

Projeto de *Software*



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Diagramas da UML (Parte A)

Responsável pelo Conteúdo:

Prof. Me. Afonso M. Luiz R. Pavão

Revisão Textual:

Prof.^a Esp. Kelciane da Rocha Campos

UNIDADE

Diagramas da UML (Parte A)



- **Introdução;**
- **Objeto;**
- **Classe.**



OBJETIVO DE APRENDIZADO

- Aplicar os elementos da engenharia de *software* e da linguagem unificada de modelagem – UML na representação da solução de um produto de *software*, abordando o modelo estrutural.



Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:

Determine um horário fixo para estudar.

Mantenha o foco! Evite se distrair com as redes sociais.

Procure manter contato com seus colegas e tutores para trocar ideias! Isso amplia a aprendizagem.

Seja original! Nunca plágie trabalhos.

Aproveite as indicações de Material Complementar.

Conserve seu material e local de estudos sempre organizados.

Não se esqueça de se alimentar e de se manter hidratado.

Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Contextualização



Como é possível ter ideias de quais informações poderão ser obtidas com o auxílio de um *software* a partir dos dados de entrada?

Após aprendermos sobre as regras de negócio, requisitos e protótipos, vamos desenvolver nesta unidade os conceitos requeridos para se fazer a composição do diagrama de classe, que é um dos diagramas da *unified modeling language – UML*.

A UML é a linguagem unificada para modelagem de *software*, ou seja, não é uma linguagem de programação de computadores, mas possibilita que o desenvolvedor prepare as “visões” dos conceitos relativos ao processo de negócio que será o alvo da elaboração de um projeto de *software* e sua implementação.

Da UML, diversos diagramas podem ser utilizados – mas normalmente cada instalação (ou equipe de projeto) determina quais deles serão incorporados como “padrão da instalação”.

Como evolução, podemos citar que a orientação e a programação a objetos se popularizaram nos anos 90 e hoje existem diversas linguagens de programação que as suportam, como a C++, Java, Python, dentre outras.

E não há dúvidas de que um dos diagramas mais utilizados para se elaborar o projeto de um *software* é o chamado diagrama de classes, utilizado para se modelar o projeto estruturalmente, conforme veremos a seguir.

Lembre-se de acessar a bibliografia recomendada para complementar este conteúdo bastante específico.

Introdução

A orientação a objetos – OO designa qualquer estilo de desenvolvimento que seja direcionado ao conceito de “objeto”. Uma estratégia orientada a objetos pode ser aplicada na programação ou na análise e no projeto.

O uso da UML traz consigo muitas vantagens:

- Permite que seja gerada uma documentação de referência;
- Garante apresentar mais claramente as classes de um sistema;
- Facilita bastante a comunicação entre os membros da equipe de projeto;
- Permite demonstrar de forma mais efetiva o modelo conceitual do sistema;
- Seu uso adequado possibilita gerar a modelagem de um sistema orientado a objetos.

Objeto

Objeto é qualquer “coisa”, ou seja, qualquer entidade do mundo real que conte-ha ou reúna características e comportamentos. É qualquer coisa que esteja bem definida e cujos limites sejam muito claros e conhecidos.

Esses objetos podem conter dados (em seus campos), os quais são também conhecidos como atributos, e os procedimentos a serem executados, que são os métodos.

Uma das características de um objeto é que um de seus métodos pode provocar alterações nos campos de dados de outro objeto ao qual esteja associado. Ou seja, objetos se comunicam com outros apenas mediante uma mensagem, chamada de métodos.

Assim, um objeto é abstruído de alguma situação ou fato no mundo real e con-tém os atributos que identificam suas características (também chamadas de proprie-dades) e os métodos (ou procedimentos) que provocam a(s) mudança(s) de compor-tamento ou de atividade no objeto ao qual se relaciona.

Assim, é possível organizar melhor os objetos. Por exemplo, as meias (masculi-nas ou femininas, sociais ou esportivas) dentro de uma gaveta.

Outro exemplo mais completo de objeto que posso citar é o cãozinho Joca.

- **Seus atributos físicos são:** é pequeno, é de cor castanha, tem olhos pretos, tem orelhas pequenas e alertas, possui rabo proporcional e patas também. Sua pelagem é curta e quando fica em pé lembra um suricato;
- **Também posso descrever algumas ações dele (métodos):** balança o rabo quando a Jaqueline chega em casa, deita-se de barriga para cima quando lhe chamam a atenção, late quando ouve um barulho estranho, fica com os pelos

das costas arrepiados quando vê ou houve um cão ou gato, atende e corre quando o chamamos.

