Modelagem orientada a objetos com UML

Claudinei Di Nuno
Elisamara de Oliveira

SUMÁRIO

APRESENTAÇÃO

Caro aluno, nesta altura do curso, você já conheceu metodologias e ferramentas ligadas à criação e ao desenvolvimento de *software* desde a coleta dos requisitos com o cliente até a sua implementação final. Para dar continuidade, é importante que conheça como é realizada a modelagem de sistemas orientada a objetos.

O objetivo desta disciplina é apresentar uma série de representações gráficas com base na UML – *Unified Modeling Language*. A UML tem sido adotada como padrão pelas empresas na área de TI. A sua importância é fundamental na construção de *software*, pois oferece uma forma padrão de modelar a arquitetura de um sistema, incluindo aplicações práticas utilizadas nos processos de negócios.

A UML utiliza diagramas para representar um *software* durante todo o ciclo de vida de um sistema. O principal conceito da modelagem orientada a objetos está no fato de imaginar o sistema como um grande conjunto de objetos. A partir desse paradigma, o engenheiro de *software* estuda o sistema sob o ponto de vista da estrutura e do comportamento dos dados.

O conteúdo desta disciplina está subdividido em três partes: modelagem de classes, modelagem de interações e modelagem de estado e de implementação. Ao final, são apresentados os padrões de projeto (*design patterns*), que desempenham um papel fundamental na reutilização de código em outros sistemas.

Todas as representações gráficas dos diagramas UML foram elaboradas na ferramenta *Astah Community*, versão 6.7. O *software* pode ser obtido gratuitamente em http://astah.net/. Nesse site, encontra-se também um tutorial para sua utilização. Além de modelar os diagramas na ferramenta Astah, é muito importante aplicar e integrar os conhecimentos adquiridos da modelagem orientada a objetos na implementação dos códigos na linguagem de programação Java. Essa prática será tratada nas próximas disciplinas específicas do curso.

Convidamos você, caro aluno, a conhecer a modelagem orientada a objetos com UML. É um conteúdo imprescindível para elaboração de projetos de *software* orientado a objetos em Java.

Claudinei Di Nuno

MÓDULO 1 – DESENVOLVIMENTO ORIENTADO A OBJETOS

Caro aluno, este módulo apresenta uma introdução ao desenvolvimento de *software* baseado em objetos e os fundamentos da UML – *Unified Modeling Language*. Esses conceitos teóricos são muito importantes no sentido de nortear os próximos conceitos práticos que serão tratados nesta disciplina.

1. Introdução ao desenvolvimento orientado a objetos

1.1. A evolução da orientação a objetos

Os primeiros conceitos da OO (orientação a objetos) surgiram no início da década de 1960. A linguagem *Simula*, desenvolvida na Noruega por Ole-Johan Dahl e Kristen Nygaard, possibilitava a modelagem de simulações. Os primeiros conceitos sobre objetos e classes surgiram dos projetos desenvolvidos em Simula. Na década de 1970, influenciada por essa linguagem, surgiu a SmallTalk, totalmente orientada a objetos. Entretanto, ela foi disponibilizada para o público somente a partir do início dos anos 1980. Nesse período, surgiram outras linguagens orientadas a objetos, tais como C++, derivada da linguagem C, ADA, Object Pascal e outras.

A linguagem Java, objetivo central deste curso, é uma linguagem OO, tendo sido lançada em 1995 pela Sun MicroSystems. Nos dias atuais, é considerada a linguagem de programação OO mais amplamente utilizada no mundo. Segundo Deitel e Deitel (2005):

> Java é uma linguagem que possibilita a criação de páginas web com conteúdo dinâmico e interativo, o desenvolvimento de aplicativos corporativos de grande porte, o aprimoramento da funcionalidade de servidores *web* e o fornecimento de aplicativos para dispositivos voltados para consumo popular (por exemplo, celulares, smartphones e PDAs).

Para que se possa criar projetos e sistemas orientados a objetos, é necessário conhecer o processo de desenvolvimento da modelagem OO antes de iniciar a criação de códigos na linguagem Java. Assim, vamos

conhecer um pouco sobre a evolução do desenvolvimento de sistemas.

Na década de 1960, não havia critérios para desenvolvimento de sistemas, nem metodologias ou formalidade para a criação de documentos dos projetos. Entretanto, no final da década de 1970, surgiu a análise estruturada. Nessa metodologia, o objetivo do processo a ser informatizado está nas funções que agem sobre os dados. O principal diagrama utilizado nela é o diagrama de fluxo de dados (DFD). Na década de 1980, os autores Tom DeMarco e Chris Gane destacaram-se ao escreverem algumas obras sobre a análise estruturada. Além deles, Edward Yourdon é uma referência muito importante sobre o assunto.

Na década de 1990, as metodologias OO começaram a surgir devido à necessidade da criação de uma que pudesse atender às demandas dessas linguagens de programação. A tabela 1 apresenta as principais metodologias orientada a objetos.

Tabela 1: Metodologias orientada a objetos.

Ano	Metodologia	Autores	Características
1988	Método Shlaer-Mellor ou *Object Oriented Systems Analysis* (OOSA)	Sally Shlaer e Steve Mellor	Metodologia que utiliza ferramentas tradicionais e modelagem de dados.
1990	Método de Coad/Yourdon	Peter Coad e Edward Yourdon	Metodologia chamada de FDD – *Feature Driven Development*.
1991	OMT – *Object Modeling Technique*	James Rumbaugh	Metodologia com ênfase na modelagem semântica de dados.
1992	Método OOSE – *Object-Oriented Software Engineering*	Ivar Jacobson	Metodologia que introduz o conceito de caso de uso.
1994/1995	Método Booch	Grady Booch	Metodologia que utiliza técnicas de desenho OO.

Fonte: Produzido pelo autor.

A partir de 1994, Grady Booch e James Rumbaugh unificaram os seus métodos e criaram o Método Unificado, versão 0.8, por intermédio da Rational Corporation. Caso não se lembre dos conceitos, das fases e das disciplinas do Processo Unificado, vale a pena revisar no material da disciplina no qual foi tratado. Esses conceitos são pré-requisitos para melhor entendimento dos conteúdos abordados aqui.

Em 1995, Ivar Jacobson juntou-se à equipe com o seu método OOSE. Em 1996, a Rational encaminhou à OMG (*Object Management Group*) uma proposta para uma notação padrão de modelagem orientada a objetos, que, por unanimidade, foi aceita pela entidade. Nesse mesmo ano, foi criada a ferramenta UML – *Unified Modeling Language*, versão 0.9, que agrega os conhecimentos dos três autores. A figura 1 indica as principais contribuições para a UML.

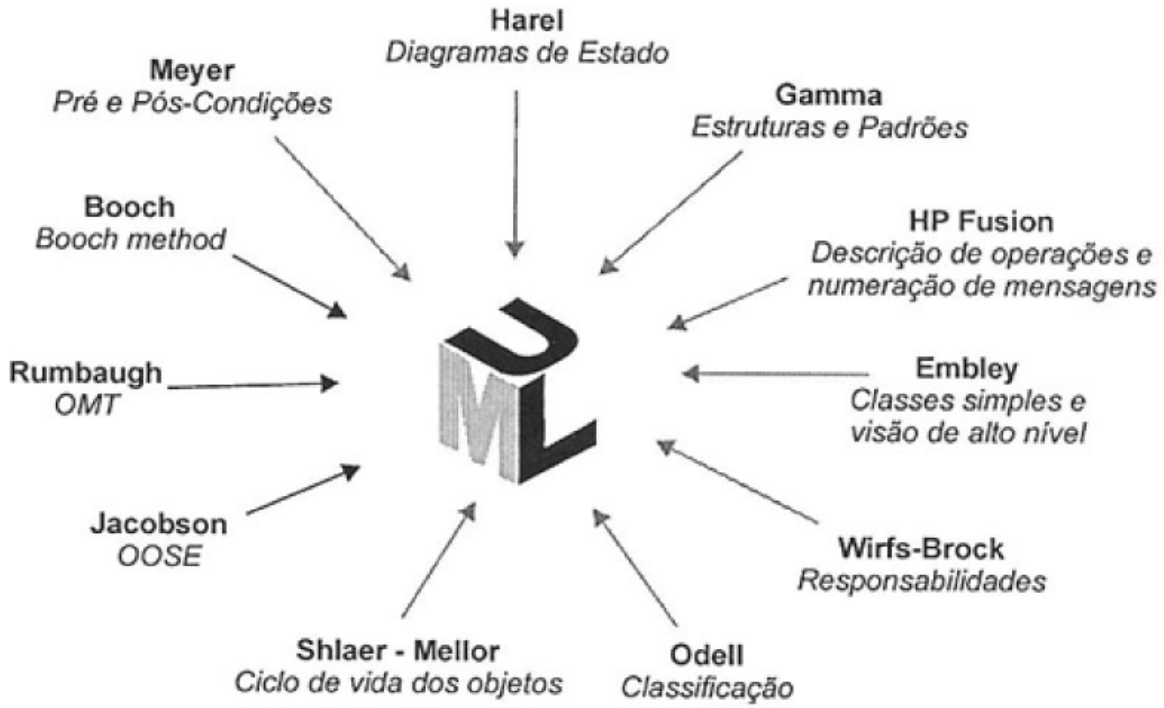

Figura 1: Principais contribuições para a UML. Fonte: Lima, 2011.

1.2. Fundamentos da orientação a objetos

De acordo com Blaha e Rumbaugh (2006), o termo orientado a objetos (OO) significa que organizamos o *software* como uma coleção de objetos distintos, que incorporam uma estrutura de dados e comportamentos.

