotar o problema das diferenças entre as representações do modelo de objetos e do modelo relacional.

"Os princípios básicos do paradigma da orientação a objetos e do modelo relacional são bastante diferentes. No modelo de objetos, os elementos (objetos) correspondem a abstrações de comportamento. No modelo relacional, os elementos correspondem a dados no formato tabular [Bezerra 2016]". Assim, neste capítulo, vamos apresentamos alguns detalhes a respeito dos diversos aspectos relativos ao mapeamento de objetos para um mecanismo de armazenamento persistente.

## 3.1 Projeto de banco de dados

Uma das principais atividades do projeto é o desenvolvimento do banco de dados. Essa atividade, durante o processo, normalmente é chamada de projeto de banco de dados. Tal projeto envolve diversas atividades, tais como:

- Construção do esquema do banco de dados.
- Criação de índices2 para agilizar o acesso aos dados armazenados.
- Definição das estruturas de dados a serem utilizadas no armazenamento físico dos dados.

- Definição de visões sobre as dados armazenados.
- Atribuição de direitos de acesso (que usuários podem acessar que recursos).
- Definição de políticas de backup dos dados.

### 3.1.1 Conceitos básicos do modelo de dados relacionais

O modelo relacional, segundo Bezerra 2016, se fundamenta no conceito de relação. De forma bastante simplista, pode-se pensar em uma relação como uma tabela, composta de linhas e de colunas. Cada relação possui um nome. Cada coluna de uma relação possui um nome e um domínio.

**Departamento**

| id | sigla | nome | idGerente |
|---|---|---|---|
| 13 | RH | Recursos Humanos | 5 |
| 14 | INF | Informática | 2 |
| 15 | RF | Recursos Financeiros | 6 |

**Figura 23** – Exemplo de relações em um banco de dados

Fonte: [Bezerra 2016]

#### 3.1.1.1 Chave Primária

Um conceito importante no modelo relacional é o de **chave primária**. Uma chave primária é uma coluna ou conjunto de colunas cujos valores podem ser utilizados para identificar unicamente cada linha de uma relação [Bezerra 2016]. Note que a Figura 23 possue uma coluna chamada id. Essa coluna é a chave primária ds relação na qual aparece.

#### 3.1.1.2 Chave Estrangéira

Outro conceito importante do modelo relacional é o de **chave estrangeira**. Linhas de uma relação podem estar associadas às de outras relações. Estas associações são representadas pela única maneira possível, considerando-se os recursos de notação do modelo relacional: deve existir, em uma das duas relações, uma coluna cujos valores fazem referência aos de uma coluna da outra relação. Na terminologia do modelo relacional, esta coluna de referência é denominada chave estrangeira. Em nossa Figura 23 a chave estrangeira é representada pelo campo idGerente, o qual apresenta-se tracejado.

## 3.2 Mapeamento de objetos para o modelo relacional

Quando utilizamos um SGBD para gerenciar nossa base de dados e nosso armazenamento persistente de informações para um sistema de software orientado a objetos, há a necessidade de se realizar o mapeamento dos valores de atributos de objetos persistentes do sistema para tabelas.

Assim, em nossa próxima sessão vamos aprender o procedimento de mapeamento de diversos elementos do modelo de classes para o modelo relacional.

### 3.2.1 Classe e seus atributos

Segundo Bezerra 2016, classes são mapeadas para relações. O caso mais simples é o de mapear cada classe como uma relação e cada um de seus atributos como uma coluna da relação correspondente. No entanto, muito frequentemente não há uma correspondência unívoca entre classes e relações. Pode ser que várias classes sejam mapeadas para uma única relação ou que uma classe seja mapeada para várias relações.

Para atributos o que vale de forma geral é que um atributo será mapeado para uma ou mais colunas. Lembre-se, também, de que nem todos os atributos são persistentes. Por exemplo, pode ser que uma classe Pedido tenha um atributo derivado, total, utilizado para guardar o valor total a ser pago por um pedido, mas que este atributo não seja armazenado no banco de dados.