A representação deste objeto, no caso o cão Joca, possui as seguintes propriedades e métodos.

- **Propriedades (os atributos):** cor castanha, cor dos olhos preta, altura: 15 cm, o comprimento – ponta do focinho à ponta do rabo: 50 cm e a largura: 35 cm;
- **Métodos:** balançar o rabo, latir, deitar, sentar, rosnar.

Classe

O conjunto de objetos que têm as mesmas **características** (ou seja, possuem os mesmos **atributos** e são manipulados pelos mesmos **métodos**) é chamado de **classe**.

Uma classe representa, assim, “um conjunto de objetos que possuem **comportamentos e características** comuns”.

Esses métodos podem ser “chamados” pela própria classe que possui os objetos ou por uma classe exterior que quer se comunicar com ela, ou alterar um estado.

As classes encapsulam os objetos. A rigor, um objeto é conhecido como sendo uma instância de uma classe. A classe é utilizada para criar ou **instanciar** objetos.

Uma **classe** define os atributos e comportamentos **comuns** de mais de um tipo de objeto; ou seja, ambos compartilham os mesmos comportamentos e atributos (as características de uma classe que são visíveis externamente).

O **comportamento** é a ação executada por um objeto quando recebe uma **mensagem** ou informa **resposta** a uma mudança de estado: é algo que um objeto faz.

Esse “passar mensagem” é a operação, chamada de método ou também chamada de função.

“Uma classe representa um conjunto de objetos que possuem comportamentos e características comuns”.

Considerando a orientação e o tratamento dado aos objetos, criamos as classes, no nosso exemplo, a **Classe Cão**.

Sugere-se o uso de nomes curtos contendo apenas letras e dígitos, e no singular com a primeira letra em maiúsculo. Uma classe descreve como os objetos se parecem do ponto de vista da programação, já que ao definir-se uma classe é necessário definir **duas** coisas:

- **Propriedades:** informações específicas, relacionadas à classe de objetos. São as características dos objetos que as classes representam. No nosso exemplo acima, cor, altura, tamanho, largura, etc.; e

- **Métodos:** ações que os objetos de uma classe podem realizar. No exemplo: latir, correr, sentar, rosnar, comer, etc.

Podemos pensar em uma classe como sendo um modelo para criarmos quantos objetos quisermos. Assim, quando criamos uma nova instância de classe, obtemos um novo objeto naquela classe, com todas as características e comportamentos definidos pela classe.

Reforçando: uma classe especifica a **estrutura e o comportamento** – ou operações dos objetos, que são as instâncias da classe.

Portanto, Joca é um **objeto da classe Cão** (criamos uma instância da classe Cão e a chamamos de Joca. Quando criamos uma nova instância de classe (um novo objeto), estamos instanciando a classe).

Em um sistema, são identificadas diversas classes necessárias.

Diagrama de Classes

A UML traz a proposta de ser uma linguagem para modelagem de dados que usa diversos artefatos para representar o modelo do negócio – e um dos artefatos é o **diagrama de classes**.

A representação de uma classe usa um retângulo dividido em três partes, contendo **métodos, atributos e nome**.