A OO é uma metodologia utilizada para entender melhor os requisitos do cliente e solucionar os problemas com maior qualidade. A construção da modelagem OO dá-se por meio de objetos, combinando estrutura de dados e comportamento. É um modo de pensar e não uma técnica de programação.

Brooks (1995) observa que a parte difícil do desenvolvimento de *software* é a manipulação de sua essência, devido à complexidade inerente do problema, em vez das especificidades de seu mapeamento para uma linguagem em particular.

O desenvolvimento de *software* ocorre por meio de fases do ciclo de vida dele, como análise, projeto e implementação. Além disso, os conceitos e a notação OO são importantes para a elaboração da documentação de todo o processo de modelagem.

Segundo Blaha e Rumbaugh (2006), a metodologia OO possui os seguintes estágios:

- **Concepção do sistema**: os analistas de negócios ou usuários concebem uma aplicação e formulam os requisitos hipotéticos.
- **Análise**: o analista examina e reespecifica rigorosamente os requisitos da concepção do sistema por meio da construção de modelos.
- **Projeto do sistema**: a equipe de desenvolvimento planeja a arquitetura do sistema para solucionar o problema da aplicação. O projetista do sistema deve decidir que características de desempenho otimizar, escolher uma estratégia para atacar o problema e fazer alocações de recursos experimentais.
- **Projeto de classes**: o projetista de classes elabora objetos tanto do domínio como de aplicação com os mesmos conceitos e notação de orientação a objetos. O foco do projeto de classes são as estruturas de dados e os algoritmos necessários para implementar cada classe.
- **Implementação**: os implementadores traduzem as classes e os relacionamentos desenvolvidos durante o projeto de classes para linguagem de programação, banco de dados ou *hardware* específico.

Os conceitos OO aplicam-se por todo o ciclo de vida de desenvolvimento do sistema. Não se trata de um ciclo de vida do tipo cascata, mas de um processo interativo e incremental.

1.3. Características da orientação a objetos

Blaha e Rumbaugh (2006) identificam algumas características importantes da OO, entre as quais podemos citar:

- **Ênfase na essência de um objeto**: a OO enfatiza o que um objeto é, e não como ele é usado. O desenvolvimento OO dá uma ênfase maior à estrutura de dados e menor à de procedimentos.
- **Sinergia**: identidade, classificação, polimorfismo e herança caracterizam as linguagens OO. Cada um desses conceitos pode ser usado isoladamente, mas juntos eles se complementam com sinergia. Esses conceitos são definidos pelos autores da seguinte forma:
  - ◊ Identidade: significa que os dados são quantizados em entidades distintas e distinguíveis, chamados objetos. Cada objeto tem sua própria identidade inerente.
  - ◊ Classificação: significa que os objetos com a mesma estrutura de dados (atributos) e comportamento (operações) são agrupados em uma classe. Esta é uma abstração que descreve propriedades importantes para uma aplicação e ignora as demais.
  - ◊ Herança: é o compartilhamento de atributos e operações (recursos) entre classes com base em um relacionamento hierárquico.
  - ◊ Polimorfismo: significa que a mesma operação pode se comportar de forma diferente para diferentes classes.
- **Abstração**: permite que o programador se concentre nos aspectos essenciais de uma aplicação, ignorando os detalhes. Isso significa focalizar o que um objeto é e faz antes de decidir como implementá-lo. A capacidade de abstrair provavelmente é a habilidade mais importante exigida para o desenvolvimento OO.
- **Encapsulamento (ocultamento de informações)**: separa os aspectos externos de um objeto, que são acessíveis a outros objetos, dos detalhes internos da implementação, que estão escondidos deles. Pode-se mudar a implementação de um objeto sem afetar as aplicações que o utilizam.
- **Compartilhamento**: as técnicas OO promovem compartilhamento em diferentes níveis. A herança tanto da estrutura de dados quanto do comportamento permite que as subclasses compartilhem um código comum. Esse compartilhamento via herança é uma das principais vantagens das linguagens OO.
- **Modelos**: é uma abstração de algo com a finalidade de entendê-lo antes de construí-lo. As finalidades dos modelos são teste de uma entidade física antes de sua construção, comunicação com os clientes, visualização e redução da complexidade. A modelagem de um sistema dá-se por meio de três modelos:
  - ◊ Modelo de classes: descreve a estrutura estática dos objetos em um sistema e seus relacionamentos. O modelo de classes define o contexto para o desenvolvimento de *software*.
  - ◊ Modelo de estados: descreve os aspectos de um objeto que mudam com o tempo.
  - ◊ Modelo de interações: descreve como os objetos em um sistema cooperam para conseguir resultados mais amplos.

Os três modelos são partes separadas da descrição de um sistema completo, mas estão interligados.

2. UML – Unified Modeling Language

2.1. Fundamentos da UML

Fonte: www.uml.org.

A UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada) foi criada em 1996 por Jacobson, Booch e Rumbaugh, conforme histórico abordado na aula anterior. Em 1997, a Rational lançou a UML versão 1.0. No mesmo ano, com a participação de novos colaboradores, foi lançada a versão 1.1 revisada, e, em meados de 1998, a versão 1.2. A partir desta, novas revisões foram editadas, conforme consta no *site* oficial da OMG (*Object Management Group*), http://www.omg.org/spec/UML/. São elas:

- Março de 2000, UML 1.3.
- Setembro de 2001, UML 1.4.
- Março de 2003, UML 1.5.
- Julho de 2005, UML 2.0.
- Agosto de 2007, UML 2.1.1 (não houve versão 2.1 como especificação formal).
- Novembro de 2007, UML 2.1.2.
- Fevereiro de 2009, UML 2.2.
- Maio de 2010, UML 2.3.
- Março de 2011, UML 2.4.
- Agosto de 2011, UML 2.4.1.
- Outubro de 2012, UML 2.5 (versão em processo de atualização).

A UML não é um método, não é uma linguagem de programação e não é uma especificação para ferramentas de modelagem. Ela é uma linguagem de modelagem.

O OMG (2014) conceitua UML como "linguagem para especificação, construção, visualização e documentação de artefatos de um sistema de *software* intensivo". Nesse caso, artefatos significam testes, projetos, protótipos, requisitos, códigos etc.

A UML utiliza uma notação padrão, comum a todos os ambientes e empresas. Os modelos são documentados e publicados com alcance mundial. Sua notação e sua semântica permitem que as necessidades dos usuários sejam atendidas com qualidade.

Para utilizar a UML de forma eficiente e produtiva, são necessárias ferramentas que auxiliam na modelagem. Algumas das principais ferramentas disponíveis no mercado para modelagem UML incluem:

- Enterprise Architect (Sparx Systems).
- MS Visio (Microsoft).
- System Architect (Choose Technologies).
- Poseidon (Gentleware).
- Together (Borland).
- Rational *Software* Architect (IBM).
- Astah (Change Vision Inc.).
- PowerDesign (Sybase).

Certamente, há outras ferramentas, cada uma delas com suas especificações e características individuais. Assim, torna-se necessário escolher aquela que atenda às reais necessidades de cada projeto.

De acordo com Page-Jones (2001), os sete objetivos que Booch, Jacobson e Rumbaugh estabeleceram para si próprios ao desenvolverem a UML são:

1. Prover aos usuários uma linguagem de modelagem visual expressiva e pronta para uso, de forma que eles possam desenvolver e intercambiar modelos significativos;
2. Prover mecanismos de extensibilidade e especialização para ampliar os conceitos centrais;
3. Ser independente de linguagens de programação e processos de desenvolvimento particulares;
4. Prover uma base formal para entendimento da linguagem de modelagem;
5. Estimular o crescimento do mercado de ferramentas orientadas a objetos;
6. Suportar conceitos de desenvolvimento de nível mais alto, tais como colaborações, estruturas, modelos e componentes;
7. Integrar as melhores práticas.

Page-Jones ainda acrescenta quatro objetivos que estão implícitos no trabalho dos criadores da UML e foram atingidos com êxito:

1. Existência de uma UML mais simples dentro de uma UML abrangente e geral.
2. Utilização em diversas perspectivas de modelagem.
3. Existência de correspondência com o código suficiente para permitir reengenharia.
4. Utilização assistida por computador e também de forma manual.

A OMG - ***Object Management Group*** é uma associação internacional aberta e não tem fins lucrativos. Foi fundada em 1989. Os participantes dela são fornecedores, usuários finais, instituições acadêmicas e agências governamentais. Ao longo do ano, ocorrem quatro reuniões técnicas para discutirem sobre produtos, especificações de processos e tendências da área. Além disso, a OMG atua como produtor de eventos para conferências e oficinas para seus membros ao redor do mundo.

Fonte: www.omg.org.

3. Exercícios do módulo 1

1) Associe:

(1) Método OOSE	( ) Grady Booch
(2) Método OMT	( ) James Rumbaugh
(3) Método OOSA	( ) Ivar Jacobson
(4) Método Booch	( ) Shlaer e Mellor

2) UML significa Linguagem de Modelagem Unificada e é um método de especificação para ferramentas de modelagem.

a. Verdadeiro

b. Falso

3) Segundo Blaha e Rumbaugh (2006) a metodologia orientada a objetos possui os seguintes estágios:

a. Identidade, classificação, herança e polimorfismo

b. Análise, projeto e implementação

c. Concepção do sistema, análise, projeto do sistema, projeto de classes e implementação

d. Classes, estados e interações

4) O OMG é uma associação aberta internacional sem fins lucrativos fundada em 1989.

a. Verdadeiro

b. Falso

5) Indique abaixo a alternativa que não está relacionada aos objetivos de Booch, Jacobson e Rumbaugh ao desenvolverem a UML:

a. Fornecer aos usuários de uma linguagem de modelagem visual expressiva e pronta para uso, de forma que eles possam desenvolver e intercambiar modelos significativos.

b. Integrar as melhores práticas.

c. Estimular o crescimento do mercado de ferramentas orientadas a objetos.

d. Ser dependente das linguagens de programação e dos processos de desenvolvimento particulares.