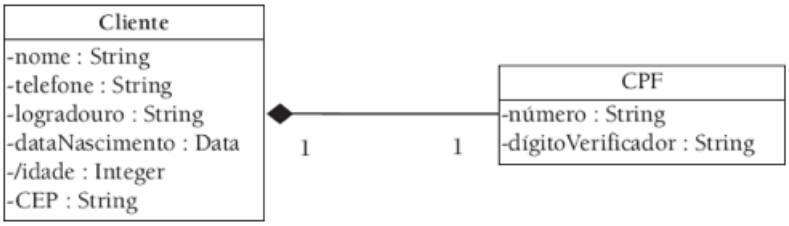



**Figura 24** – Classe cliente é CPF

Fonte: [Bezerra 2016]

Considere a classe apresentada na Figura 24. A Figura 25 exibe duas alternativas de mapeamento possíveis para essa classe. Note que, em ambos os casos, o atributo derivado idade não é mapeado. Além disso, nas duas alternativas, as classes **Cliente** e **CPF** são mapeadas para uma única relação. No entanto, na primeira alternativa, o CEP de um cliente é mapeado para uma relação separada e o atributo CEP da classe é dividido em duas partes (número e dígitoVerificador).

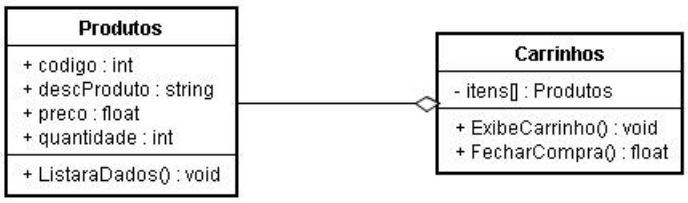
| | |
|---|---|
| 1ª | Cliente( id, CPF, nome, telefone, logradouro, dataNascimento, idCEP )<br>CEP(id, número, sufixo) |
| 2ª | Cliente( id, nome, telefone, logradouro, dataNascimento, CPF, CEP ) |

**Figura 25** – Classe cliente é CPF

Fonte: [Bezerra 2016]

## 3.3 Exercício de fixação

Utilizando os conceitos até o momento apresentados, realize o mapeamento do digrama abaixo.




### 3.3.1 Associações

Para que possamos mapear associações, nós utilizarmos o conceito de **chave estrangeira**. Uma chave estrangeira, como descrito na sub-sessão 3.1.1.2, são utilizadas para relacionar linhas de relações diferentes ou linhas de uma mesma relação. Para isso, há três casos para mapeamento de associações, cada um correspondente a um tipo de conectividade.

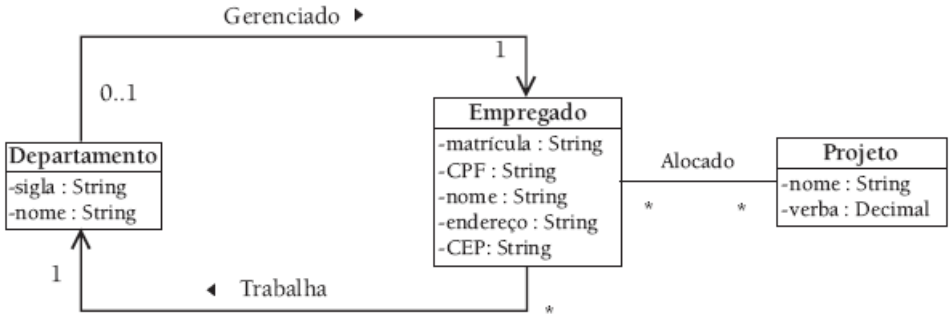



**Figura 26** – Diagrama de classes ilustrando os três tipos de conectividade em associações

Fonte: [Bezerra 2016]

#### 3.3.1.1 Associação um-para-um

Quando há uma associação um-para-um entre **Ca** e **Cb**, deve-se adicionar uma chave estrangeira em uma das duas relações para referenciar a chave primária da outra relação.

