



## UNIDADE II

---

Projeto de Sistemas  
Orientado a Objetos

Prof. Me. Edson Moreno

# Introdução

- Em um mundo onde a tecnologia de *hardware* evoluiu e segue em constante evolução, em que devemos nos preocupar em desenvolver sistemas de *software* que se adaptem às diversas tecnologias e plataformas de computadores pessoais, sistemas operacionais, dispositivos móveis, entre outros.
- Não só fazer o *software* se adaptar a diversas tecnologias, como também fornecer uma experiência rica ao usuário final, fazer com que ele tenha segurança e conforto no uso, fazer com que ele possa realizar suas atividades de maneira produtiva.
- Na Unidade II de Projeto de Sistemas Orientado a Objetos será tratada a tecnologia de apoio ao projeto orientado a objetos e a transição da fase de análise para a fase de projeto. Boa parte desse suporte é baseado na UML.
  - A UML é uma linguagem de modelagem unificada, utilizada para representar as diversas visões de um *software* durante o seu ciclo de vida de desenvolvimento.

Saiba mais:

LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao processo unificado*. 2. ed. Porto Alegre: Bookman, 2007.

# Tecnologia de Apoio ao Projeto Orientado a Objetos

- Cada visão do projeto orientado a objetos possui uma finalidade bem-definida, cada uma possui um conjunto de diagramas da UML– Unified Modeling Language (Linguagem de Modelagem Unificada) específico para cada objetivo do projeto.
- Antes de utilizar a UML é preciso ter a noção de qual visão estamos querendo modelar. Basta ter uma boa especificação do requisito e é feita a escolha do diagrama correto a ser aplicado.
- Para desenhar os diagramas da UML são utilizadas ferramentas de modelagem. A adoção e o uso consciente de uma ferramenta de mercado são fatores fundamentais para o sucesso nessa fase do projeto.
  - No desenvolvimento orientado a objetos, as principais ferramentas que auxiliam o desenvolvimento orientado a objetos e que serão discutidas nesta unidade são:
    - Ferramentas de modelagem;
    - Ferramentas CASE;
    - *Frameworks*.

# A UML

## A UML não é:

- Uma linguagem de programação.
- Uma plataforma de desenvolvimento.
- Uma ferramenta de modelagem.
- Um *software*.

## A UML é:

- A UML é apenas uma linguagem.
- É independente do modelo de processo adotado.
- É destinada à visualização, especificação e documentação de artefatos.

- A UML (Unified Modeling Language), na definição de seus criadores, Booch, Jacobson e Rumbaugh (2006) “é uma linguagem-padrão para elaboração da estrutura de projetos de *software* [...] adequada para a modelagem de sistemas”.

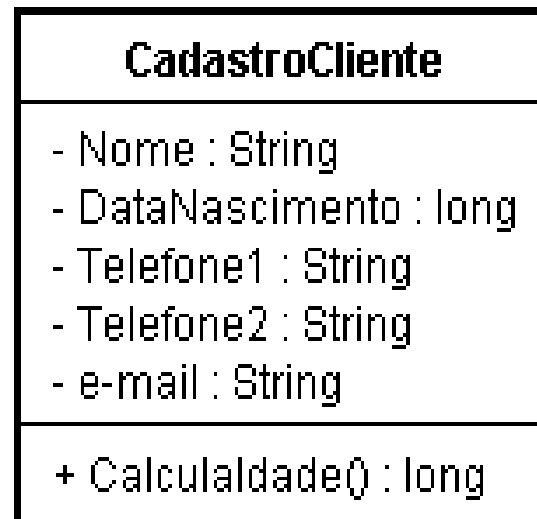
Saiba mais:

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *UML: guia do usuário*. 2. ed. Rio de Janeiro: Campus, 2006.

# A UML – Paradigma

- A UML não é uma linguagem de programação, embora seja possível a geração de código a partir de alguns diagramas, o diagrama de classes, por exemplo, assim como o inverso, ou seja, a geração de diagramas a partir de código-fonte, como é o caso de alguns *frameworks* para o desenvolvimento do código.
- A geração de código não é algo propriamente da UML, mas sim de ferramentas de modelagem que tenham a UML como padrão e que tenham recursos para geração de código ou de engenharia reversa.

## CLASSE



Fonte: MORENO (2020).

## CÓDIGO JAVA GERADO (CLASSE INSTANCIADA)

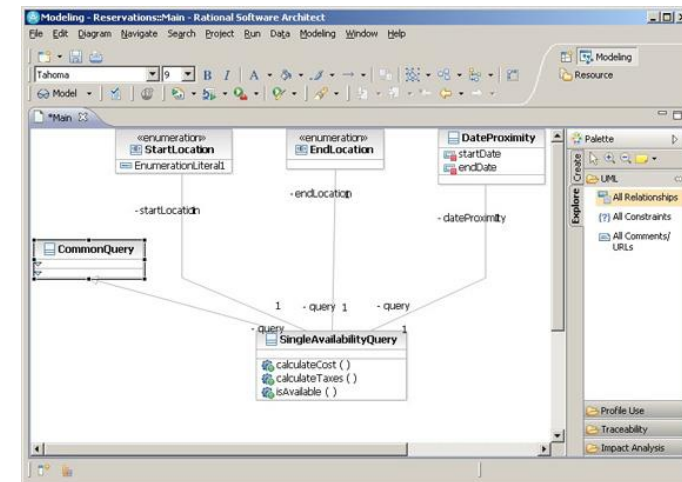
```
public class CadastroCliente {  
    private String Nome;  
    private long DataNascimento;  
    private String Telefone1;  
    private String Telefone2;  
    private String e-mail;  
  
    public long CalculaIdade() {return 0;}  
}
```

# A UML – Ferramentas e técnicas

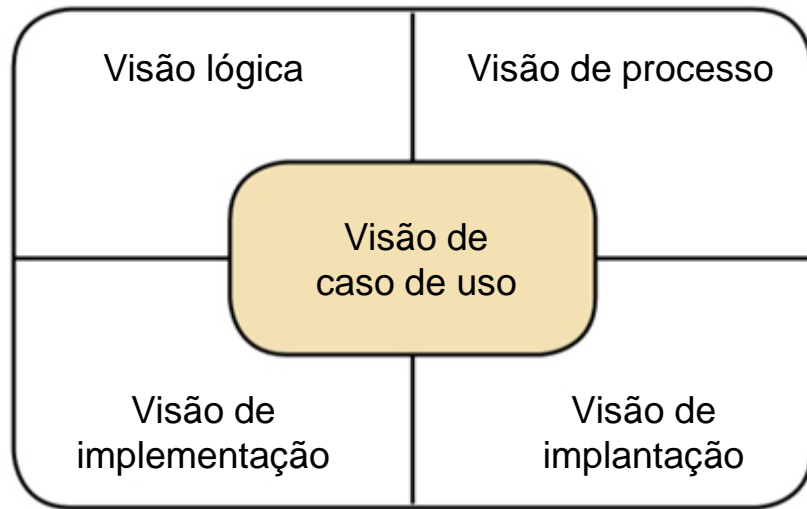
Principais ferramentas que auxiliam o desenvolvimento orientado a objetos:

- Ferramentas de modelagem – são ferramentas específicas para o desenho da arquitetura procedural (não orientada a objetos) e orientada a objetos.
- Ferramentas CASE – Computer Aided Software Engineering (Engenharia de Software Auxiliada a Computador) – dispõe de um conjunto completo de ferramentas (dependendo do fabricante do CASE), que vão desde o planejamento, modelagem, geração de códigos e testes do *software* e do sistema.
- Framework – é um conjunto de ferramentas e conceitos automatizados que une códigos comuns entre os vários projetos de *software* para resolver um problema de um domínio específico.

Saiba mais:  
DIU, Wayne. Using the new features of UML Modeler  
in IBM Rational Software Architect Version 7.5.  
Disponível em  
[https://www.ibm.com/developerworks/rational/library/08/0926\\_diu/index.html](https://www.ibm.com/developerworks/rational/library/08/0926_diu/index.html), 26/09/2008.  
Consultado em 21/06/2020.