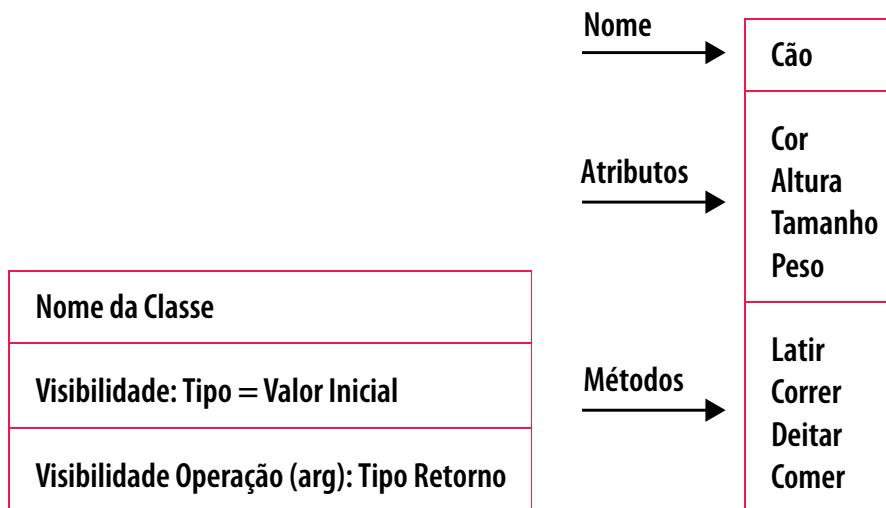


Figura 1

- **Atributos:** representam as **propriedades** que **todos os objetos** da classe têm (todos os cães têm pelos, orelhas, altura, etc. Mas cada objeto terá **valores** particulares para seus **atributos** (alguns são mais baixos, outros são maiores, etc.). Uma classe pode ter qualquer número de atributos. Num modelo, os atributos devem ser de um tipo simples (inteiro, texto e talvez data); não podem conter outros objetos;

- **Métodos:** ações que implementam uma **operação**. Uma classe pode ter qualquer número de métodos e dois métodos em duas classes podem ter o mesmo nome. Todos os métodos que vão implementar a operação precisam respeitar exatamente sua assinatura (mesmo nome, mesmo número de atributos, com os mesmos tipos e o mesmo número de ordem). Um método não pode acrescentar ou cortar um parâmetro, pois isto seria uma **violação** do polimorfismo. O que pode ser feito, no caso de cortar um parâmetro, é simplesmente ignorá-lo na implementação.

Pode-se dizer que o diagrama de classe é um dos principais diagramas estruturais da UML, pois ilustra as classes, as interfaces e os relacionamentos entre elas.

Os atributos e as operações da classe, as restrições de como os objetos podem ser conectados e também os tipos de objetos no sistema e seus relacionamentos, que podem ser de **associações ou de abstrações**.

Para representar a visibilidade dos atributos e operações em uma classe, usam-se os símbolos com os significados:

- + **público:** visível em qualquer classe de **qualquer** pacote;
- # **protegido:** visível para qualquer classe do **mesmo** pacote;
- - **privado:** visível somente para **a** classe.

E para representar o relacionamento entre as diversas classes de um *software*, é necessário compreendermos a integração entre elas.

Relacionamento entre Classes

O diagrama de classes identifica todos os conceitos do domínio que serão implementados no sistema e as relações entre eles.

Veja um exemplo de como os objetos se relacionam:

Um professor **leciona** uma disciplina **para** alunos **numa** sala; um aluno **faz** uma seleção **de** algumas disciplinas optativas **para** cursar; um cliente **seleciona** produtos para **colocar** no carrinho numa **compra on-line**, etc. Essas relações são identificadas no diagrama de classes.

Ele é muito importante, pois define a estrutura do sistema a desenvolver. O diagrama de classes não surge do nada - ele é obtido como consequência do levantamento de requisitos (após a coleta de dados); com isto, é possível se definir as classes e os casos de usos (veremos na próxima unidade).

Os 3 tipos mais importantes de relações entre classes são: de **associação**, de **generalização** (ou herança) e de **dependência**. O relacionamento e a comunicação entre as classes definem responsabilidades. As representações usam a seguinte notação (Figura 2):

Associação: _____

Agregação: _____ 

Composição: _____ 

Herança: _____ 

Dependência: 

Figura 2

- **Associação (agregação ou composição):** são relacionamentos estruturais entre instâncias e especificam que objetos de uma classe estão ligados a objetos de outras classes. A associação pode ser unária, binária, etc. A associação pode existir entre classes ou entre objetos e é representada com uma linha sólida que conecta duas classes;
- **Agregação:** tipo de associação (é parte de, todo/parte, todo-agregado) onde o objeto é um atributo do todo; os objetos somente são criados se o todo ao qual estão agregados for criado. Pedido é composto por itens de pedidos. Um apartamento pode ter de zero a muitas garagens;