**Departamento(id, sigla, nome, idEmpregadoGerente)**

**Empregado( id, matrícula, CPF, nome, endereço, CEP )**

Contudo, há casos que a associação um-para-um pode ser mapeada para uma única relação, como no caso do diagrama apresentado na Figura 24

#### 3.3.1.2 Associação um-para-muitos

Em uma associação um-para-muitos entre objetos de **Ca** e de **Cb**, seja **Ca** a classe na qual cada objeto se associa com muitos objetos da classe **Cb**. Neste caso, deve-se adicionar uma chave estrangeira em **Ca** para referenciar a chave primária de **Cb** [Bezerra 2016].

Como exemplo, considere novamente a Figura 26, na qual se apresenta a associação um para muitos Trabalha. A seguir é apresentada uma extensão do mapeamento anterior considerando essa associação.

**Departamento(id, sigla, nome, idEmpregadoGerente)**

**Empregado(id, matrícula, CPF, nome, endereço, CEP, idDepartamento)**

#### 3.3.1.3 Associações de conectividade muitos para muitos

Quando há uma associação de conectividade muitos para muitos entre objetos de Ca e de Cb, uma relação de associação deve ser criada. Uma relação de associação tem o objetivo de representar a associação muitos para muitos entre duas ou mais relações [Bezerra 2016].

**Departamento(id, sigla, nome, idEmpregadoGerente)**

**Empregado(id, matrícula, CPF, nome, endereço, CEP, idDepartamento)**

**Alocação(id, idProjeto, idEmpregado, nome, verba)**

**Projeto(id, nome, verba)**

Assim, de forma geral, independente da conectividade, o conceito de chave estrangeira é sempre utilizado quando realizamos o mapeamento. Esta técnica é a mesma utilizada para a **agregação** ou para a **composição**. Contudo, para que se possa alcançar o comportamento esperado, devemos utilizar técnicas que envolvem os gatilhos (**triggers**) e procedimentos armazenados (**stored procedures**), os quais não serão tratados neste capítulo.

#### 3.3.1.4 Associação reflexiva

Uma associação reflexiva, ou também conhecida como auto-associação, é um tipo especial de associação. Contudo, o mapeamento aplicado a associações é igualmente ao aplicado em outras associações.

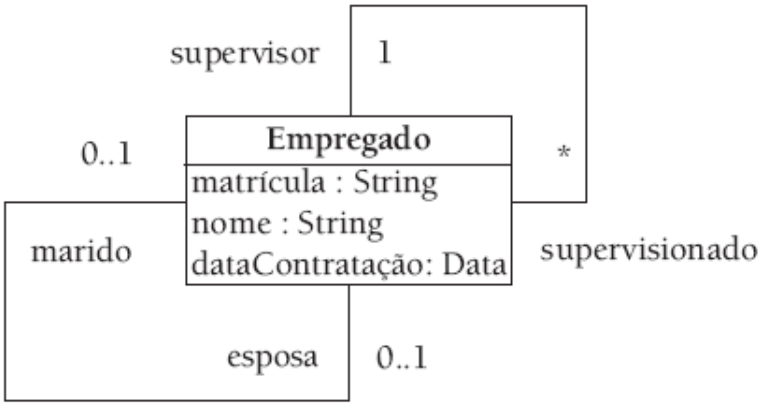



**Figura 27** – Associações reflexivas entre objetos da classe Empregado

Fonte: [Bezerra 2016]

Um possível mapeamento desse diagrama é apresentado a seguir. Note que as chaves estrangeiras idCônjunge e idSupervisor foram definidas na relação Empregado como era esperado, visto que ambas as associações são reflexivas.