# A UML – Projeto do ponto de vista da UML

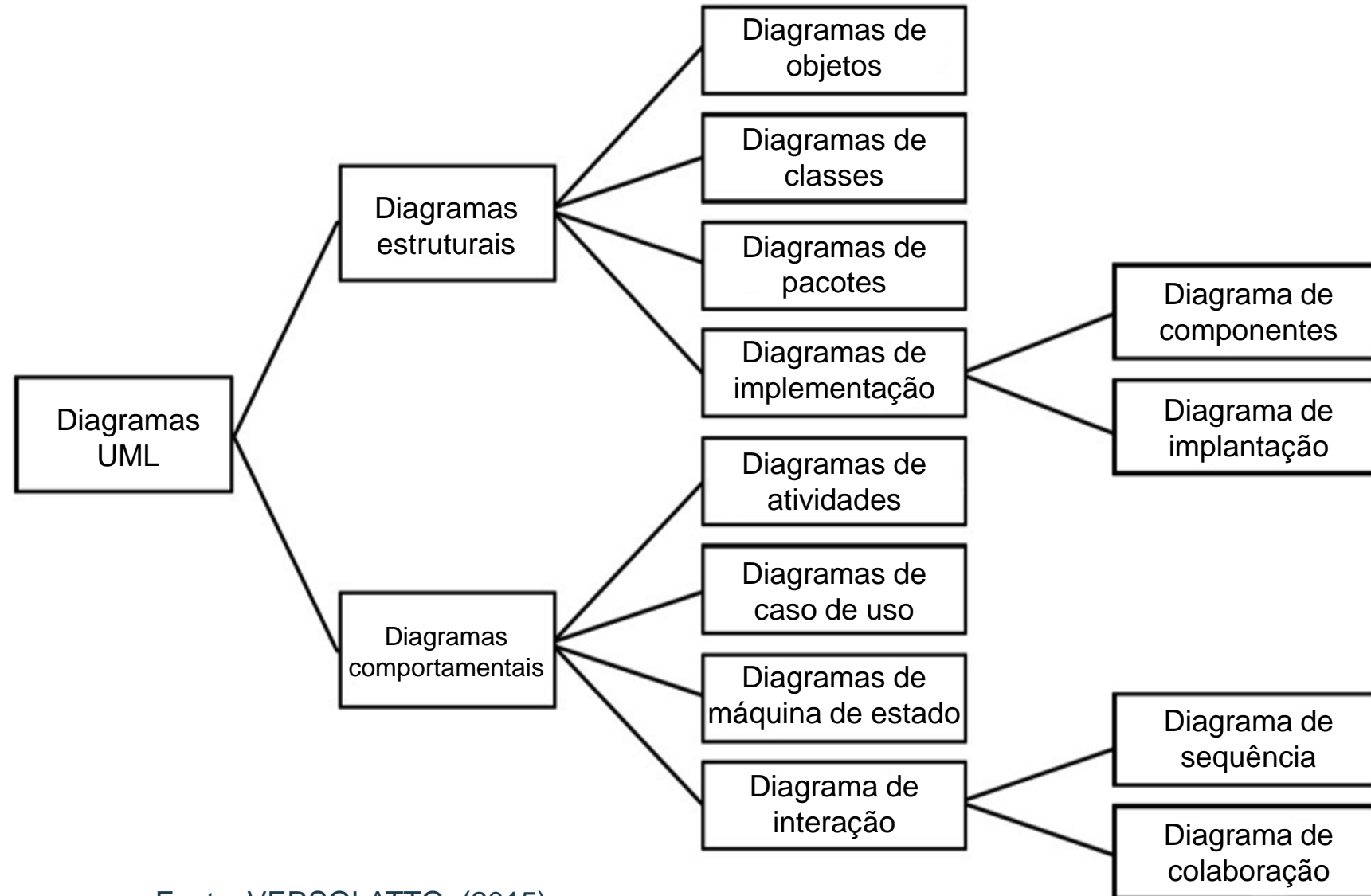


Fonte: VERSOLATTO (2015).

- Caso de uso – o objetivo do caso de uso é capturar os requisitos, as funcionalidades e o comportamento sob a ótica do usuário. O caso de uso é centrado porque o projeto orientado a objeto começa pelo caso e serve de base para as outras atividades do desenvolvimento do sistema.
- Lógica – representa as funcionalidades que serão implementadas no projeto. Se refere à lógica de processamento e à lógica da estrutura da informação.
- Processo – captura aspectos de paralelismo de execução de atividades sob o ponto de vista não funcional, ou seja, não são funções requisitadas pelo cliente ou usuário, porém são práticas que estabelecem a qualidade do produto, tais como usabilidade, segurança, linguagem de programação que será usada e outras mais.
- Implementação – interação dos componentes do sistema.
- Implantação – organização física do sistema para entrega.

# A UML – Projeto do ponto de vista da UML

- Apresentado por Booch, Jacobson e Rumbaugh (2006). A figura ao lado mostra a estrutura dos diagramas da UML.



Fonte: VERSOLATTO, (2015).



# Interatividade

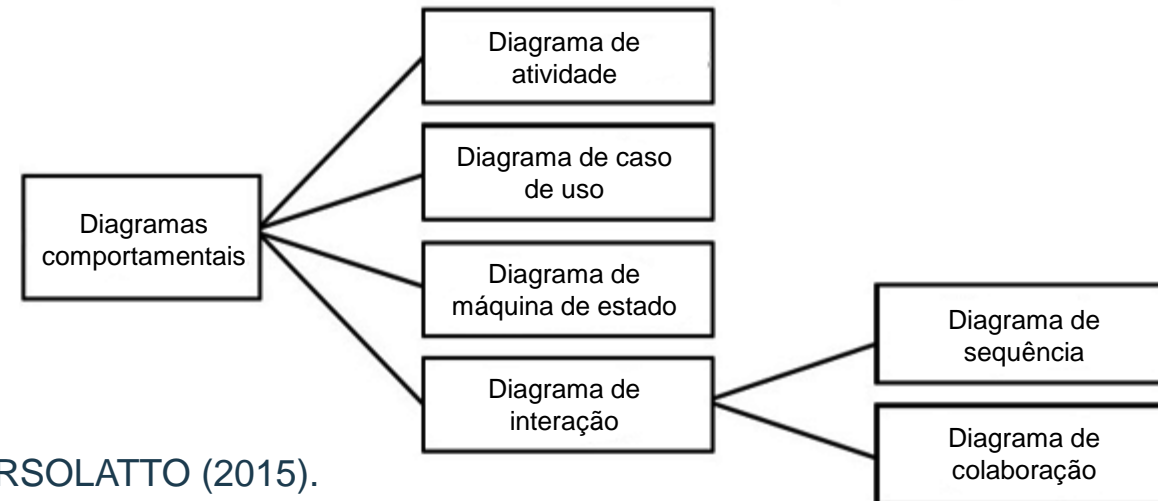
De acordo com Booch, Jacobson e Rumbaugh (2006), a estrutura dos diagramas da UML apresenta duas vertentes para o projeto de sistema orientado a objetos, são os diagramas estruturais e os diagramas comportamentais. Qual das alternativas abaixo expressa somente diagramas comportamentais?

- a) Diagrama de Atividades, Diagrama de Casos de Uso e Diagrama de Sequência.
- b) Diagrama de Atividades, Diagrama de Sequência e Diagrama de Componentes.
- c) Diagrama de Casos de Uso, Diagrama de Classes e Diagrama de Sequência.
- d) Diagrama de Classes, Diagrama de Casos de Uso e Diagrama de Implantação.
- e) Diagrama de Classes, Diagrama de Componentes e Diagrama de Implantação.

# Resposta

De acordo com Booch, Jacobson e Rumbaugh (2006), a estrutura dos diagramas da UML apresenta duas vertentes para o projeto de sistema orientado a objetos, são os diagramas estruturais e os diagramas comportamentais. Qual das alternativas abaixo expressa somente diagramas comportamentais?

- a) Diagrama de Atividades, Diagrama de Casos de Uso e Diagrama de Sequência.
- b) Diagrama de Atividades, Diagrama de Sequência e Diagrama de Componentes.
- c) Diagrama de Casos de Uso, Diagrama de Classes e Diagrama de Sequência.
- d) Diagrama de Classes, Diagrama de Casos de Uso e Diagrama de Implantação.
- e) Diagrama de Classes, Diagrama de Componentes e Diagrama de Implantação.



Fonte: VERSOLATTO (2015).