6) O desenvolvimento orientado a objetos dá uma ênfase maior à estrutura de dados e uma ênfase menor à estrutura de procedimentos.

a. Verdadeiro

b. Falso

7) A modelagem de um sistema dá-se por meio de três modelos. Qual dos itens listados abaixo não se constitui num desses modelos?

a. Herança

b. Classes

c. Estados

d. Interações

8) A orientação a objetos é uma _____ utilizada para entender melhor os _____ _____ e solucionar os problemas com maior qualidade. A construção da modelagem orientada a objetos dá-se por meio de _____, que combina _____ _ e _____. É um modo de pensar e não uma técnica de programação.

MÓDULO 2 – MODELAGEM DE CLASSES

Este módulo apresenta os principais conceitos para a elaboração da modelagem de classes: objetos, classes, atributos, operações, herança, polimorfismo, encapsulamento, ligações, associações, multiplicidade e agregação. Ao final, apresentamos um exemplo prático de um diagrama de classes relativo a um sistema bancário, que aplica os principais conceitos abordados.

1. Principais conceitos de modelagem de classes

1.1. Objetos e classes

Antes de iniciar a prática da modelagem orientada a objetos, é importante conhecer alguns conceitos que são aplicados nos diagramas UML.

Para Blaha e Rumbaugh (2006), objeto é um conceito, uma abstração ou algo com identidade que possui significado para uma aplicação. Portanto, objetos são coisas do mundo real, como pessoas, animais, carros etc.

Todos os objetos possuem identidade, conforme vimos. Um cliente é diferente de outro cliente, pois cada um possui as suas características próprias (atributos), por exemplo, o poder aquisitivo, o sexo, a altura etc. Dessa forma, a modelagem OO modela objetos do mundo real. Modelar objetos significa estudá-los e observá-los com o objetivo de representá-los dentro de um contexto.

Quando você observa o que é e o que faz um objeto em uma aplicação, você está usando a abstração. Os objetos podem ser agrupados pelos seus atributos e pelas suas operações. Por meio desses agrupamentos, podemos classificá-los. Esse procedimento é chamado classificação, no qual os objetos com a mesma estrutura de dados (atributos e operações) são agrupados em uma classe.

"Uma classe descreve um grupo de objetos com as mesmas propriedades (atributos), comportamentos (operações), tipos de relacionamentos e semântica" (BLAHA; RUMBAUGH, 2006).

Uma classe, na notação UML, é representada por um retângulo subdividido em três partes: nome da classe, atributos e operações.

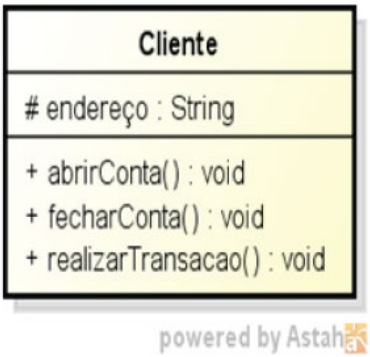

Figura 2: Classe cliente.

A figura 2 apresenta um exemplo de uma classe. *Cliente* indica o nome da classe, *#endereço* especifica o atributo da classe, que é do tipo de dado *string* (texto), e *abrirConta()*, *fecharConta()* e *realizarTransaçao()* indicam as operações da classe, que são do tipo *void* (nulo).

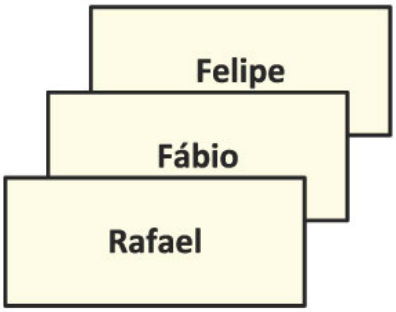

Figura 3: Instâncias (objetos) da classe cliente.

Um objeto é uma instância de uma classe. Isso significa que um cliente pode ser uma instância da classe Cliente. A figura 3 apresenta três instâncias dessa classe.

1.2. Atributos e operações

Um atributo é uma propriedade de uma classe. Os atributos são compostos por nome, tipo de dado, visibilidade e valor inicial ou padrão, cujas especificações são:

- O nome do atributo geralmente indica o seu conteúdo.
- O tipo de dado indica a organização dos conteúdos dos atributos. Por exemplo: o número da conta corrente é formado por números. Assim, o atributo número da conta corrente é do tipo "número". Os tipos de dados genéricos podem ser texto, número, lógico, data etc. Esses tipos genéricos deverão ser substituídos pelos nomes dos tipos de dados da respectiva linguagem de programação utilizada para o desenvolvimento da aplicação.
- A visibilidade de um atributo é definida como:
  - ◊ Pública (*public*): representada por um sinal de adição (+). Indica que o atributo é acessível por outras classes.
  - ◊ Privada (*private*): representada por um sinal de subtração (-). Indica que o atributo é acessível somente pela própria classe.
  - ◊ Protegida (*protected*): representada por um sinal de sustenido (#). Indica que o atributo é acessível somente pela própria classe e pelas subclasses.
  - ◊ Pacote (*package*): representada por um til (~). Indica que o atributo é acessível pelas classes do pacote que o contém.
- O valor inicial ou padrão é um valor indicado para identificar o conteúdo inicial do atributo.

As operações são funções (ações) ou comportamentos que podem ser aplicados a objetos ou por objetos em uma classe. Elas estão relacionadas aos verbos. A classe Cliente especifica três operações: abrirConta(), fecharConta() e realizarTransacao(). Todos os objetos de uma classe compartilham as mesmas operações, e a mesma operação pode se aplicar a muitas classes diferentes.

Vale aqui lembrar o conceito de polimorfismo, que significa que a mesma operação pode se comportar de forma diferente para diferentes classes. A figura 4 apresenta um exemplo disso.

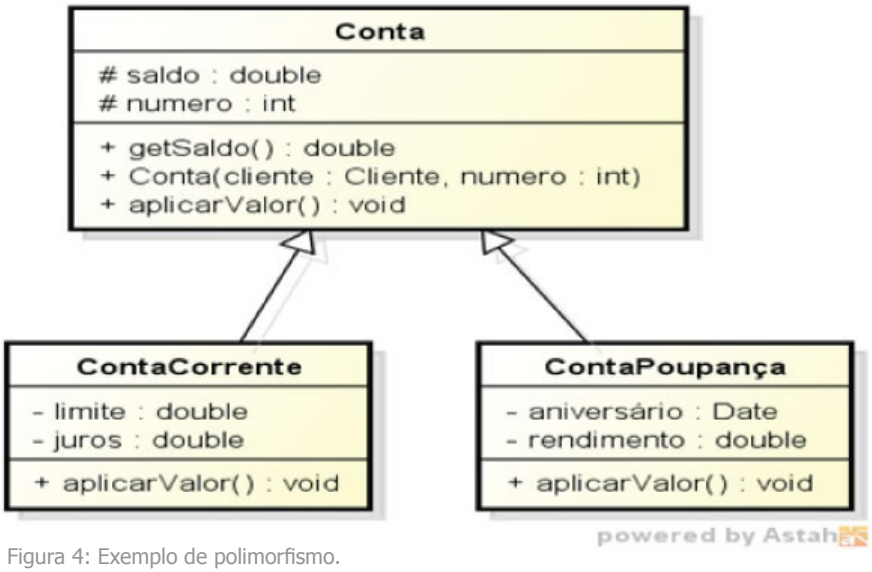

Figura 4: Exemplo de polimorfismo.

Nesse caso, a operação aplicarValor() é a mesma na superclasse Conta e nas subclasses ContaCorrente e ContaPoupança. Entretanto, a implementação e o comportamento dessa operação nas subclasses são diferentes. A diferença encontra-se no processo de cálculo dos juros e rendimentos das respectivas aplicações.

Um método ou função é a implementação de uma operação para uma classe. Isso significa que, quando o programador codifica uma operação em arquivos de códigos numa linguagem de programação, ele transformou uma operação em um método. Assim, a operação é definida pelo analista, e o método ou função, pelo programador.

As características de uma operação são: nome, lista de argumentos (parâmetros), visibilidade e tipo de retorno, conforme as especificações:

- Nome e visibilidade correspondem às mesmas definições dos atributos.
- Lista de argumentos (parâmetros) e valor de retorno dependem da linguagem de programação utilizada.

A figura 5 apresenta um resumo da notação de modelagem para classes.

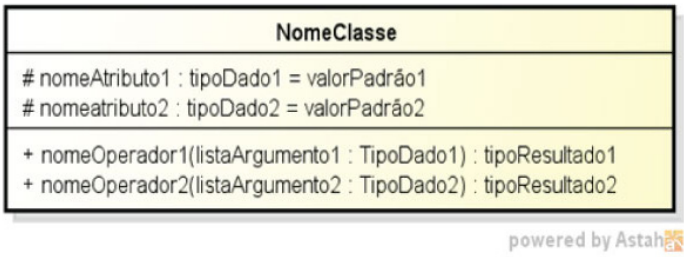

Figura 5: Resumo da notação de modelagem para classes. Fonte: Adaptado de Blaha e Rumbaugh, 2006.