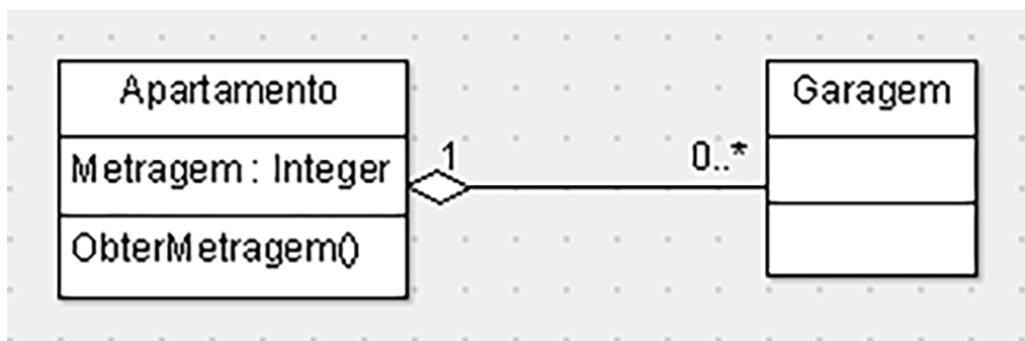


Figura 3 – Associação por agregação

Fonte: Acervo do Conteudista

- **Composição:** relacionamento entre um elemento (o todo) e outros elementos (as partes), onde as partes só podem pertencer ao todo e são criadas e destruídas com ele.

Uma empresa pode ter de zero (se Eireli) a muitos departamentos.

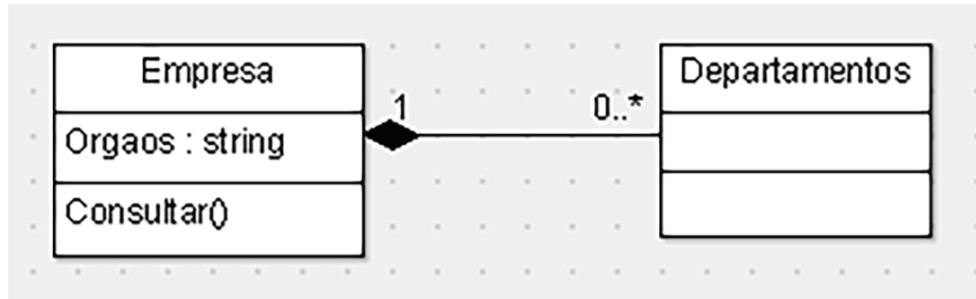


Figura 4 – Associação por composição

Fonte: Fonte: Acervo do Conteudista

A figura 5 mostra um exemplo combinando agregação com composição: um prédio comercial pode ser composto por um ou mais andares, e cada andar pode compor-se de uma ou mais salas. E cada sala pode ter agregada a ela de zero a muitas mesas.

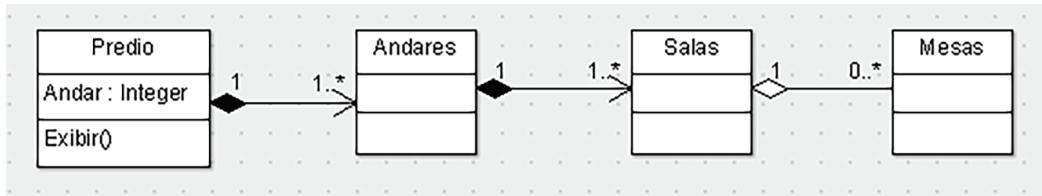


Figura 5 – Associação por agregação com composição

Fonte: Fonte: Acervo do Conteudista

- **Generalização (herança – simples ou composta):** relacionamento entre um elemento mais geral e um mais específico, onde o mais específico **herda** as propriedades e os métodos do elemento mais geral. A relação de generalização no modelo de objetos também é conhecida como herança. Como relação de dependência, ela existe só entre classes. Um objeto particular não é um caso geral de outro objeto; só conceitos (classes no modelo de objetos) são generalizações de outros conceitos.

As classes Terrestre, Marítimo e Aéreo herdam as propriedades da classe Veículo.