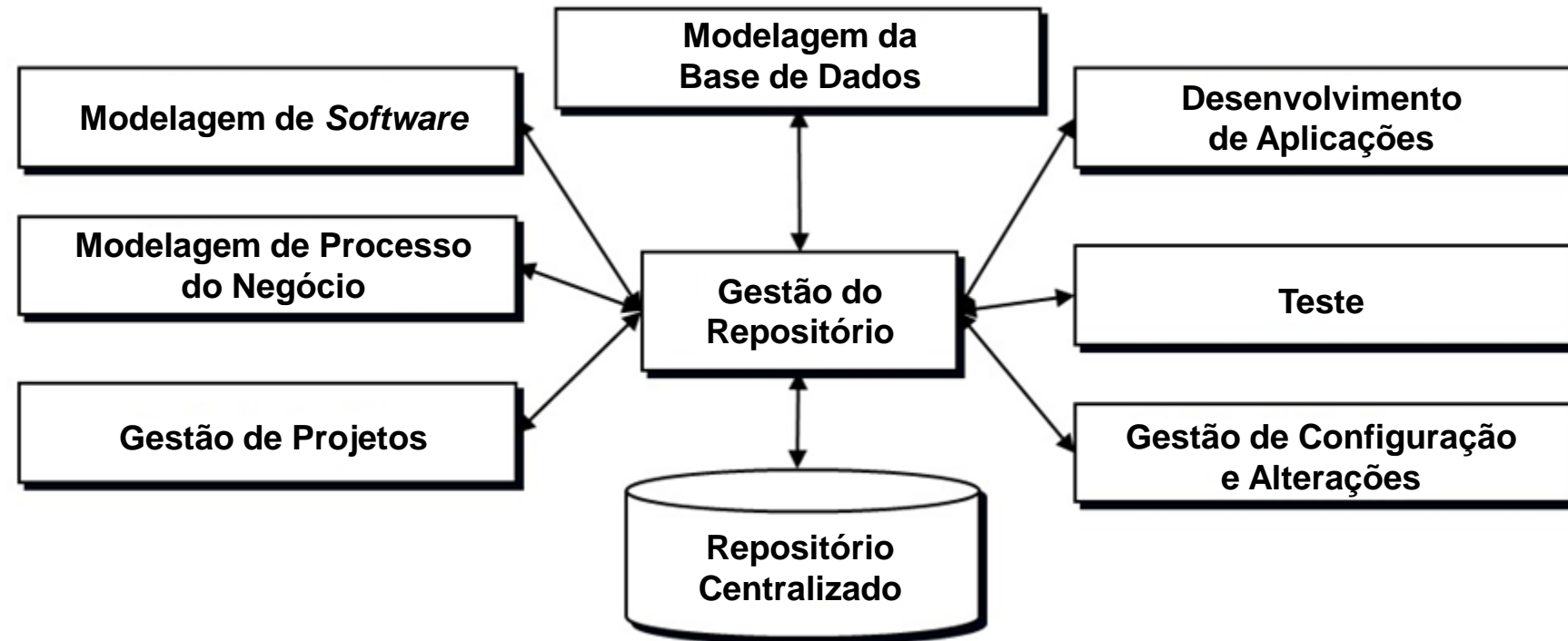
# Ferramentas de modelagem UML

- As ferramentas agem como grandes facilitadoras para o uso da UML. O grande desafio é escolher a ferramenta correta, que atenda aos padrões da UML.
  - Existem muitas ferramentas de modelagem UML, contudo, nem todas atendem aos padrões da UML, são limitadas em uso, limitadas em quantidade de diagramas, ausência de elementos de controle e falta de algumas outras características.
- 
- Fazer um projeto de sistema orientado a objetos com UML não se trata apenas fazer desenhos de diagramas.
    - Na tela a seguir *links* das ferramentas mais usadas no projeto de sistemas orientado a objetos.



# Ferramentas CASE

- CASE – Computer-Aided Software Engineering
- Sommerville (2010) define CASE como “o processo de desenvolvimento de *software* com uso de suporte automatizado”.
- Pressman (2006) define CASE como um sistema de *software* que dá suporte a profissionais da engenharia de *software* em todas as atividades do processo de *software*.



# Ferramentas CASE – As mais usadas no desenvolvimento

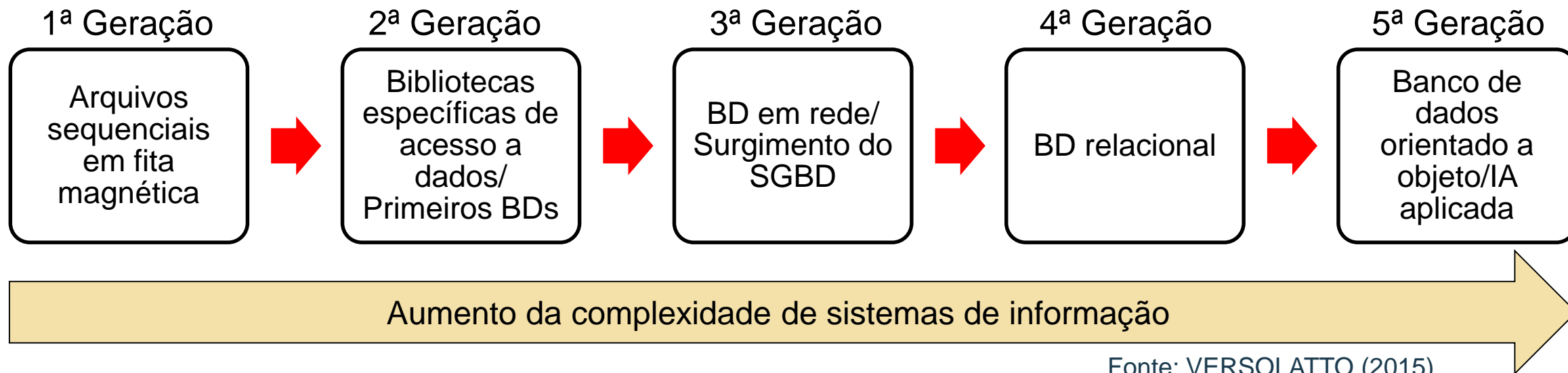
- Astah Community – Disponível em <https://astah.net/products/astah-community/> (*free*)
- Enterprise Architect, Sparx systems – Disponível em <https://sparxsystems.com/> (*free*)
- ERwin Data Modeler, *Computer Associates Technologies Erwin* – Disponível em <http://erwin.com>
- Genexus, ARTech. – Disponível em: <http://www.genexus.com.br>
- Poseidon for UML – *Model to business* – Disponível em: <http://www.gentleware.com> (*trial*)
- Rational Rose Enterprise, IBM – Disponível em [https://www.ibm.com/support/pages/node/306477?mhsrc=ibmsearch\\_a&mhq=Rational%20Rose](https://www.ibm.com/support/pages/node/306477?mhsrc=ibmsearch_a&mhq=Rational%20Rose)



Fonte: acervo pessoal.

# Tecnologia *back-end*

- A tecnologia *back-end* está relacionada ao gerenciamento e armazenamento dos dados e das informações, ou seja, faz a ligação da aplicação com o SGBD\*.



- Observe na figura acima que o ambiente de SGBD\* também evolui para uma estrutura de Banco de Dados Orientado a Objetos.
- (\*) SGBD – Sistemas de Gerenciamento de Banco de Dados (DBMS – Data Base Management System).

# Tecnologia *back-end* – BD orientado a objetos *versus* BD relacional

- A tecnologia *back-end* utiliza dois modelos de SGBD: BD relacional ou de entidade-relacionamento (E-R) e o BD orientado a objetos (OO), similares na definição dos atributos.

## Diferença conceitual do modelo orientado a objetos para o modelo E-R:

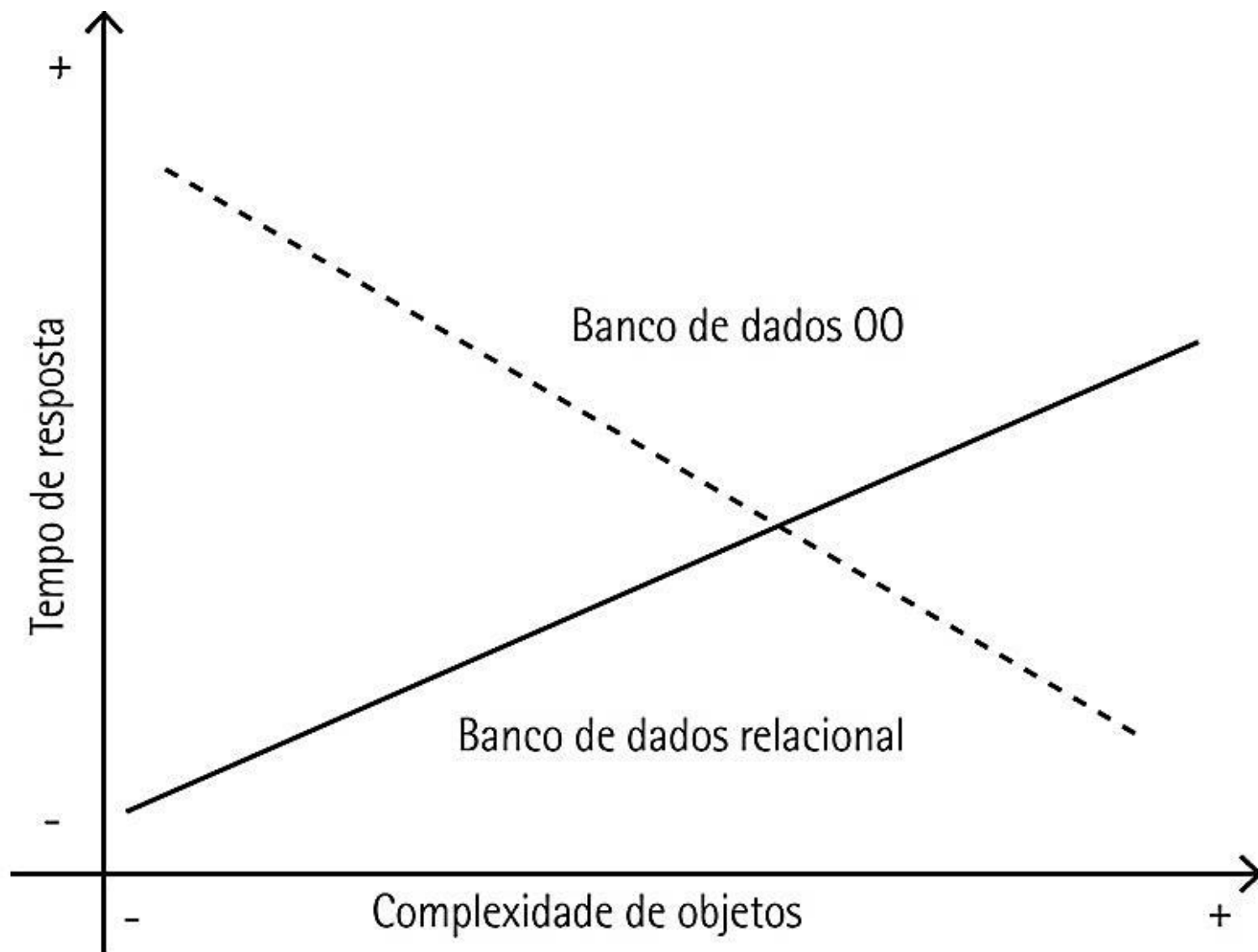
- No modelo orientado a objetos, além dos atributos, cada objeto possui um conjunto de códigos que operam sobre este objeto, chamado de método (ou operação). Na orientação a objetos, o modelo que representa os atributos e os métodos é chamado de classe.
- As identidades dos objetos são diferentes, mesmo que possuam os mesmos valores.
- Adoção de mecanismos de relacionamento: composição, agregação e herança.