1.3. Generalização e herança

O conceito de generalização é definido por Blaha e Rumbaugh (2006) como sendo o relacionamento entre uma classe (superclasse) e uma ou mais variações dela (subclasses). A generalização organiza as classes por suas semelhanças e diferenças.

Para entender melhor o conceito, observe a figura 6.

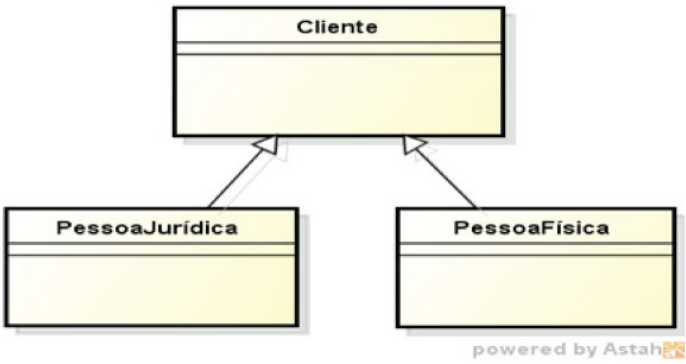

Figura 6: Generalização.

A figura mostra um relacionamento entre uma classe mais geral (superclasse ou classe pai) e uma mais específica (subclasse ou classe filho). Assim, a superclasse generaliza as subclasses, e estas herdam os recursos (atributos e operações) daquela.

Os relacionamentos entre a superclasse e as subclasses devem ser indicados por setas com uma ponta “aberta”.

Com base no conceito de generalização, observe agora o exemplo da figura 7, que apresenta as especificações da superclasse e das subclasses com os seus respectivos atributos e operações.

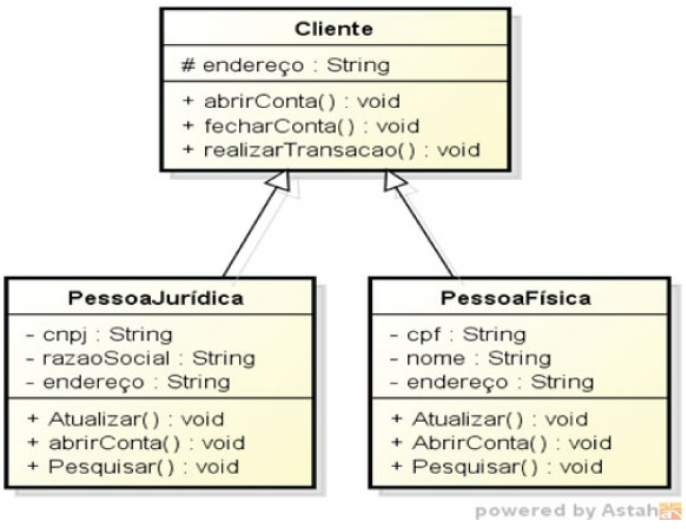

Figura 7: Herança.

Observe que as subclasses PessoaJurídica e PessoaFísica herdam os atributos e as operações da superclasse Cliente. Vale ressaltar que as subclasses podem ter outros atributos e operações específicos. Dessa forma, a figura 7 é um exemplo de herança. Herança é a capacidade de uma ou mais subclasses herdarem os atributos e as operações da superclasse por meio do relacionamento de generalização.

1.4. Encapsulamento

Segundo Lima (2011),

> encapsulamento ou ocultação de informações é uma técnica que consiste em separar aspectos externos dos internos da implementação de um objeto, isto é, determinados detalhes ficam ocultos aos demais objetos e dizem respeito apenas ao próprio objeto.

A técnica consiste no ocultamento das informações, ou seja, qualquer aplicação poderá utilizar uma determinada classe, mas não terá acesso à forma como foi definida. Os atributos e as operações encapsulados somente serão visíveis pelo próprio objeto.

Vale ressaltar que, nessa técnica, os objetos da subclasse herdam todos os atributos e operações da superclasse. Isso significa que o encapsulamento estabelece uma dependência com a relação de herança.

1.5. Ligações, associações e multiplicidade

De acordo com Blaha e Rumbaugh (2006),

> ligações e associações são o meio de estabelecer relacionamentos entre objetos e classes. Uma ligação é uma conexão física ou conceitual entre objetos e associação é uma descrição de um grupo de ligações com estrutura e semântica comuns.

Assim, os objetos e as classes relacionam-se por meio de uma notação específica da UML. A figura 8 indica uma ligação entre duas classes.

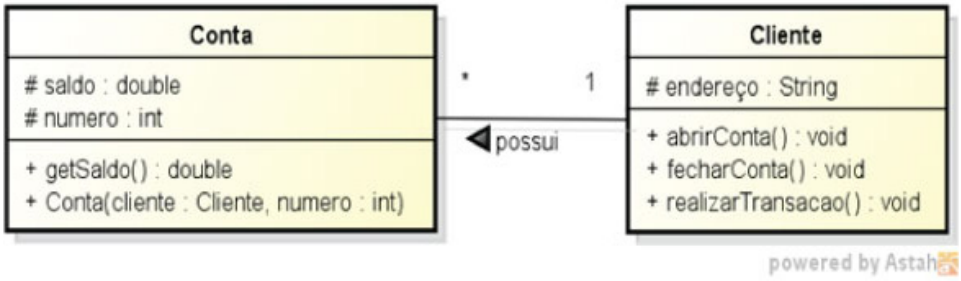

Figura 8: Ligação entre classes.

As classes *Cliente* e *Conta* relacionam-se entre si. A linha que une as duas é chamada de ligação. No exemplo, a classe Cliente possui a classe Conta. As ligações e as associações normalmente são indicadas por meio de verbos, conforme indicado na ligação da figura 8.

Além disso, há uma notação da UML chamada de multiplicidade. É uma simbologia utilizada em ligações e associações. Para Blaha e Rumbaugh (2006), "multiplicidade especifica o número de instâncias de uma classe que podem se relacionar a uma única instância de uma classe associada".

No exemplo da figura 8, são utilizados os seguintes símbolos: o número 1 e o caractere asterisco (*). O símbolo (1) indica o próprio valor numérico "exatamente um", e o símbolo (*) indica "muitos" (zero ou mais). Logo, lê-se o seguinte: "um para muitos" ou "um cliente para muitas contas". Blaha e Rumbaugh (2006) advertem em sua obra que, apesar de a literatura utilizar esse significado, a multiplicidade é um subconjunto (possivelmente infinito) de inteiros não negativos.

Além da simbologia apresentada no exemplo, a multiplicidade permite outras possibilidades de simbologias, tais como: "1.." – um ou mais; "5..7" – cinco a sete inclusive; "0..1" – zero ou um; etc. A indicação de notação de multiplicidade é considerada opcional. Portanto, se você não indicá-la, não haverá problemas, mas é recomendável usá-la.

No exemplo da figura 9, encontra-se uma associação entre três classes: Conta, Cliente e Transação.

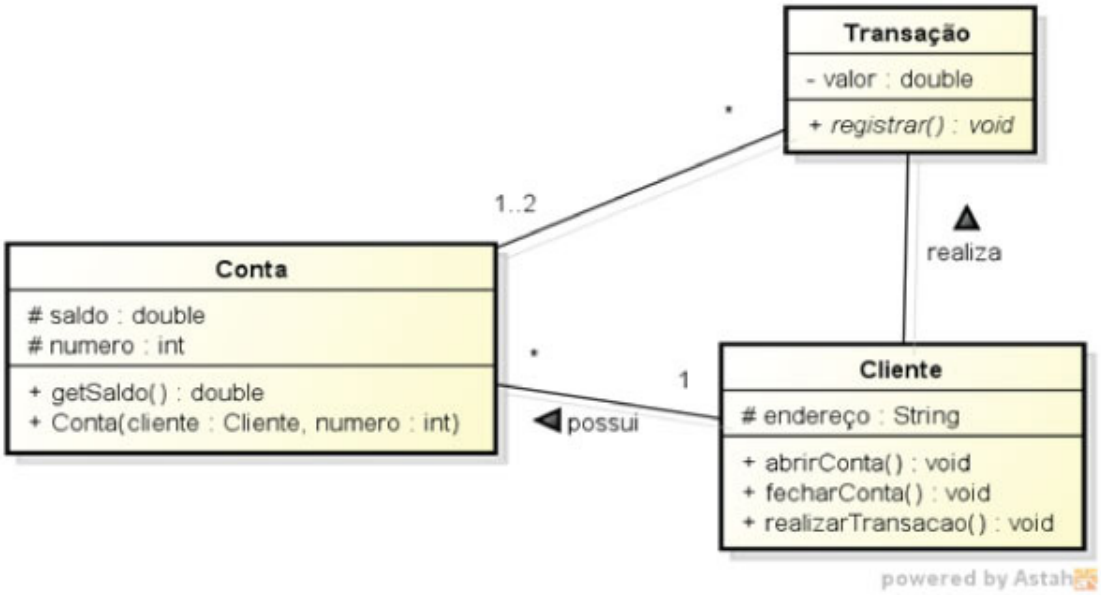

Figura 9: Associação entre classes.

A ligação entre as classes Conta e Cliente é a mesma do exemplo da figura anterior. No caso daquela entre as classes Conta e Transação, a multiplicidade é de "uma ou duas contas inclusive (1..2)" para "muitas transações (*)" e não foi indicado o verbo. Na ligação entre Transação e Cliente, não há notação de multiplicidade, mas consta a indicação do verbo "realiza", ou seja, "Cliente realiza Transação".