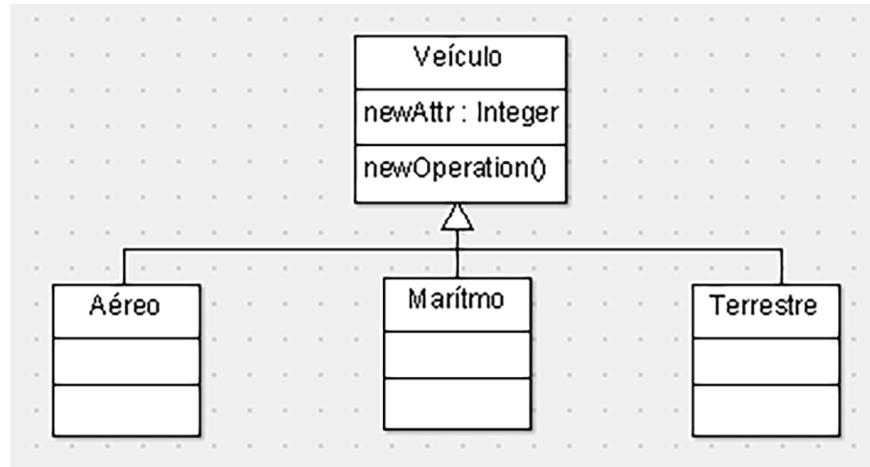


Figura 6 – Associação por generalização ou herança

Fonte: Fonte: Acervo do Conteudista

- **Dependência:** são relacionamentos de utilização nos quais uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe. Ou seja, a alteração de um objeto (o objeto independente) pode afetar outro objeto (o objeto dependente).

A classe Paciente depende de um serviço da classe Médico; qualquer alteração de estado da classe Médico irá afetar um objeto da classe Cliente. Outro aspecto é que a classe Paciente não terá em seus atributos um objeto da classe Médico – o objeto da classe Médico é recebido por um parâmetro de método da classe Paciente.

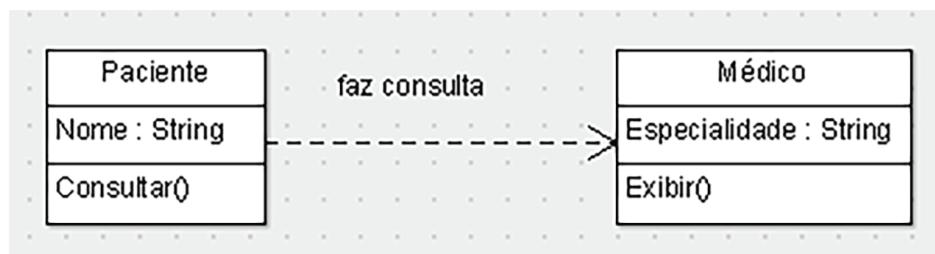


Figura 7 – Associação por dependência

Fonte: Acervo do Conteudista

É importante perceber que há diversos detalhes que devem ser identificados quando se faz o relacionamento entre classes.

Estes devem ter nome, sentido, tipo, etc., conforme Tabela 1.

Tabela 1 – Características de relacionamentos

Nome	Descrição do Relacionamento (faz, tem, possui, consulta, pertence...)
Sentido de leitura	Identifica onde se inicia a relação
Navegabilidade	indicada por uma seta no fim do relacionamento
Multiplicidade	0..1, 0..*, 1, 1..*, 2, 3..7, *..*
Tipo	associação, agregação, composição, generalização e dependência
Papéis	são aqueles desempenhados pelas classes em um relacionamento

Fonte: Acervo do conteudista

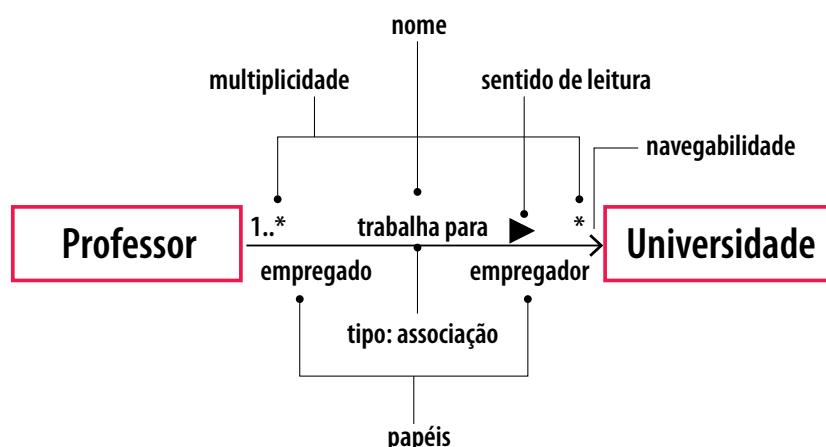


Figura 8 – Características de relacionamentos

Vejam o exemplo de Macoratti:

Supondo que fosse necessário desenvolver um sistema para automatizar um consultório dentário, as etapas básicas (**que na verdade servem para qualquer sistema, mudando-se os atores**) seriam:

- **Levantamento e análise de requisitos do sistema a ser desenvolvido.** No caso, entrevista com o(s) dentista(s) e com as pessoas que trabalham no consultório;
- **Definição dos objetos do sistema:** paciente, agenda, dentista, serviço, contrato, consulta, pagamento, etc.;
- **Definição dos atores do sistema:** paciente, dentista, secretária;
- **Definição e detalhamento dos casos de uso:** marcar consulta, confirmar consulta, cadastrar paciente, cadastrar serviços, etc.;
- **Definição das classes:** paciente, dentista, exame, agenda, serviço; e
- Definição dos atributos e métodos das classes.