# Tecnologia *back-end* – Desempenho no acesso ao BD

- Observe o gráfico de desempenho comparativo entre o BD OO e o BD Relacional na pesquisa de Saxena e Pratap (2013).

Principais padrões de BD OO:

- CORBA (Common Object Request Broker Architecture).
- OMA (Object Management Architecture).





# Interatividade

Analise cada afirmativa como Verdadeira (V) ou Falsa (F) e assinale a alternativa correta.

- I. A tecnologia *back-end* está relacionada com o SGBD.
- II. O *framework* é para modelar a estrutura de componentes do *front-end* e do *back-end*.
- III. Uma ferramenta CASE permite-se trabalhar com as tecnologias *front-end* e *back-end*.

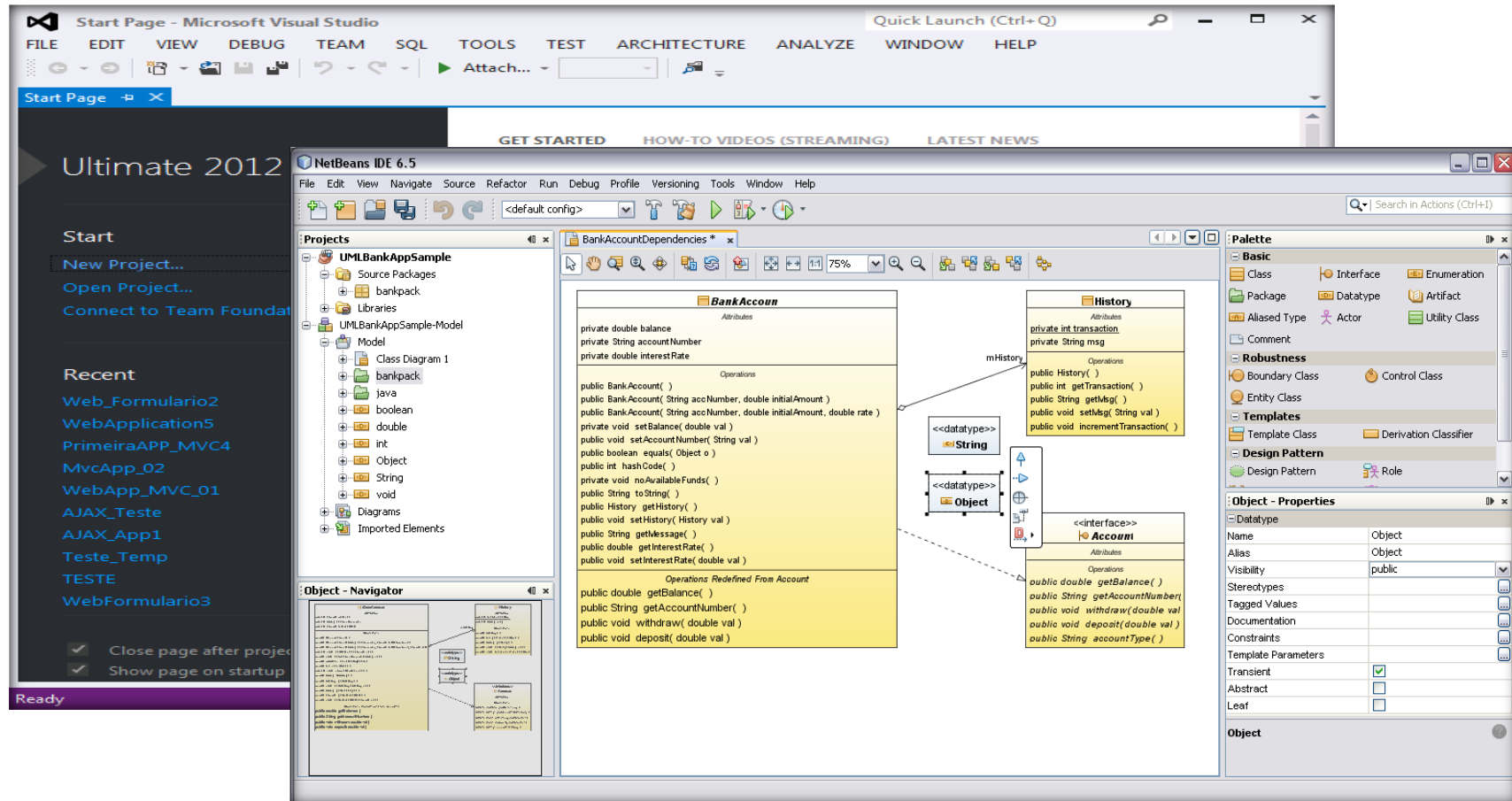
- a) F, F, V.
- b) F, V, F.
- c) V, F, V.
- d) V, V, F.
- e) V, V, V.

# Resposta

Analise cada afirmativa como Verdadeira (V) ou Falsa (F) e assinale a alternativa correta.

- A tecnologia *back-end* está relacionada com o SGBD.
- O *framework* é para modelar a estrutura de componentes do *front-end* e do *back-end*.
- Uma ferramenta CASE permite-se trabalhar com as tecnologias *front-end* e *back-end*.

- a) F, F, V.
- b) F, V, F.
- c) **V, F, V.**
- d) V, V, F.
- e) V, V, V.



# Tecnologia *front-end*

- A tecnologia *front-end* é dividida em duas categorias: ferramentas de modelagem e linguagens de programação OO.
- A lógica de construção do código do *software* expressada para o usuário, sua apresentação, a interface de operação do *software*, a inteligência operacional do *software* é desenvolvida pela tecnologia *front-end*.
- A primeira linguagem orientada a objetos pura utilizada para a construção dentro do paradigma OO foi a linguagem Smalltalk.

Saiba mais:

A linguagem Smalltalk .

Disponível em <http://www.smalltalk.org/>. Consultado em 22/06/2020.

# Tecnologia *front-end* – Linguagens de Programação OO

- De acordo com (LEE; TAPFENHART, 2001), a linguagem de programação OO deve atender quatro características principais:
  1. Encapsulamento de dados – garantir a confiabilidade, independência dos dados e a mantenebilidade do *software*.
  2. Abstração de dados – lógica de processamento dos dados definida pelos métodos.
  3. Coesão dinâmica – mecanismos de troca de mensagens entre os objetos.
  4. Herança – permitir herdar atributos de outras classes.
- Principais Linguagens de Programação OO e que dão suporte a OO: JAVA, C, C++, Python, JavaScript (JS), Perl, PHP, Ruby, Google Go, Swift, VB .NET, Linguagem R e Objective-C.

Saiba mais:

DORNELLES, Nemora. As 15 principais linguagens de programação do mundo!  
Disponível em <https://becode.com.br/principais-linguagens-de-programacao/>, 13/07/2017.

Consultado em 22/06/2020.