**Empregado(id, matrícula, nome, dataContratação, idCônjunge, idSupervisor)**

#### 3.3.1.5 Associação ternária

Associações ternárias, ou também conhecidas como associações n-árias, podem ser mapeadas por meio de um procedimento semelhante ao utilizado para associações binárias de conectividade muitos para muitos. Uma relação para representar a associação é criada e são adicionadas nelas chaves estrangeiras para as **n** classes participantes da associação. Se a associação ternária possuir uma classe associativa, os atributos dela são mapeados como colunas da relação de associação [Bezerra 2016].

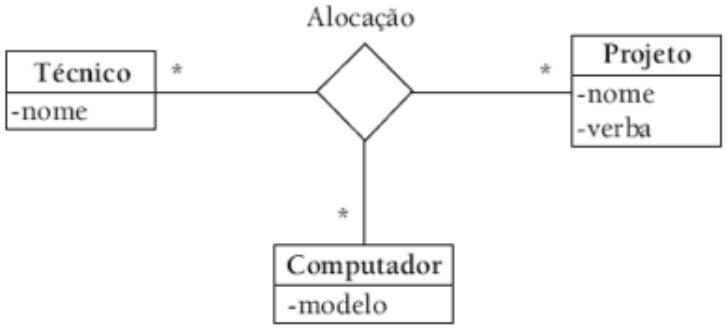



**Figura 28** – Associação terminária/N-ária

Fonte: [Bezerra 2016]

Generalização

Há basicamente três alternativas de se mapear relacionamentos de generalização.

1. Uma relação para cada classe da hierarquia;
2. Uma relação para toda a hierarquia;
3. Uma relação para cada classe concreta da hierarquia.

Assim, vamos aplicar as três formas de mapeamento para o diagrama de classe abaixo.

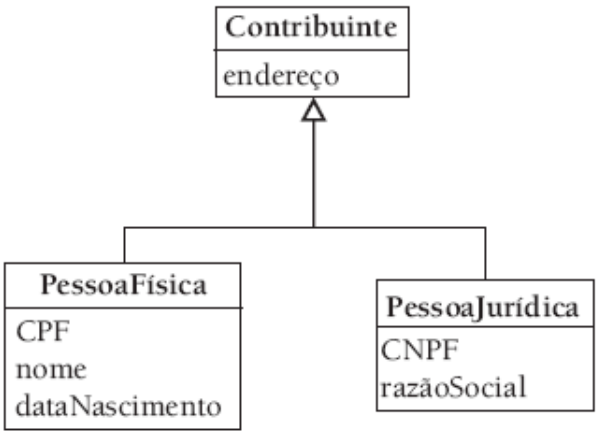



**Figura 29** – Hierarquia de generalização

Fonte: [Bezerra 2016]

Realizando o mapeamento por meio das três alternativas possíveis, obtemos o seguinte resultado:

Alternativa 1

**Contribuinte(id, endereço)**

**PessoaFísica(id, nome, dataNascimento, CPF, idContribuinte)**

**PessoaJurídica(id, CNPJ, razãoSocial, idContribuinte)**

Alternativa 2

**Pessoa(id, nome, endereço, dataNascimento, CPF, CNPJ, razãoSocial, tipo)**

Alternativa 3

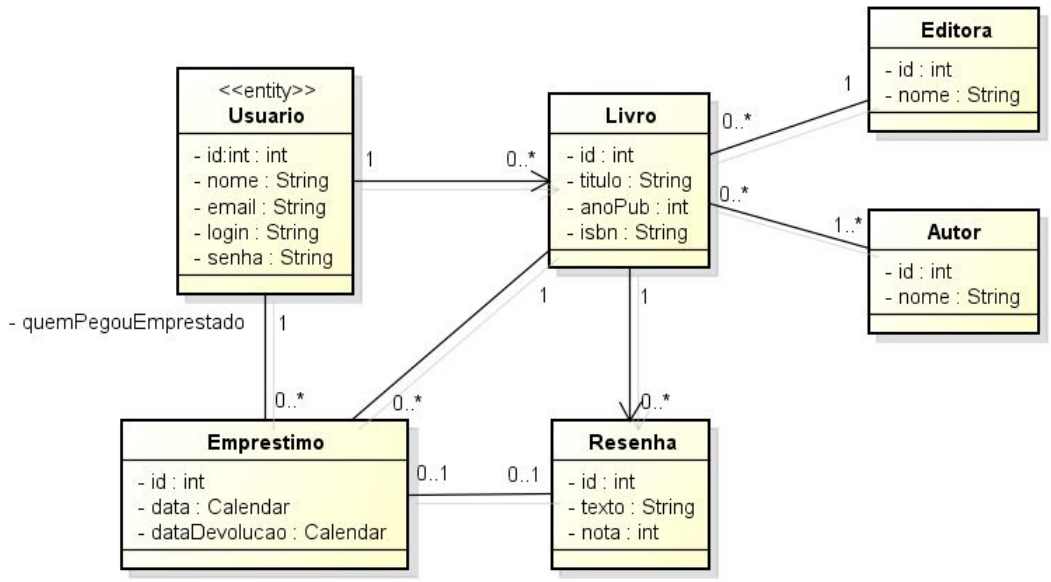
**PessoaFísica(id, dataNascimento, nome, endereço, CPF)**

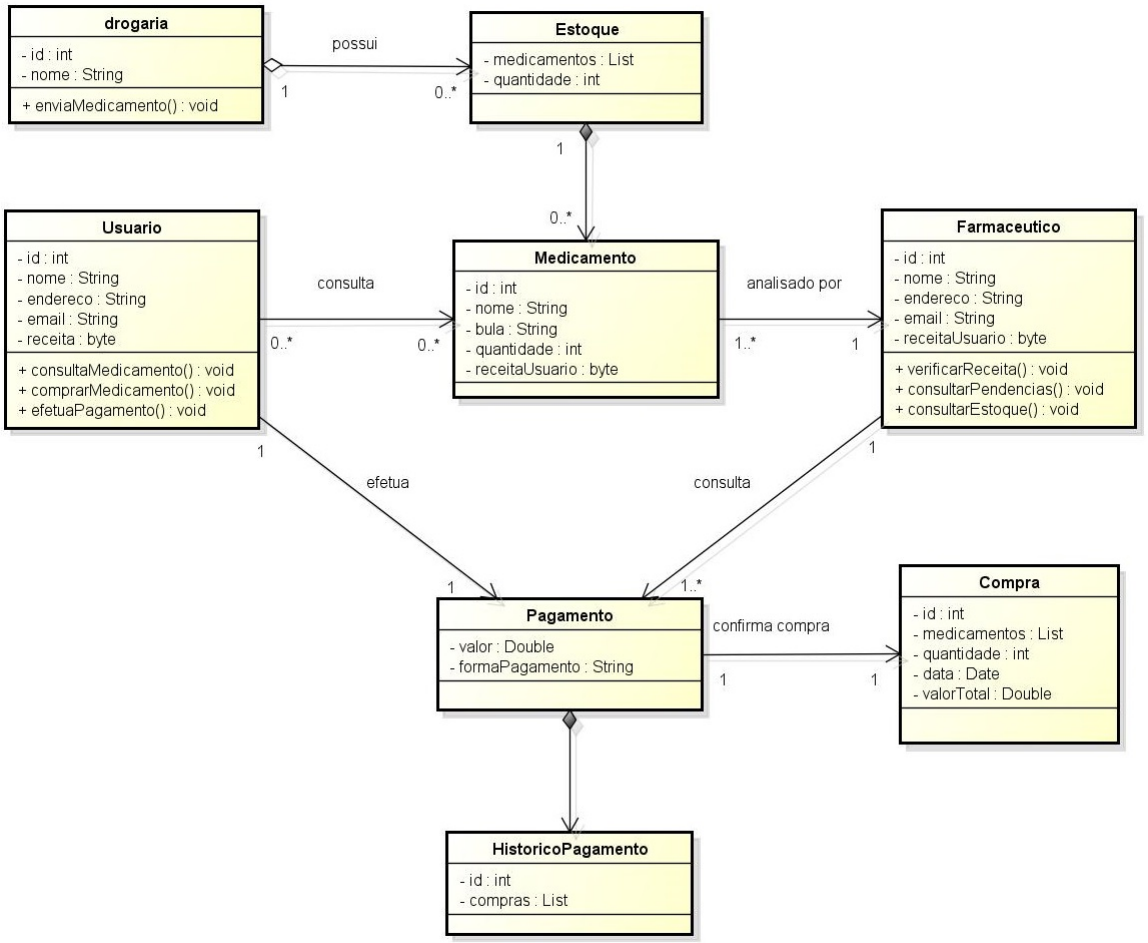
**PessoaJurídica(id, CNPJ, endereço, razãoSocial)**