Como você pode notar, essa associação mostra um exemplo prático da relação entre três classes no mundo real. Num sistema bancário, há relação do cliente com as suas contas e transações, além da relação das contas com as suas respectivas transações.

Dessa forma torna-se mais fácil entender as relações de dependência entre as classes, o que possibilita melhor documentação de todos os processos do sistema.

1.6. Agregação e composição

"A agregação é uma forma especial de associação utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação de todo/parte", afirma Furlan (1998). Assim, agregação é uma associação em que um objeto é parte do outro, ainda que a parte possa existir sem o todo.

A figura 10 apresenta um exemplo simples de agregação.

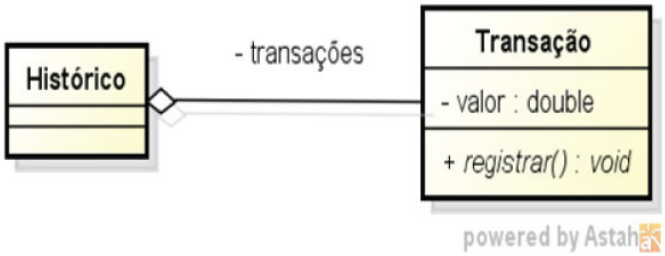

Figura 10: Agregação.

Nesse caso, notam-se duas classes: Transação e Histórico. A primeira refere-se às transações financeiras (transferência entre contas, pagamento de contas etc.) realizadas em um banco, e a segunda, ao histórico das respectivas transações bancárias.

Na ligação entre as classes, observa-se a forma de losango vazio (aberto) na extremidade da classe Histórico. Esse símbolo representa uma agregação, o que significa que essa classe é um agregado da classe Transação, ou seja, as transações financeiras (partes – objeto constituinte) estão descritas no histórico (todo – objeto agregado).

De acordo com Medeiros (2004), para sabermos se um relacionamento é de agregação, basta perguntarmos

se, em primeiro lugar, cabe a estrutura todo-parte. Em seguida, perguntamos se o objeto constituinte "vive" sem o objeto agregado. Se a resposta for sim para ambas as perguntas, teremos uma agregação.

Outro caso especial de associação, semelhante à agregação, é a composição, que é também uma forma de relacionamento todo/parte. A composição é constituída por partes (componentes) que formam o todo (composto), de maneira que o todo não existe sem as suas partes. A figura 11 é um exemplo de composição.

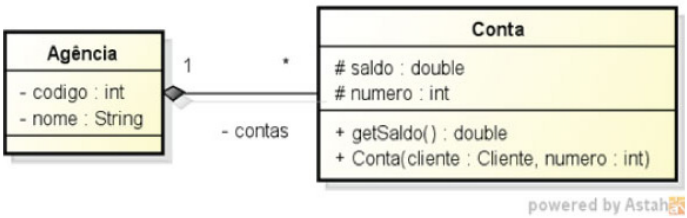

Figura 11: Composição.

A classe Agência (todo) é composta pela classe Conta (partes). Você deve ter observado que a notação UML para a composição é muito semelhante à para a agregação. O símbolo é o mesmo, entretanto, o losango é preenchido (cheio) na cor preta.

Para identificar uma composição, utilizaremos as mesmas perguntas utilizadas para identificar uma agregação sugeridas por Medeiros (2004). A diferença é que, se a resposta à segunda pergunta for não, você terá uma composição.

2. Diagrama de classe

2.1. Representação de um diagrama de classe

Como você percebeu, todas as figuras apresentadas na aula passada utilizaram-se de um mesmo assunto para exemplificar os principais conceitos sobre a modelagem de classes, ou seja, de um sistema bancário. Esse sistema não é complexo e tem finalidade didática para este material. Assim, esta aula e as demais são apresentadas com base nele para facilitar seu aprendizado.

Um diagrama de classe é representado por um conjunto de classes relacionadas entre si. É um dos diagramas mais importantes da modelagem orientada a objetos.

A figura 12 apresenta um diagrama de classes de um sistema bancário. Veja que algumas das classes utilizadas já foram apresentadas na aula passada, com os seus respectivos conceitos.

Observe atentamente cada parte do diagrama de classe e identifique:

- As características de uma classe (atributos, operações etc.).
- As associações e sua multiplicidade.
- O polimorfismo.
- A diferença entre generalização e herança.
- A diferença entre agregação e composição.
- O contexto do sistema bancário como um todo no mundo real.

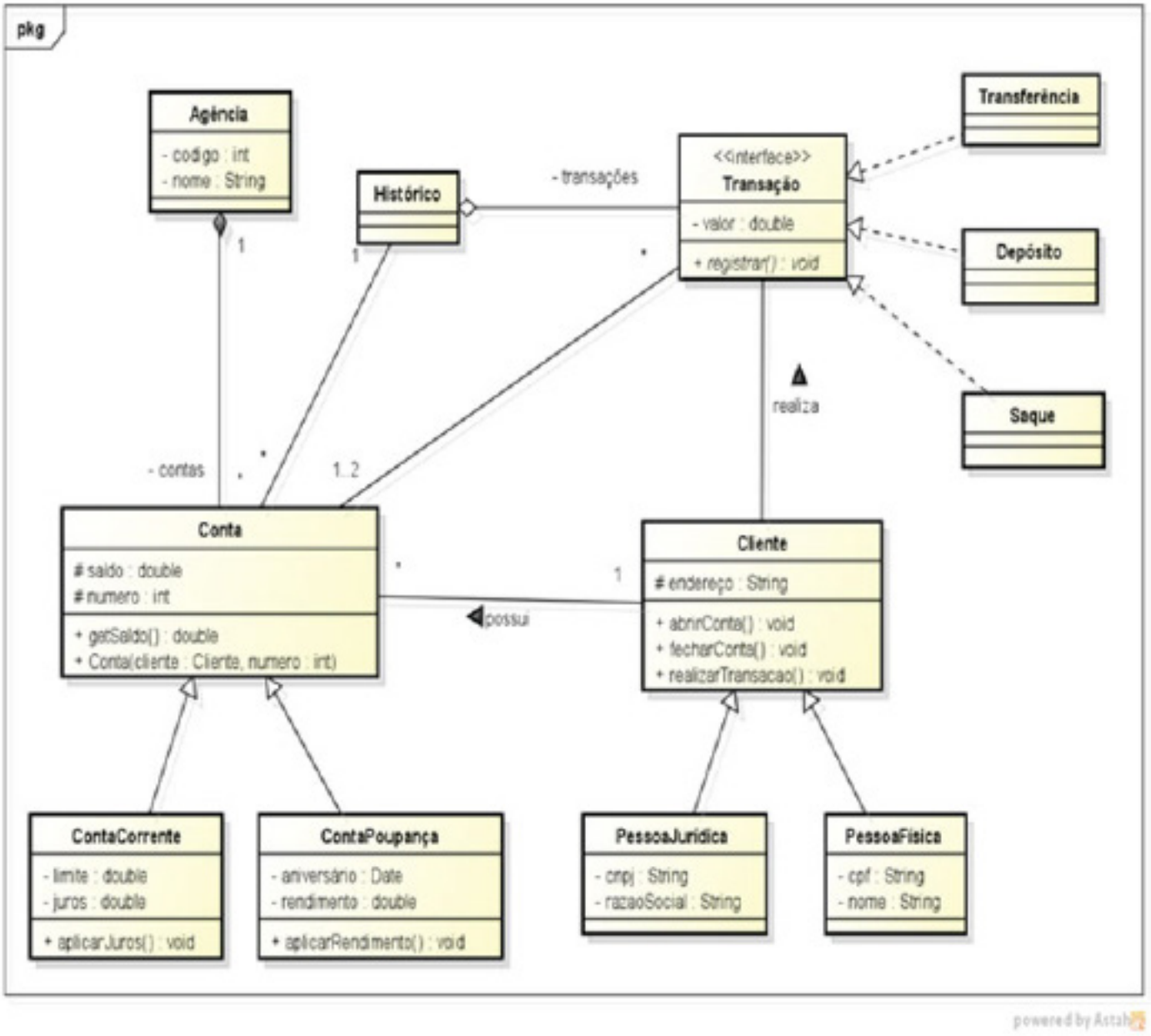

Figura 12: Diagrama de classes do sistema bancário.

Você deve ter notado dois substantivos indicados no diagrama de classe: o primeiro, chamado “contas”, indicado na composição, e o segundo, chamado “transações”, indicado na agregação. Esses nomes indicados nas extremidades de uma associação indicam os nomes de papéis. *Nome de papel* é um nome descritivo que indica o papel que a classe possui no relacionamento, e o seu uso é opcional.

Outro detalhe importante é a indicação de <<interface>> em Transação. Os símbolos “<<” e “>>” são chamados de aspas francesas e indicam estereótipos, que são mecanismos de extensibilidade da UML. Eles são predefinidos na UML ou criados pelo próprio analista.

Nesse caso, o estereótipo <<interface>> representa uma interface e não uma classe. Assim, ele foi especificado como uma notação para indicar essa diferença.

A interface indica um comportamento obrigatório da classe e, geralmente, esconde alguns detalhes, por exemplo, os de implementação.

Observe que a interface Transação relaciona-se com outras três classes: Transferência, Depósito e Saque. O relacionamento entre a interface e as classes é indicado por meio de setas vazias (brancas) tracejadas. Isso representa um relacionamento do tipo “realização”. Assim, há um contrato de implementação estabelecido

entre a interface Transação e as classes Transferência, Depósito e Saque.