Após isto, chega-se ao diagrama de classes do sistema:

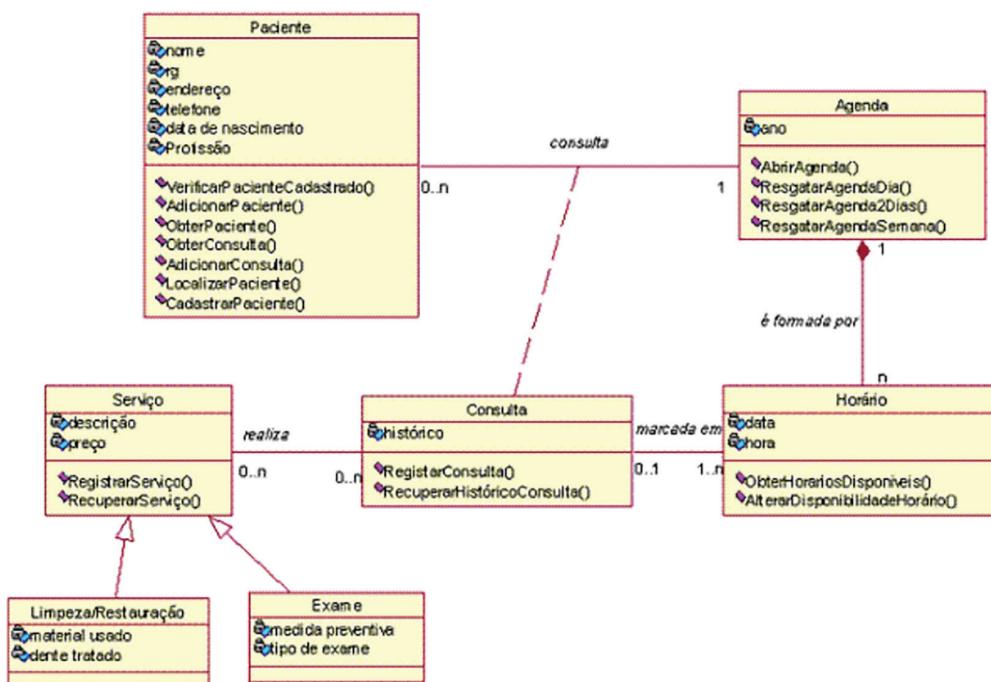


Figura 9 – Diagrama de classes simplificado de um consultório dentário

Fonte: Acervo do Conteudista

- A agenda está **associada** a pacientes;
- A agenda é **composta** de horários;
- O horário está **associado** a consultas;
- As consultas estão **associadas** a serviços;
- O agendamento de uma **consulta depende** da **consulta**; e
- A marcação de exames ou limpeza/restauração **herdam** de serviço.

Provavelmente você já teve contato com esses conceitos da tabela a seguir, mas é bom entender a aplicação dos principais conceitos de UML ao diagrama de classes.

Tabela 2 – Principais conceitos de UML aplicados ao diagrama de classes

Objeto	Tipo abstrato que contém dados e os procedimentos que irão manipular esses dados.
Métodos	Procedimentos dos objetos que determinarão como irão agir quando receber as mensagens.
Classe	Representação de um tipo específico de objeto, composta por sua descrição e identifica as variáveis de classe (suas propriedades) e os métodos.
Subclasse	Uma classe nova com origem em uma classe maior ou superior (pai).
Mensagem	Informação enviada ao objeto para que este mude seu comportamento. A programação orientada a objetos consiste em envios, interpretações e respostas a essas mensagens. Os objetos se comunicam entre si mediante mensagens.
Instância	São os objetos contidos numa classe. Cada objeto usado numa aplicação e pertencente a uma classe é uma instância dessa classe.
Variáveis de instância	São as variáveis que armazenam informações ou dados do próprio objeto e podem ser chamadas de propriedades do objeto.
Hereditariedade	Mecanismo que possibilita compartilhar métodos e dados entre classes, subclasses e objetos. Isto permite criar novas classes programando apenas as diferenças entre a nova classe e a classe-pai.
Encapsulamento	Mecanismo que permite acessar os dados do objeto apenas mediante a utilização dos métodos deste objeto. A comunicação entre os objetos é feita apenas através de mensagens.
Polimorfismo	Uma mesma mensagem pode gerar respostas distintas se forem recebidas por objetos diferentes. O polimorfismo permite que se envie uma mensagem genérica e deixar a implementação a cargo do objeto receptor dessa mensagem.