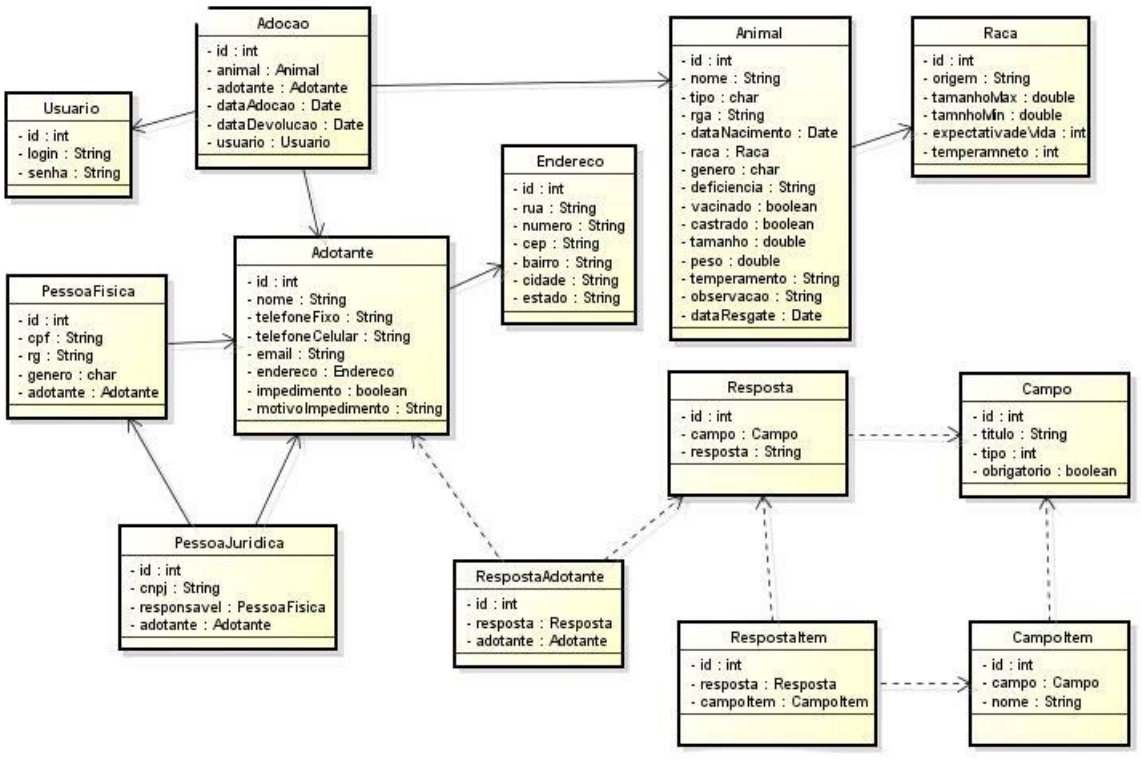
## 3.4 Exercício de fixação

Utilizando os conceitos até o momento apresentados, realize o mapeamento dos diagramas abaixo.