Caro aluno, assista à animação 2, que ilustra a elaboração de um diagrama de classes, realizada por um analista, a respeito de algumas transações financeiras no Banco UML.

3. Exercícios do módulo 2

1) Um diagrama de classes é representado por um conjunto de classes com os seus respectivos atributos e operações.

a. Verdadeiro.

b. Falso.

2) Segundo Blaha e Rumbaugh (2006), uma classe descreve um grupo de objetos com as mesmas propriedades (atributos), comportamentos (operações), tipos de relacionamentos e semântica.

a. Verdadeiro

b. Falso

3) Em relação ao polimorfismo, podemos afirmar que:

a. A mesma operação pode se comportar de forma diferente para todas as classes.

b. A mesma operação pode se comportar de forma igual para diferentes classes.

c. A mesma operação pode se comportar de forma diferente para diferentes classes.

d. A mesma operação pode se comportar de forma igual para todas as classes.

4) Quais são os tipos de visibilidade de um atributo?

a. Pública e privada.

b. Pública, privada e protegida.

c. Pública, privada, protegida e pacote.

d. Pública, privada, protegida, pacote e oculta.

5) Podemos dizer que a composição é:

a. Um caso especial de associação e uma forma de relacionamento todo/parte.

b. Formada por um losango aberto.

c. É um relacionamento todo/parte, entretanto a parte poderá existir sem o todo.

d. Não é um relacionamento todo/parte, pois a composição é constituída por componentes que formam o composto.

6) Encapsulamento é o mesmo que:

a. Herança, ou seja, estabelece uma relação de ocultamento de informação.

b. Ocultação de informação.

c. Generalização, pois as informações são herdadas da superclasse para a subclasse.

d. Ocultar determinados detalhes da implementação.

7) Multiplicidade é uma notação _____. É uma simbologia utilizada em _____ e _____. Ela especifica o _____ de instâncias de uma _____ que podem se relacionar a uma _____ instância de uma classe associada (BLAHA; RUMBAUGH, 2006).

8) Herança é a capacidade de _____ ou mais _____ herdarem os _____ e as _____ da superclasse por meio do relacionamento de _____.

MÓDULO 3 – MODELAGEM DE INTERAÇÕES

Neste módulo, são apresentados três diagramas da modelagem de interação: o diagrama de caso de uso, o diagrama de sequência e o diagrama de comunicação. Todos foram elaborados com base no exemplo do sistema bancário.

1. Diagrama de caso de uso

1.1. Representação de um diagrama de caso de uso

O diagrama de caso de uso descreve as funcionalidades do sistema, demonstrando as interações externas. Segundo o OMG (2013), um caso de uso é a especificação de um conjunto de ações executado tipicamente por um sistema que entrega um resultado observável que é de valor para um ou mais atores ou outros interessados no sistema.

Esse diagrama permite a coleta dos requisitos comportamentais do sistema. Para Cockburn (2005), toda organização captura requisitos para satisfazer suas necessidades. Há padrões disponíveis para descrições desses requisitos e, em qualquer deles, casos de uso ocupam somente uma parte do total. Eles não são todos os requisitos – são apenas os requisitos comportamentais, contudo, são todos os comportamentais.

Um diagrama de caso de uso é representado por símbolos, conforme a figura 13.

Figura 13: Símbolos do diagrama de caso de uso.

Os símbolos utilizados são o ator, representado pelo "homem palito", o caso de uso, representado por uma elipse, e os relacionamentos, que, nesse caso, são representados por uma linha

O ator representa uma entidade externa que interage com o sistema. Ele não faz parte deste. O ator pode ser representado por pessoas, *hardware* ou sistemas.

A elaboração de um diagrama de caso de uso deve iniciar-se pela identificação dos atores. O analista deve questionar quais são os grupos de usuários, equipamentos ou sistemas que interagirão com o sistema. Em seguida, deverá indicar as responsabilidades de cada ator, ou seja, encontrar as funções deles em relação ao sistema. Por exemplo: podemos afirmar que uma das funções de um funcionário num banco é "abrir conta". Assim, o funcionário (ator) tem a responsabilidade de "abrir conta" (caso de uso) em um sistema bancário.

Em relação ao caso de uso, é necessário identificar as funcionalidades do sistema em relação às interações com os atores. As funções indicadas nos casos de uso são ações que estes desempenham no sistema. Geralmente, utiliza-se um verbo no infinitivo acompanhado de um substantivo. Por exemplo: abrir conta, efetuar saque, atualizar histórico, realizar pagamento etc. poderiam ser casos de uso de um sistema bancário.

O ator deve interagir com o sistema por meio de relacionamentos. A comunicação ocorre por meio de linhas com ou sem setas. A sem setas indica um relacionamento bidirecional ou de duas direções. Vale ressaltar que as linhas utilizadas para indicar relacionamentos em diagramas de caso de uso não são fluxo de dados.

Os diagramas de caso de uso devem ser elaborados após análise e estudo aprofundados sobre o funcionamento do sistema a ser desenvolvido. Na próxima seção, esse assunto é tratado com mais detalhes.

Blaha e Rumbaugh (2006) relacionam os seguintes princípios para a construção de diagramas de casos de uso:

- Em primeiro lugar, determine os limites do sistema.
- Tenha certeza de que os atores estão focalizados. Cada um deles deve ter um propósito único e coerente.
- Cada caso de uso deve fornecer valor aos usuários.
- Relacione casos de uso e atores. Cada caso de uso deve ter pelo menos um ator, e cada ator deve participar de pelo menos um caso de uso. Um caso de uso pode envolver vários atores, e um ator pode participar de vários casos de uso.
- Lembre-se de que os casos de uso são informais. É importante não exagerar no formalismo ao especificá-los.
- Os casos de uso podem ser estruturados. Para muitas aplicações, os casos de uso individuais são completamente distintos. Para grandes sistemas, eles podem ser construídos de fragmentos menores usando relações.

As próximas figuras especificam diagramas de casos de uso de um sistema bancário. Na figura 14, é apresentado o cenário de um cliente efetuando algumas transações no sistema bancário, e, na figura 15, um funcionário do banco realiza manutenção de contas do cliente.

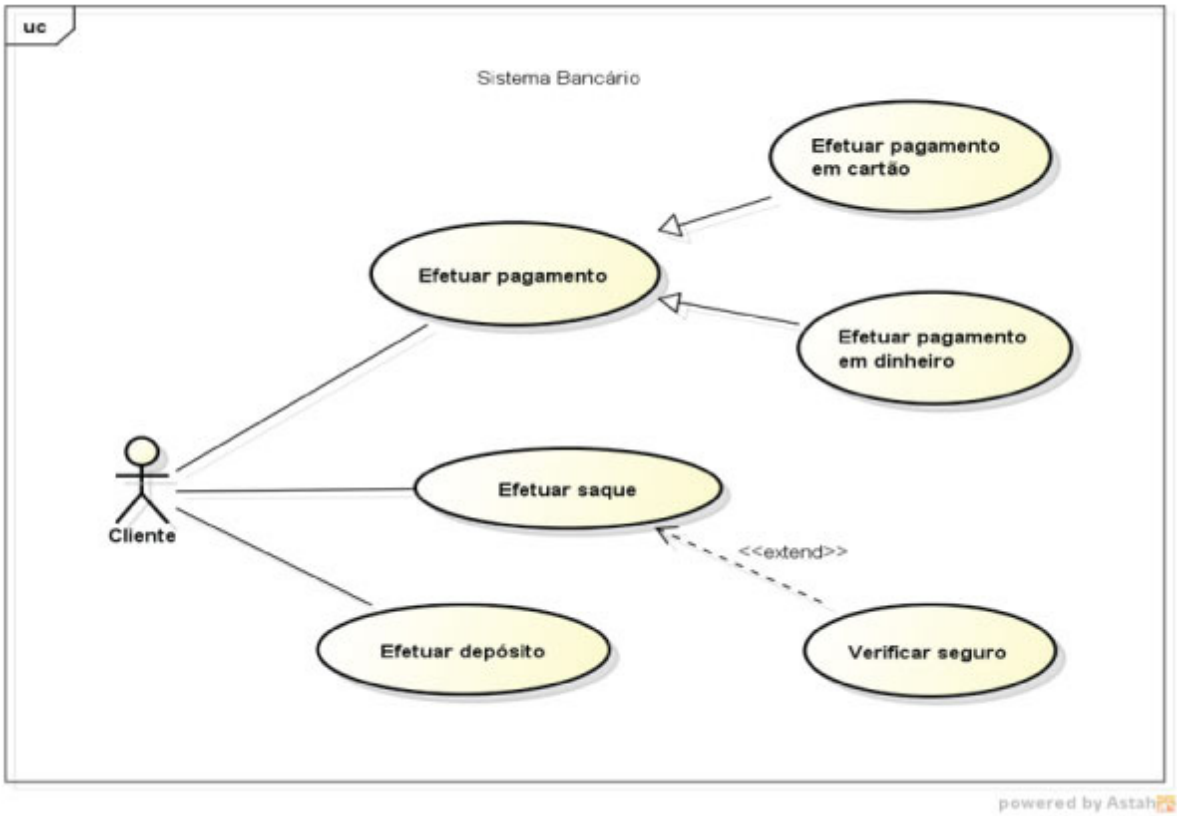

Figura 14: Diagrama de caso de uso: transação bancária.