# Linguagens OO – *Frameworks* para o desenvolvimento do *software*

- O framework funciona no domínio da aplicação, provendo uma solução completa para uma família de problemas específicos de uma determinada funcionalidade. Usando basicamente um conjunto de classes e interfaces, decompostas em quadros com seus respectivos códigos.
- Modelos e plataformas de desenvolvimento com frameworks
  - Eclipse Modeling Framework (EMF) – Disponível em: <https://eclipse.org/downloads/>
  - Netbeans IDE – Disponível em: <https://netbeans.org/downloads/>
  - Microsoft .NET Framework\* – Disponível em: [https://msdn.microsoft.com/pt-br/library/5a4x27ek\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/5a4x27ek(v=vs.110).aspx)
    - (\*) O Microsoft .NET Framework faz parte do sistema operacional Windows e pode ser usado por outras IDEs. No desenvolvimento do *software* é usado especificamente pelo Visual Studio.

# Passando da Análise ao Projeto

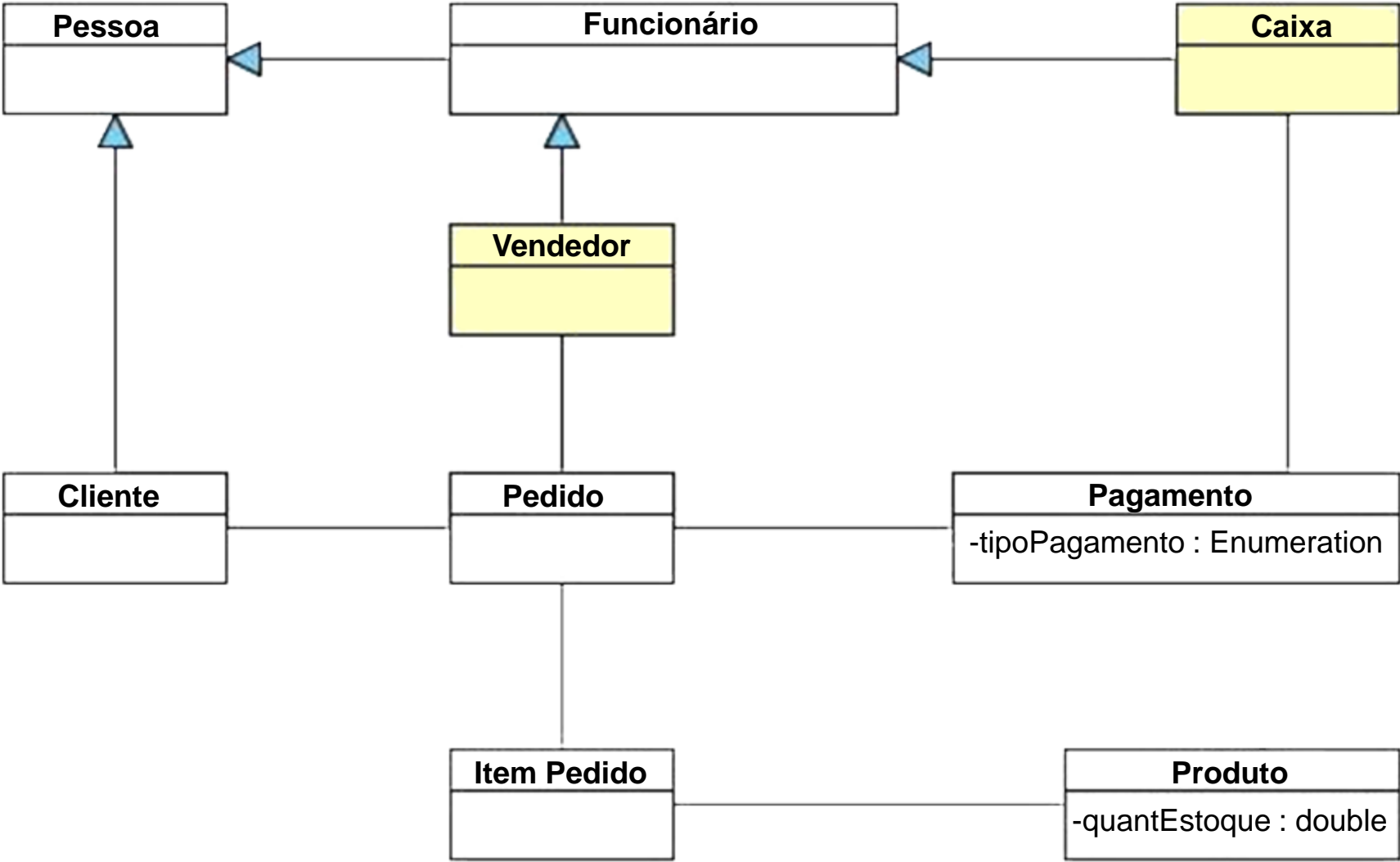
- O projeto inicia a partir do caso de uso, que é a base de tudo e que fornece uma perspectiva do *software* sob o ponto de vista do externo a este *software* (negócio/ator).
- Do caso de uso são produzidas as classes. Os objetos servem de apoio na interpretação dos atributos da classe e a sequência da lógica de processamento responde pelos métodos das classes. É nesta fase que se tem a transição da análise para o projeto.

Modelos de classes:

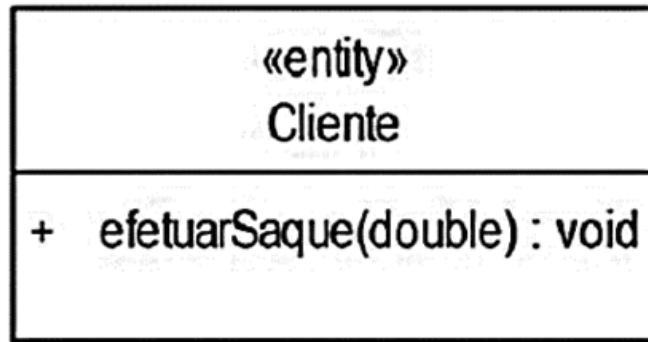
- Modelo conceitual – representa as classes no domínio do negócio.
- Modelo de projeto – desenvolvido na fase de projeto (*design*), é uma evolução do modelo conceitual.
  - Modelo de implementação – é a conversão da classe em código por meio de uma linguagem de programação.

# Modelos de classes – Modelo conceitual

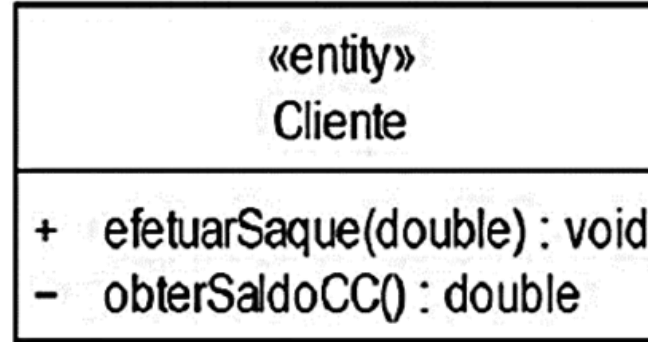
Saiba mais:  
Modelo conceitual de classe para  
“atendimento ao cliente”.  
Disponível em [https://dtic.tjpr.jus.br/wiki/-/wiki/Governan%C3%A7a-TIC/Modelo+Conceitual+de+Classes/pop\\_up](https://dtic.tjpr.jus.br/wiki/-/wiki/Governan%C3%A7a-TIC/Modelo+Conceitual+de+Classes/pop_up). Consultado em 22/06/2020.



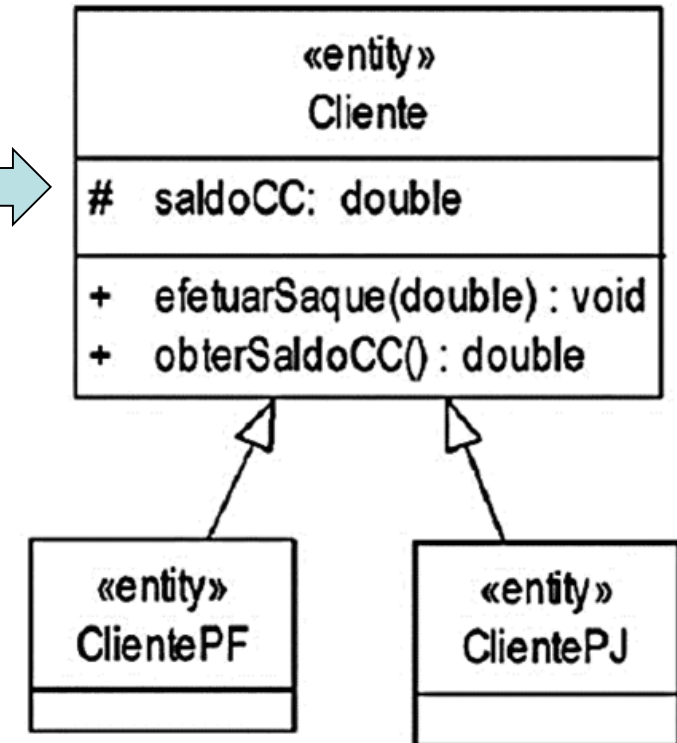
# Modelos de classes – Modelo de projeto



Nível Público – possível de ser acessada por qualquer classe.