Fonte: Acervo do conteudista

Voltemos agora ao raciocínio da elaboração de um projeto de um *software* orientado a objetos.

Após ter compreendido as necessidades dos usuários (e clientes) com a coleta de dados na engenharia de requisitos e ter preparado os **wireframes, protótipos ou mockups**, já se tem uma boa ideia de como ficará o *software*.

O mapeamento das telas (mapa navegacional) ajudou a identificar os campos necessários para tornar o *software* funcional e, consequentemente, a se ter uma boa visão para preparar as classes dos objetos que o *software* irá manipular.

É interessante notar como o diagrama de classes faz sentido depois de se ter “a história dos usuários – as *user stories*” e de se ter preparado os *layouts* das telas e o mapa navegacional.

No exemplo acima de consultório dentário, os itens **b/c/d** (conversa com os usuários) identificam o que o sistema deverá controlar e dão base para a preparação dos itens **e/f** (que se tornarão as classes e os objetos).

Em outras palavras, a tabela 1 e a figura 8 serviram de base para se preparar a figura 9 – as classes de objetos que deverão ser manipuladas pelo *software* em um consultório dentário.

Complemente seus estudos com a bibliografia da disciplina constante no plano de ensino e com o material complementar desta unidade, sugerido a seguir.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Leitura

Orientações Básicas na Elaboração de um Diagrama de Classes

Este artigo orienta o estudante na elaboração de um diagrama de classe, procurando estabelecer, de forma sintética, os principais pontos para a abstração dos objetos e classes de um cenário específico.

<http://bit.ly/2uogT0o>

Artigo Java Magazine 43 – Modelagem com ArgoUML

Este artigo apresenta uma introdução ao ArgoUML, focando nos diagramas de casos de uso e diagramas de classes.

<http://bit.ly/2RdxBZ2>

IBM –Diagrama de Classes

<https://ibm.co/2G9CUm5>

O Diagrama de Classes

<https://ibm.co/30He1Ye>

Diagrama de Casos de Uso

<https://bit.ly/2RG9H7t>

Free Class Diagram Tool

VISUAL PARADIGM. *Free class diagram tool.*

<http://bit.ly/2RqJBsl>

Entendendo o Diagrama de Classes da UML

VENTURA, Plínio. Entendendo o diagrama de classes da UML.

<http://bit.ly/2RB4Rss>

Diagrama de Classes de Projeto

OLIVEIRA, Lucas Bueno R. Diagrama de classes de projeto.

<http://bit.ly/2sOKSOy>

Diagrama de Classe

SOUZA, Givanaldo Rocha de. Diagrama de classe.

<http://bit.ly/37fMvnn>

Referências

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário.** São Paulo: Elsevier, 2000.

MACORATTI.NET. **Modelando sistemas em UML – casos de uso.** Disponível em: <http://www.macoratti.net/net_uml2.htm>. Acesso em: 22 jan. 2012.

MEDEIROS, E. **Desenvolvendo software com UML 2.0.** São Paulo: Pearson, 2004.

PADRÃO, L. N. **Análise e projeto de sistemas.** São Paulo: Viena, 2014.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software:** uma abordagem profissional. 7^a ed. Porto Alegre: AMGH, 2016 (*e-book*).

SINTES, A. **Aprenda programação orientada a objetos em 21 dias.** São Paulo: Pearson, 2002.

SOMMERVILLE, I. **Engenharia de software.** 8^a ed. São Paulo: Pearson Addison Wesley, 2007 (*e-book*).

WAZLAWICK, R. S. **Análise e projeto de sistemas de informação orientados a objetos.** São Paulo: Campus, 2004.



Cruzeiro do Sul
Educacional