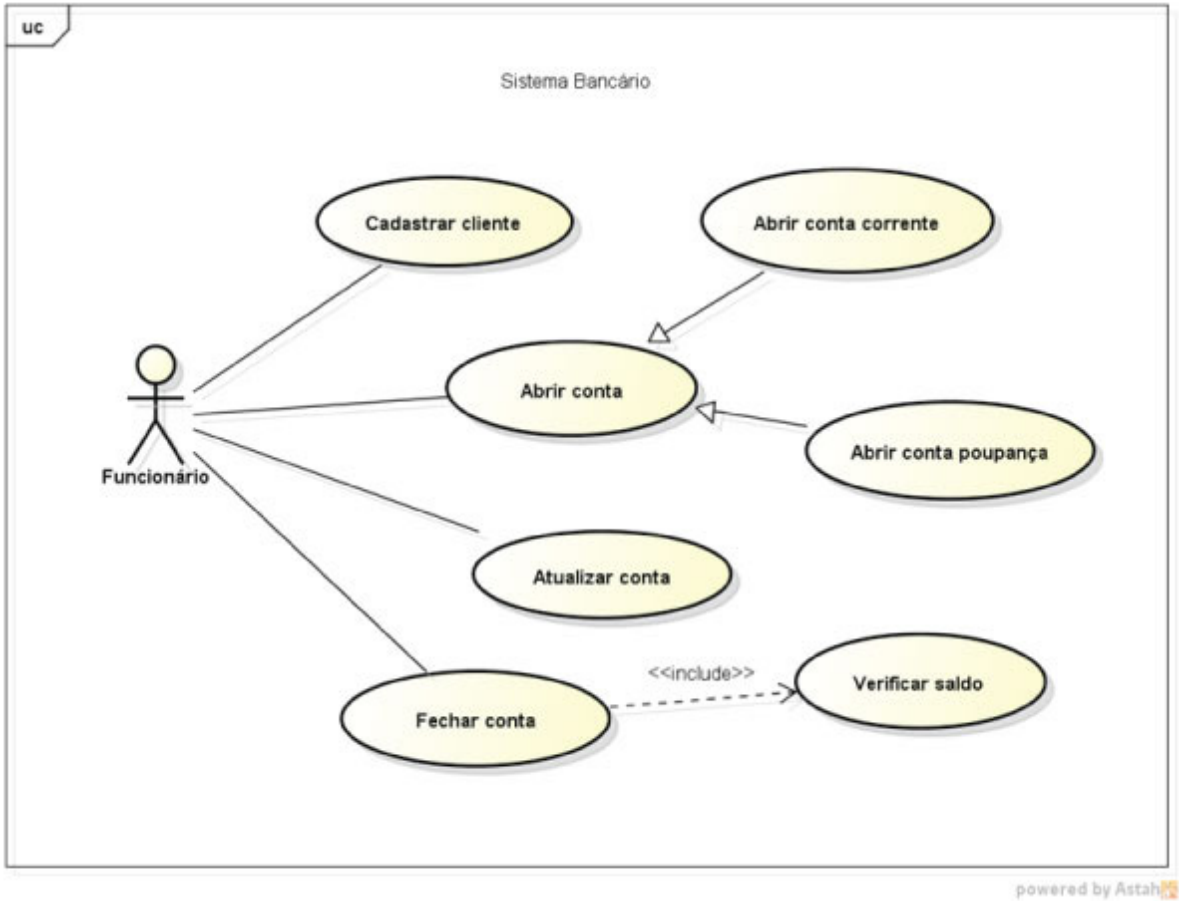

Figura 15: Diagrama de caso de uso: manutenção de conta bancária.

Você deve ter notado que existem alguns casos especiais de relacionamentos indicados nas figuras acima. O primeiro é chamado de relacionamento de extensão <<*extend*>>. No exemplo, o caso de uso Efetuar saque é estendido pelo caso de uso Verificar seguro. Isso significa que o primeiro pode ser acrescentado do segundo.

No segundo caso, encontra-se o relacionamento de inclusão <<*include*>>. No exemplo, o caso de uso Fechar conta conterá o comportamento do caso de uso Verificar saldo. Isso significa que toda vez que ocorrer o fechamento de uma conta ocorrerá a verificação de saldo dela.

Os casos especiais de relacionamentos <<*extend*>> e <<*include*>> são estereótipos. Esse conceito já foi tratado na aula passada.

Em ambas as figuras, são apresentados exemplos de generalização. Na figura 14, os casos de uso Efetuar pagamento em cartão e Efetuar pagamento em dinheiro compartilham a estrutura e o comportamento do caso de uso Efetuar pagamento. O mesmo processo ocorre na figura 15, com os casos de uso Abrir conta, Abrir conta corrente e Abrir conta poupança.

1.2. Estudo para elaboração de um diagrama de caso de uso

Para elaborar um diagrama de caso de uso, é necessário um estudo detalhado sobre cada parte do funcionamento do sistema. O analista deve criar uma documentação própria para especificar cada um dos detalhes. Como sugestão, podem-se criar tabelas ou formulários em que as identificações estarão documentadas.

Na tabela 2 encontram-se alguns exemplos desta documentação. Os exemplos são baseados nas figuras 14 e 15. Para início do diagrama de caso de uso, é necessário identificar os atores e os seus respectivos casos de uso. O analista deverá identificá-los por meio de algumas questões, tais como:

- Quem são os usuários que irão interagir com o sistema?
- Quais são as suas ações em relação ao sistema?

Tabela 2: Identificação de atores e casos de uso.

Atores	Principais casos de uso
Cliente	Efetuar pagamento.
	Efetuar saque.
	Efetuar depósito.
Funcionário	Cadastrar cliente.
	Abrir conta.
	Fechar conta.
	Atualizar conta.

Outra pergunta que poderá ser feita pelo analista é sobre as ações de cada um dos atores com relação aos seus respectivos casos de uso. A tabela 3 apresenta os resultados do ator Cliente em relação ao caso de uso Efetuar saque.

Tabela 3: Ações lógicas realizadas pelo cliente ao efetuar um saque num terminal bancário.

Ator	Caso de uso	Ações
Cliente	Efetuar saque	Inserir cartão no terminal.
		Digitar a senha.
		Escolher a opção “saque”.
		Digitar ou escolher o valor.
		Retirar o dinheiro.
		Escolher a opção “emitir comprovante”.
		Retirar o comprovante.

Além dessas ações, o analista deverá prever as exceções que poderão ocorrer ao se efetuar saque no terminal bancário. A tabela 4 indica quais são elas.

Tabela 4: Possibilidades de exceção ao Efetuar saque no terminal pelo cliente.

Ator	Caso de uso	Ações	Possibilidades de exceção
Cliente	Efetuar saque	Inserir cartão no terminal.	Cartão recusado.
		Digitar a senha.	Senha inválida.
		Escolher a opção. “saque”	Opção inválida.
		Digitar ou escolher o valor.	Erro de digitação ou opção inválida.
		Retirar o dinheiro.	Falta de dinheiro.
		Escolher a opção de “emitir comprovante”.	Opção inválida.
		Retirar o comprovante.	Falta de papel.

O analista deve analisar cada uma dessas possibilidades de exceção e tomar medidas cabíveis para cada caso. Por exemplo, no caso de senha inválida, o cliente terá a possibilidade de digitá-la novamente no sistema.

2. Diagrama de sequência e diagrama de comunicação

2.1. Representação do diagrama de sequência

Além do diagrama de caso de uso, o diagrama de sequência também é considerado um modelo de interação. Entretanto, eles apresentam informações diferentes. O diagrama de caso de uso apresenta a interação dos atores externos com o sistema, e o diagrama de sequência apresenta como ocorre o fluxo de mensagens entre os objetos ao longo do tempo sobre um determinado caso de uso.

Assim, diagrama de sequência é um diagrama que apresenta, numa ordem lógica, os fluxos de mensagens entre os objetos pertinentes ao respectivo caso de uso num determinado período de tempo. Isso significa que ele deve ser elaborado após o desenvolvimento do diagrama de caso de uso.

É necessário conhecer os símbolos utilizados para a elaboração de um diagrama de sequência. Ele é composto por objetos e fluxos de mensagens.

Os objetos são desenhados como um retângulo ao topo de uma linha vertical tracejada e projetada para baixo, que recebe o nome de linha de vida. Além deles, geralmente, uma instância de ator é desenhada como a primeira linha de vida do diagrama de sequência.

As mensagens são indicadas nessa linha. Ao indicar uma mensagem, uma base retangular vertical é criada sobre a linha de vida. Essa base é chamada de ativação e determina a execução de uma ação, num período de tempo da interação, entre os objetos.

Os fluxos de mensagens são identificados por um número, que indica a sequência delas. Eles podem ser do tipo de procedimento síncrono ou assíncrono.

O fluxo do tipo síncrono é desenhado com a seta do fluxo de forma sólida (preenchida) na cor preta e indica que o objeto remetente esperará um retorno da mensagem pelo objeto destinatário.

O fluxo do tipo assíncrono é desenhado com a seta do fluxo de forma não sólida e indica que as mensagens são enviadas, mas não se espera o retorno de forma imediata.

A figura 16 indica os principais símbolos do diagrama de sequência.