Nível Privado - inclusão de atributo ou operação que só pode ser chamada na classe. Identificado pelo sinal “-”.



Nível Protegido - inclusão de atributo protegido, restrito apenas para essa classe. Identificado pelo sinal “#”.



# Modelos de classes – Modelo de implementação

CLASSE	DEFINIÇÃO DOS CAMPOS	CÓDIGO JAVA GERADO (CLASSE INSTANCIADA)
<div><div>CadastroCliente</div><div><ul style="list-style-type: none"><li>- Nome : String</li><li>- DataNascimento : long</li><li>- Telefone1 : String</li><li>- Telefone2 : String</li><li>- e-mail : String</li></ul></div><div>+ Calculaldade() : long</div></div>	<div>→ nome da classe</div> <div>→ atributos</div> <div>→ operações (ou métodos)</div>	<pre>public class CadastroCliente {      private String Nome;     private long DataNascimento;     private String Telefone1;     private String Telefone2;     private String e-mail;      public long Calculaldade() {         return 0;     } }</pre>

Fonte: MORENO (2020).

- O exemplo acima é a codificação da classe com o uso da linguagem de programação JAVA. No JAVA, observe a instrução “public class CadastroCliente” que determina a criação da classe e que esta classe pode ser acessada por outras classes. A instrução “private” instrui a classe que estes atributos só podem ser chamados na classe. A instrução “public long Calculaldade()” que determina que pode ser acessado por todos.

# Interatividade

Quanto ao modelo de classe de projeto, classifique o nível a ser aplicado:

- I. Cliente executa cadastro pessoal em loja virtual.
- II. Efetua pagamento de compra e gera protocolo de pagamento.
- III. Informa ao usuário da situação de falta de papel de uma impressora.

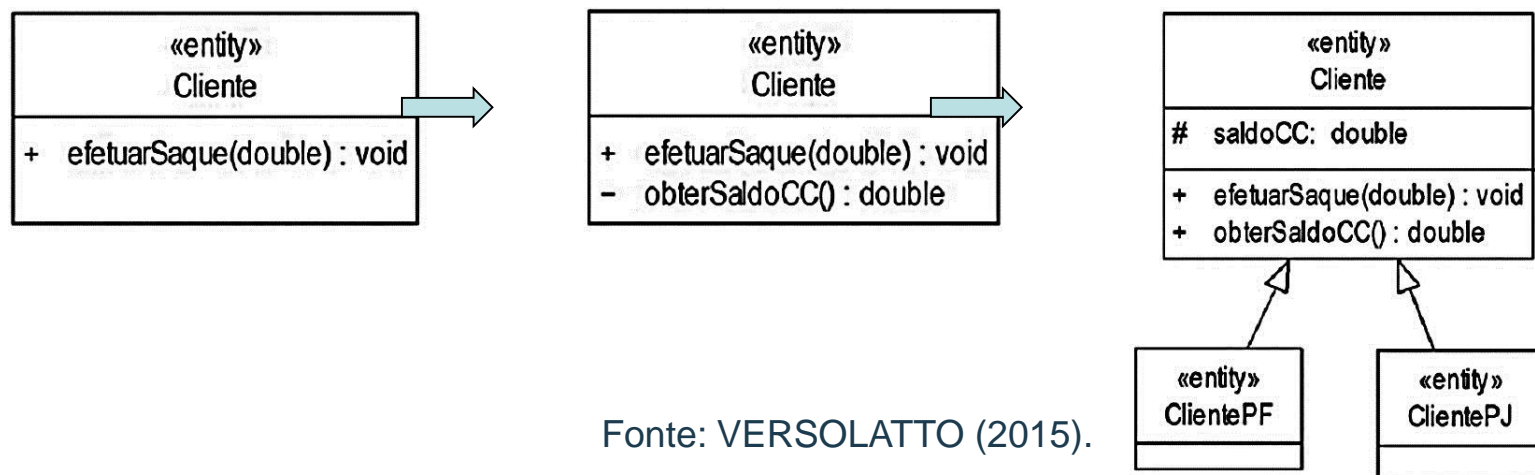
- a) I - Nível privado; II - Nível protegido e III – Nível público.
- b) I - Nível privado; II - Nível público e III – Nível privado.
- c) I - Nível privado; II - Nível público e III – Nível público.
- d) I - Nível protegido; II - Nível privado e III – Nível público.
- e) I - Nível protegido; II - Nível protegido e III – Nível público.

# Resposta

Quanto ao modelo de classe de projeto, classifique o nível a ser aplicado:

- I. Cliente executa cadastro pessoal em loja virtual.
- II. Efetua pagamento de compra e gera protocolo de pagamento.
- III. Informa ao usuário da situação de falta de papel de uma impressora.

- a) I - Nível privado; II - Nível protegido e III – Nível público.
- b) I - Nível privado; II - Nível público e III – Nível privado.
- c) I - Nível privado; II - Nível público e III – Nível público.
- d) I - Nível protegido; II - Nível privado e III – Nível público.**
- e) I - Nível protegido; II - Nível protegido e III – Nível público.



Fonte: VERSOLATTO (2015).

# Atividades clássicas para passagem da análise para o projeto

- O objetivo das atividades de análise necessárias para o projeto está no refinamento do processo de desenvolvimento, ou seja, o modelo de projeto deve ser detalhado para o modelo de implementação.

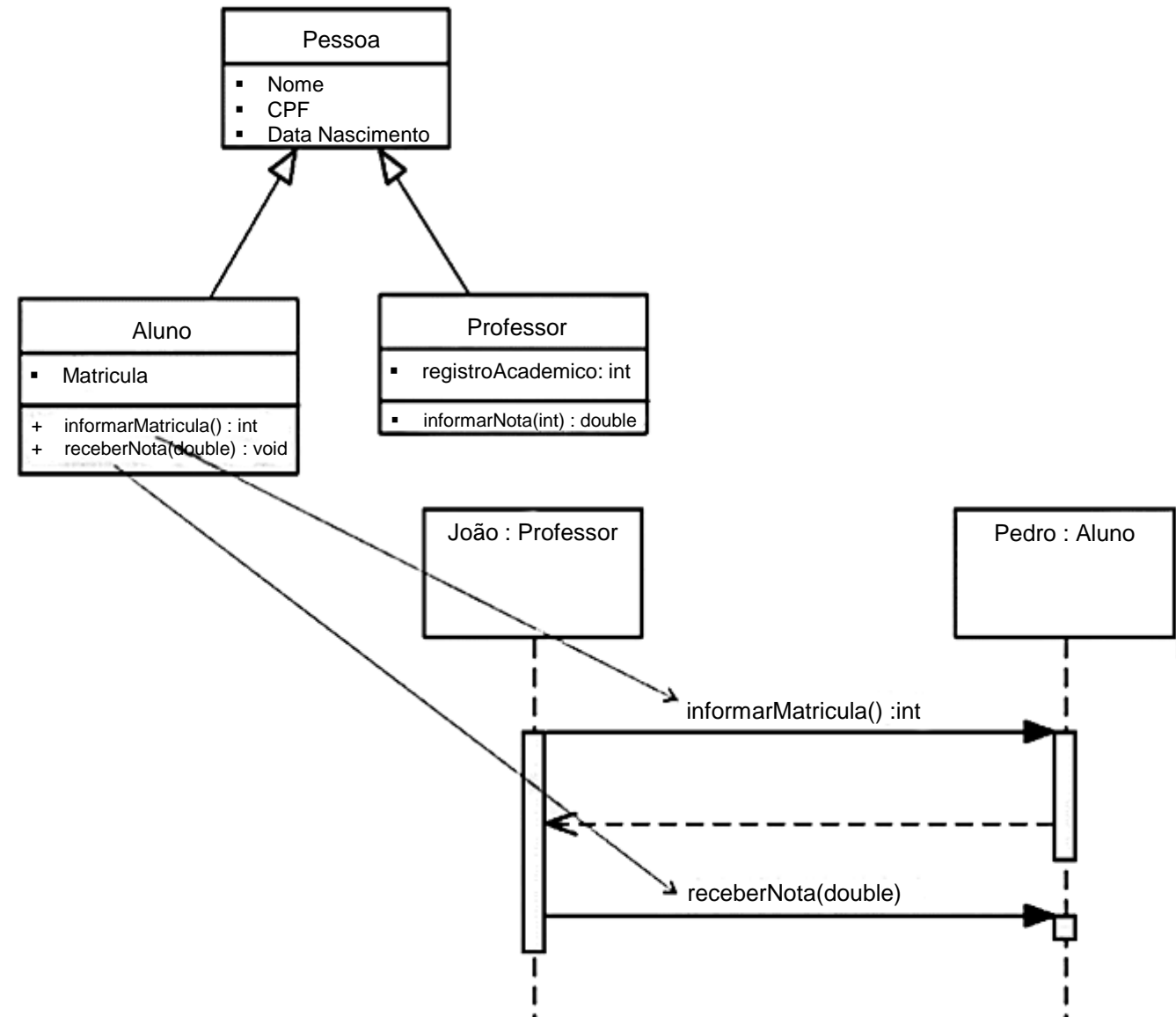
Enumeradas por Bezerra (2006) são três as atividades clássicas de passagem da análise para o projeto:

- Detalhamento dos aspectos dinâmicos do sistema;
- Refinamento dos aspectos estáticos e estruturais do sistema;
- Definição de outros aspectos da solução.