1.




drogaria
- id : int
- nome : String
+ enviaMedicamento() : void
possui
1
0..*
Estoque
- medicamentos : List
- quantidade : int
1
0..*
Usuario
- id : int
- nome : String
- endereco : String
- email : String
- receita : byte
+ consultaMedicamento() : void
+ comprarMedicamento() : void
+ efetuaPagamento() : void
consulta
0..*
0..*
Medicamento
- id : int
- nome : String
- bula : String
- quantidade : int
- receitaUsuario : byte
analisado por
1..*
1
Farmaceutico
- id : int
- nome : String
- endereco : String
- email : String
- receitaUsuario : byte
+ verificarReceita() : void
+ consultarPendencias() : void
+ consultarEstoque() : void
1
efetua
consulta
1
1..*
Pagamento
- valor : Double
- formaPagamento : String
confirma compra
1
1
Compra
- id : int
- medicamentos : List
- quantidade : int
- data : Date
- valorTotal : Double
HistoricoPagamento
- id : int
- compras : List


Adocao
- id : int
- animal : Animal
- adotante : Adotante
- dataAdocao : Date
- dataDevolucao : Date
- usuario : Usuario
Usuario
- id : int
- login : String
- senha : String
Animal
- id : int
- nome : String
- tipo : char
- rga : String
- dataNacimento : Date
- raca : Raca
- genero : char
- deficiencia : String
- vacinado : boolean
- castrado : boolean
- tamanho : double
- peso : double
- temperamento : String
- observacao : String
- dataResgate : Date
Raca
- id : int
- origem : String
- tamnholvlax : double
- tamnholvlin : double
- expectativadeVida : int
- temperamneto : int
Endereco
- id : int
- rua : String
- numero : String
- cep : String
- bairro : String
- cidade : String
- estado : String
Adotante
- id : int
- nome : String
- telefoneFixo : String
- telefoneCelular : String
- email : String
- endereco : Endereco
- impedimento : boolean
- motivoImpedimento : String
PessoaFisica
- id : int
- cpf : String
- rg : String
- genero : char
- adotante : Adotante
Resposta
- id : int
- campo : Campo
- resposta : String
Campo
- id : int
- titulo : String
- tipo : int
- obrigatorio : boolean
PessoaJuridica
- id : int
- cnpj : String
- responsavel : PessoaFisica
- adotante : Adotante
RespostaAdotante
- id : int
- resposta : Resposta
- adotante : Adotante
RespostaItem
- id : int
- resposta : Resposta
- campoItem : CampoItem
CampoItem
- id : int
- campo : Campo
- nome : String

# Referências

BEZERRA, E. *Princípios de Análise e Projeto de Sistema com UML*. [S.l.]: Elsevier Brasil, 2016. v. 3. Citado 10 vezes nas páginas 13, 14, 15, 23, 24, 25, 26, 27, 28 e 29.

GUEDES, G. T. *UML 2-Uma abordagem prática*. [S.l.]: Novatec Editora, 2018. Citado 2 vezes nas páginas 13 e 15.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado na página 4.

RUIZ, E. E. S. *IBm1030 Programação Orientada a Objetos*. 2008. Disponível em: <http://dcm.ffclrp.usp.br/~evandro/ibm1030/constru/heranca.html>. Citado na página 6.

SILVA, D. Lucas da et al. Ontologias e unified modeling language: uma abordagem para representação de domínios de conhecimento. v. 10, 10 2009. Citado na página 8.

SOMMERVILLE, I. *Engenharia de Software. 6ª*. [S.l.: s.n.], 2003. Citado 3 vezes nas páginas 2, 3 e 4.