# Atividade – Detalhamento dos aspectos dinâmicos do sistema

O objetivo dessa fase é a representação da interação dos objetos de forma dinâmica. Este dinamismo está relacionado:

- À representação que acompanha as mudanças do objeto da classe pelo Diagrama de Estados.
- Ao comportamento do objeto em uma linha de tempo pelo Diagrama de Sequência, que mostra a troca de mensagens entre os objetos.



Fonte: VERSOLATTO (2015).

## Atividade – Dinâmica do sistema – Mensagens

- As mensagens do diagrama de sequência devem obrigatoriamente refletir os métodos e a assinatura descritos no diagrama de classes.
- A mensagem é diferente de uma informação. O termo mensagem é usado na comunicação dos objetos, que nem sempre é uma informação útil para o ser humano.
  - Veja bem: se um determinado *software* em um servidor está fora do ar, ele sinaliza, às vezes até por um *bit*. A estação que recebe esta situação identifica o código e o transforma em informação para o ser humano.

As mensagens devem obrigatoriamente refletir os métodos e assinaturas descritas na classe:

- Envio explícito de mensagem: refere-se a uma sequência específica de mensagens contidas, por exemplo, em uma fila de mensagens.
- Evento: são mensagens enviadas ao objeto, originárias do ambiente externo ao sistema. Por exemplo: a comunicação do ator com o objeto.

## Atividade – Dinâmica do sistema – Tipos de mensagens

- O tipo de mensagem que identifica os aspectos dinâmicos do sistema refere-se à forma como esta mensagem será apresentada, o que define o comportamento do sistema.

Segundo o padrão de comunicação de interação de objetos, que pode ser observado em Stadzisz (2002), existem dois tipos de mensagem entre objetos:

- Mensagens síncronas: este tipo de mensagem é utilizado quando o método do objeto chamado possui algum tipo de retorno no qual o objeto chamador espera.
- Mensagens assíncronas: o objeto chamador envia a mensagem e continua o seu processamento.

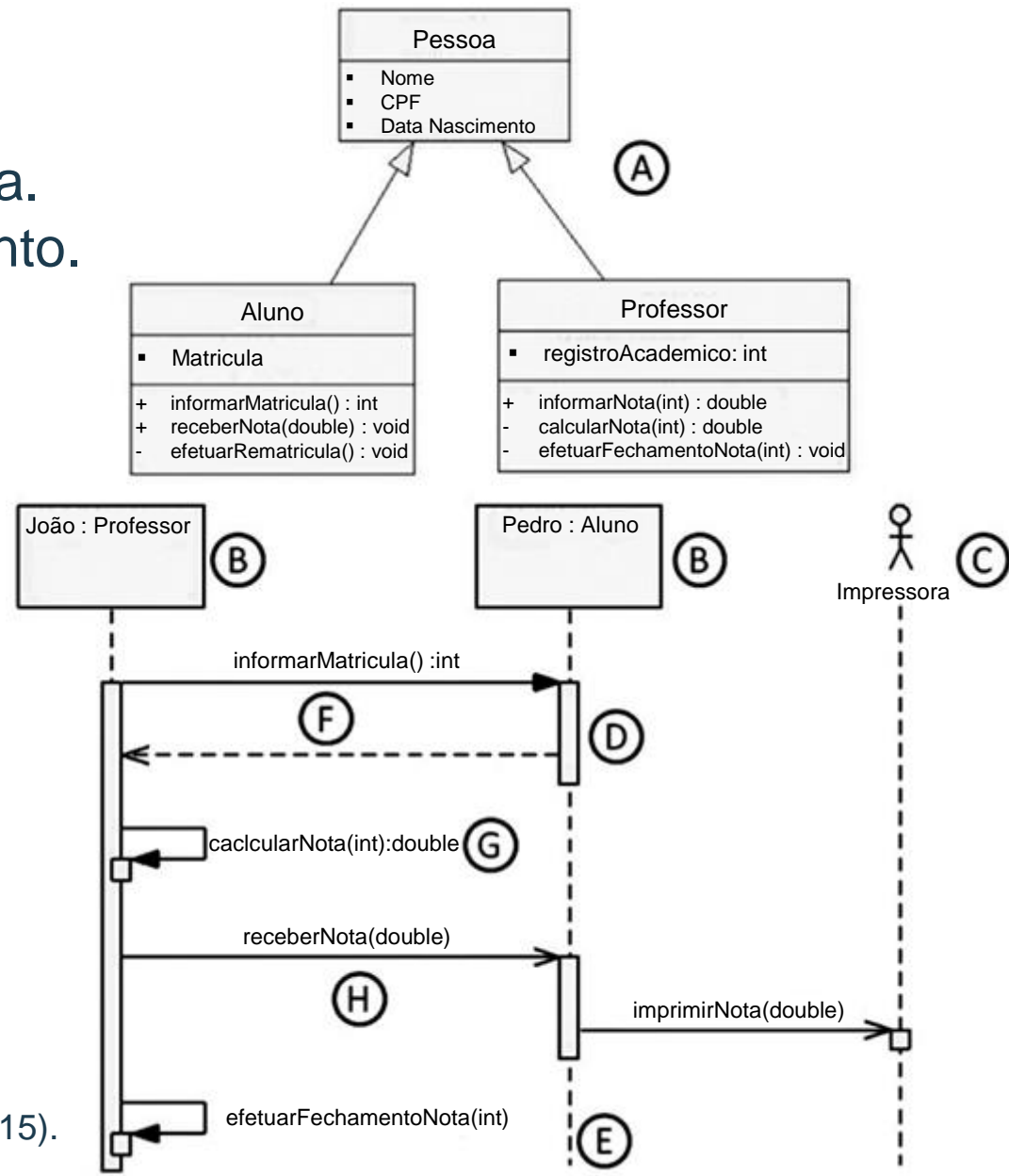
Larman (2007) observa outros tipos de mensagens:

- Autodelegação de mensagens: é quando o objeto envia mensagem para ele mesmo.
- Criação e destruição de objetos: é quanto um objeto é criado, por exemplo, no caso de abertura de um formulário ou destruído, que é quando um objeto é removido da memória.



## Atividade – Dinâmica do sistema – Análise da sequência de mensagens

- A – especificação dos métodos.
- B – objetos instanciados dos métodos da classe.
- C – ator, representa um elemento externo ao sistema.
- D – início de vida do objeto criado para processamento.
- E – linha de tempo de vida do objeto ou ator.
- F – linha tracejada, retorno de mensagem para o objeto.
- G – seta recursiva, autodelegação de mensagem.
- H – linha contínua com seta aberta, envio de mensagem.



Fonte: VERSOLATTO (2015).




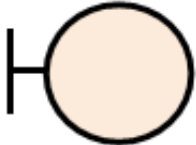

# Refinamento dos aspectos estáticos e estruturais do sistema

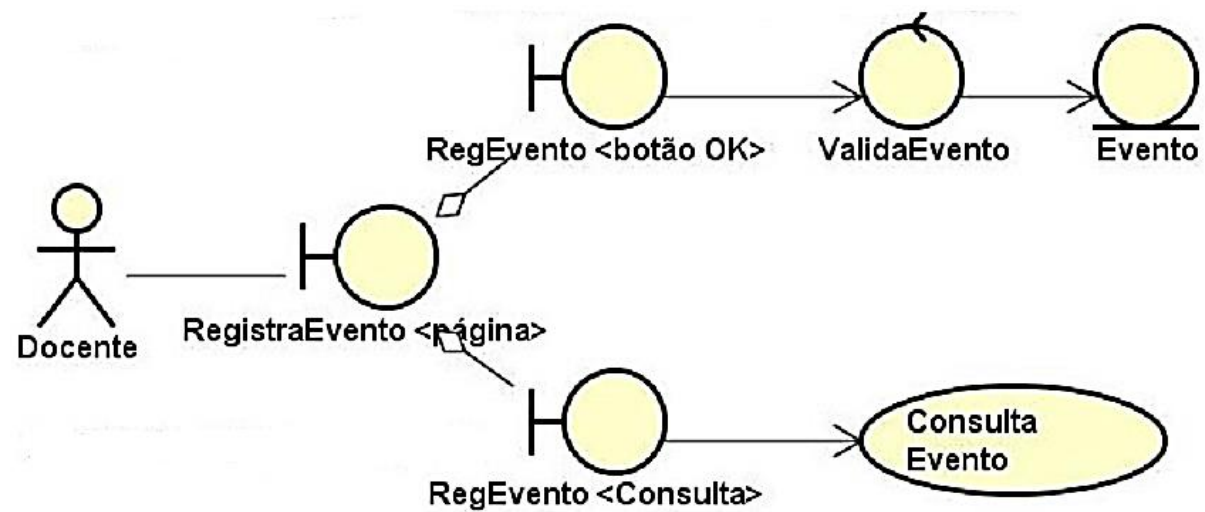
- O refinamento dos aspectos estáticos do sistema tem como objetivo promover a passagem do modelo de classes de domínio para o modelo de classes de projetos (BEZERRA, 2006).
- No caso, a divisão de responsabilidades pode ser encarada como um padrão de projeto com o objetivo de aumentar o reuso e diminuir o acoplamento entre objetos de um sistema. Esse conceito é a base para o padrão de projeto Model-View-Controller (MVC), que será visto com maiores detalhes.
- A definição das responsabilidades dos objetos no sistema é dada pelos estereótipos de classes de análise mostrados na figura baixo:



Fonte: MORENO (2019).

# Aspectos estáticos e estruturais – Estereótipos de classes de análise

	<b>Entidade («entity»)</b> é uma classe passiva que contém informações geradas ou recebidas pelo sistema. Seus objetivos não iniciam interações. Este objeto participa em diferentes realizações de casos de utilização.
	<b>Fronteira («boundary»)</b> é uma classe que se encontra na fronteira de um sistema, contudo, ainda dentro dele. Estes objetos fazem a interface de comunicação entre atores externos com objetos internos do sistema.
	<b>Controle («control»)</b> indica que a classe está na camada de controle de aplicação, que envolve a interpretação e aplicação das regras de negócios. Esta classe tem um comportamento específico para um determinado caso de utilização.



Fonte: MORENO (2019).

Fonte: MORENO (2019).

# Definição de outros aspectos da solução

- A definição de outros aspectos da solução passa para um nível arquitetural do processo de passagem da fase de análise para a fase de projetos. Com o modelo de classes de projeto pronto, começamos a pensar em como organizar essas classes da melhor forma.
- Inicialmente é fazer a decomposição do sistema em subsistemas (ou componentes); esse é o processo de componentização do sistema ou do *software*.
- Um componente de sistema é similar a uma peça para construir o sistema, uma peça que, independentemente do sistema, possa ser dada manutenção, ser substituída e ser feitas melhorias. Pode ser um sistema operacional, uma aplicação, uma estrutura de dados, um computador, um *hardware*, regras de acesso a rede de computadores.

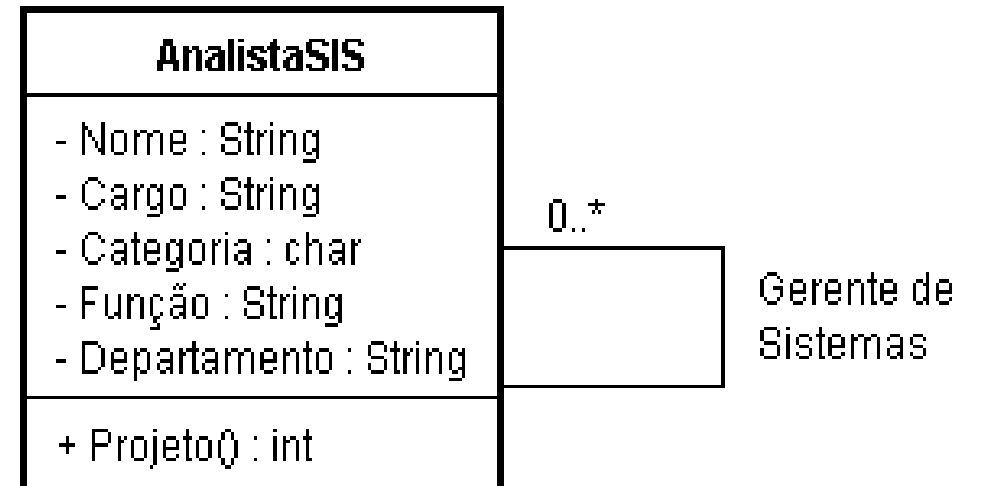
Saiba mais:

HEINEMAN G. T.; COUNCILL, W. T. Component-based software engineering: putting the piecetogether. Boston: Addison-Wesley Longman Publishing, 2001.

SZYPERSKI, C. Component software: beyond object-oriented programming. 2. ed. Boston: Addison-Wesley Longman Publishing, 2002.

# Interatividade

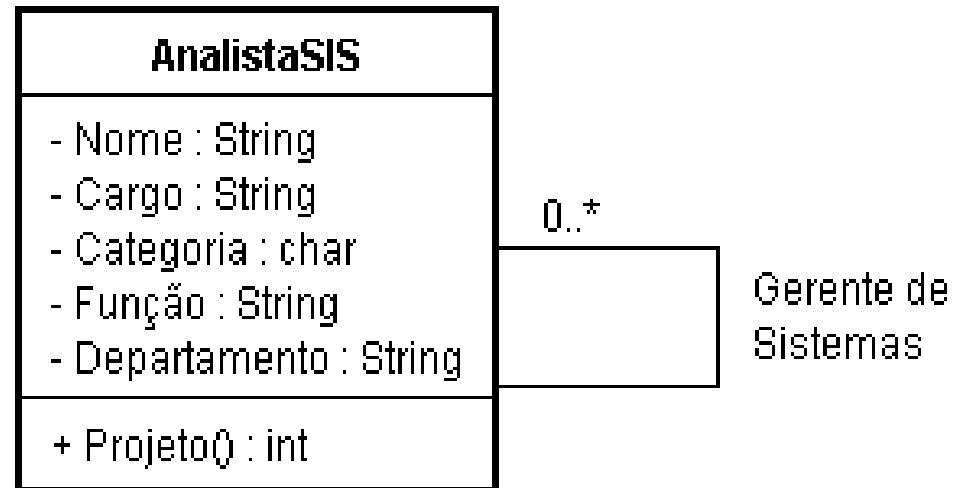
- Com base na classe AnalistaSIS (de Analista de Sistemas) abaixo, assinale a alternativa que corresponde a uma interpretação correta do diagrama.
  - a) A classe AnalistaSIS gerencia outras classes de Analistas de Sistemas.
  - b) A classe AnalistaSIS ocupa cargos de Gerentes de Sistemas.
  - c) A classe AnalistaSIS pode ter influência de Gerente de Sistemas sobre outros Analistas de Sistemas da mesma classe.
  - d) Os Analistas de Sistemas da classe AnalistaSIS são Gerentes de Sistemas.
  - e) Os Analistas de Sistemas desta classe estão sob o comando de um Gerente de Sistemas de outra classe.



Fonte: MORENO (2020).

# Resposta

- Com base na classe AnalistaSIS (de Analista de Sistemas) abaixo, assinale a alternativa que corresponde a uma interpretação correta do diagrama.
  - a) A classe AnalistaSIS gerencia outras classes de Analistas de Sistemas.
  - b) A classe AnalistaSIS ocupa cargos de Gerentes de Sistemas.
  - c) A classe AnalistaSIS pode ter influência de Gerente de Sistemas sobre outros Analistas de Sistemas da mesma classe.**
  - d) Os Analistas de Sistemas da classe AnalistaSIS são Gerentes de Sistemas.
  - e) Os Analistas de Sistemas desta classe estão sob o comando de um Gerente de Sistemas de outra classe.



Fonte: MORENO (2020).

# Referências

- BEZERRA, E. *Princípios de análise e projeto de sistemas com UML*: um guia prático para modelagem de sistemas orientados a objetos através da linguagem de modelagem unificada. Rio de Janeiro: Campus, 2006.
- BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *UML*: guia do usuário. 2. ed. Rio de Janeiro: Campus, 2006.
- KRUCHTEN, P. The 4+1 view model of architecture. *IEEE Software*, Washington, v. 12, n. 6, p. 42-50, nov. 1995.
- LARMAN, C. *Utilizando UML e padrões*: uma introdução à análise e ao projeto orientados a objetos e ao processo unificado. 2. ed. Porto Alegre: Bookman, 2007.

# Referências

- PRESSMAN, R. S. *Engenharia de software*. 6. ed. São Paulo: McGraw-Hill, 2006.
- SAXENA, V.; PRATAP, A. Performance comparison between relational and object-oriented databases. *International Journal of Computer Applications*, p. 6-9, 2013.
- SILVA, Alberto M. R. da; VIDEIRA, Carlos A. E.. UML, *Metodologias e Ferramentas CASE*. Portugal: Edições Centro Atlântico, 2001.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson, 2010.
- VERSOLATTO, Fábio Rossi. *Projeto de Sistemas. Projeto de Sistemas Orientado a Objetos*. São Paulo: Sol, 2015.
- STADZISZ, P. C. *Projeto de software usando a UML*. Paraná: UTFPR, 2002.

**ATÉ A PRÓXIMA!**