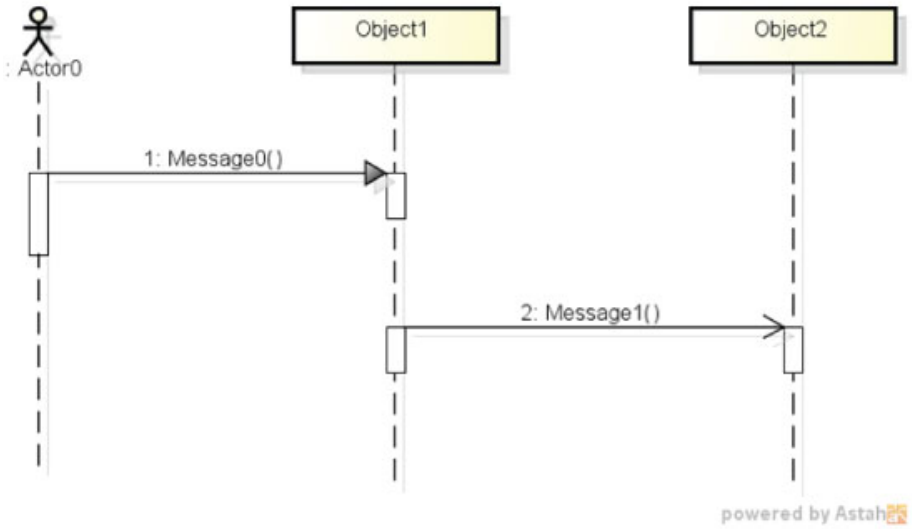

Figura 16: Símbolos do diagrama de sequência.

É possível observar que a elaboração do diagrama de sequência da figura 17 foi iniciada por um ator (*Actor0*) e, na sequência, foram indicados dois objetos: o *Object1* e o *Object2*. Na interação entre o ator e o primeiro objeto, encontra-se uma mensagem síncrona, e, na interação entre os dois objetos, uma mensagem assíncrona.

No diagrama de sequência, também podem ser encontrados alguns casos especiais, como autodelegação e retorno automático de mensagem.

Autodelegação ou autochamada, de acordo com Furlan (1998), é uma técnica utilizada em algoritmos para mostrar que uma operação chama a si própria. Na prática, isso significa que a mensagem é enviada para o próprio objeto. A mensagem de autodelegação é sempre síncrona.

Uma outra forma de representar as mensagens é indicar o fluxo por uma linha tracejada, que significa um retorno automático da mensagem. Assim, a mensagem é enviada para um destinatário e, em seguida, obtém-se o retorno esperado. Observe esses casos especiais na figura 17.

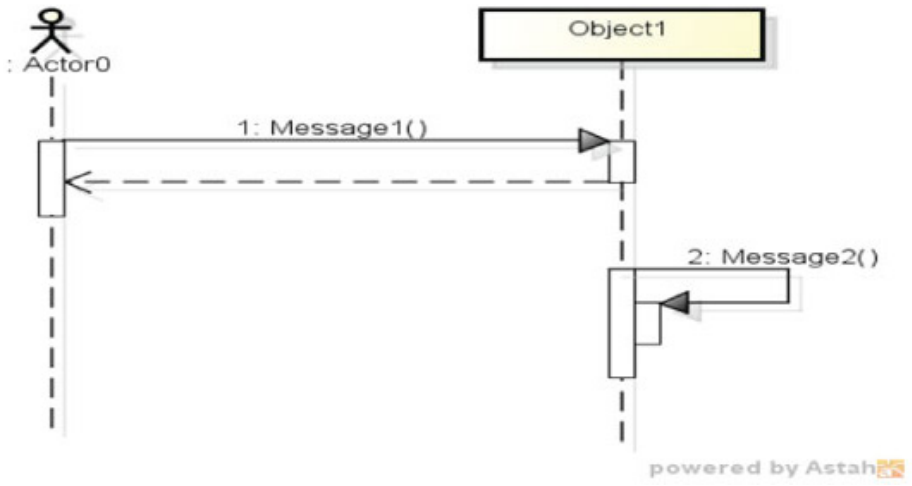

Figura 17: Retorno automático de mensagens e autodelegação.

Agora que você já conhece os principais conceitos sobre o diagrama de sequência, vamos elaborar um exemplo prático?

A tabela 3, descrita anteriormente, especifica as ações lógicas do caso de uso Efetuar saque. Conforme já foi dito, é necessário tomar como base os casos de uso representados no diagrama de caso de uso para elaborar o diagrama de sequência. Vamos, então, dar continuidade aos exemplos que usamos até o momento relembrando o que foi descrito nessas ações lógicas:

1. Inserir o cartão no terminal.
2. Digitar a senha.
3. Escolher a opção “saque”.
4. Digitar ou escolher o valor.
5. Retirar o dinheiro.
6. Escolher a opção de “emitir comprovante”.
7. Retirar o comprovante.

Essas ações lógicas não necessitam ser representadas exatamente da mesma forma no diagrama de sequência. Elas são apenas uma forma de facilitar o entendimento do que um cliente necessita fazer para efetuar o saque de um valor num terminal bancário.

Observe atentamente a figura 18. Analise cada uma das interações e imagine o caso de uso Efetuar saque no mundo real. Vale ressaltar que, para a realização de um processo na realidade, há possibilidade de uma série de variáveis e, muitas vezes, não é possível a representação de todas elas num único diagrama de sequência.

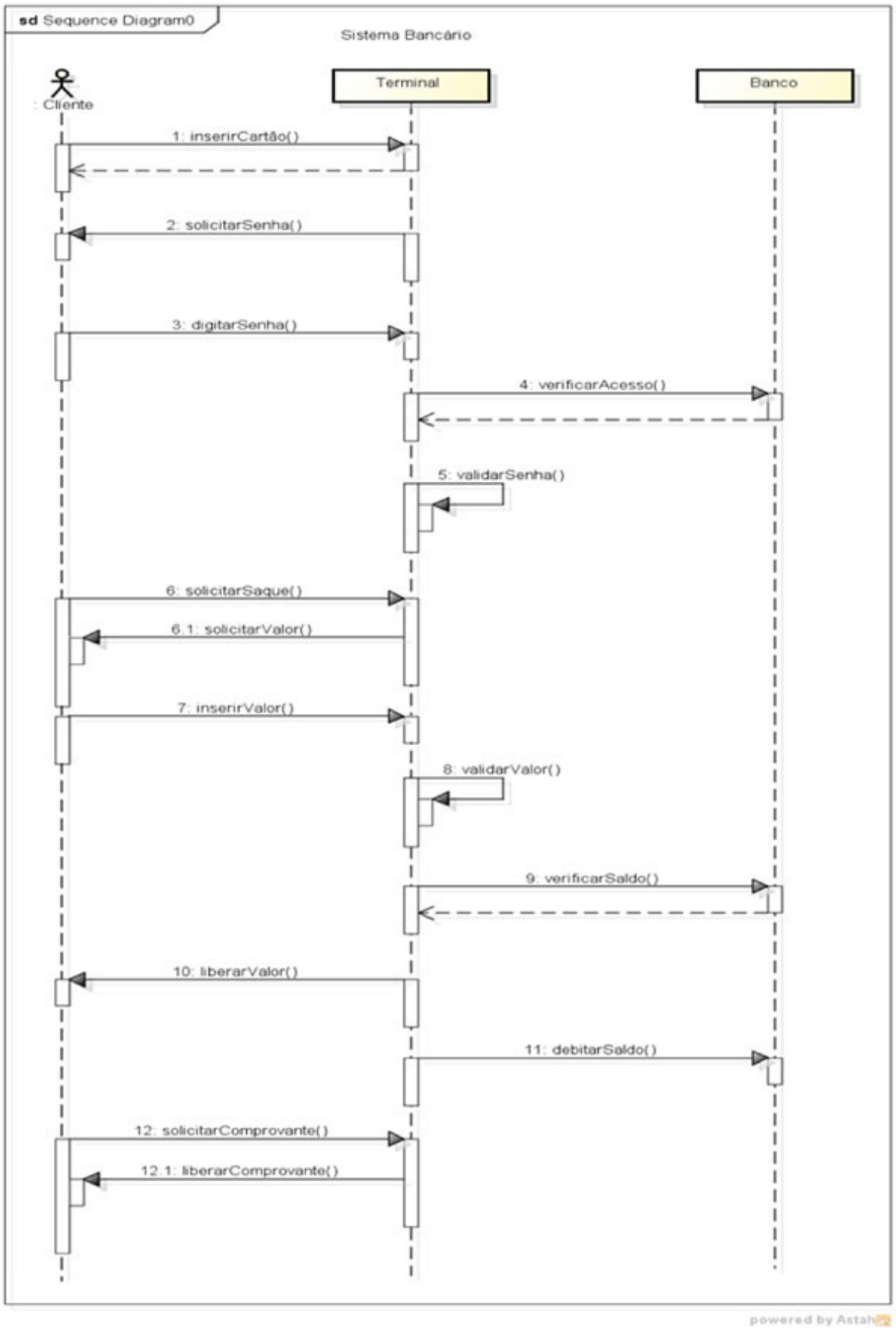

Figura 18: Diagrama de sequência: caso de uso Efetuar saque.

Outra observação importante é que o diagrama de sequência representado na figura 18 está muito extenso. É possível elaborar um para as mensagens relacionadas às ações sobre validação de senha e outro para as relacionadas à realização de saque, por exemplo. O objetivo de essas ações lógicas estarem em um único diagrama de sequência está relacionado a questões didáticas aplicadas ao contexto prático dos exemplos.

2.2. Representação do diagrama de comunicação

O diagrama de comunicação (ou diagrama de colaboração, como era denominado até a versão UML 1.5), como o diagrama de sequência, apresenta um fluxo de mensagens, porém não em sequência. Além disso, a disposição dos objetos é estrutural. O diagrama de comunicação deve ser elaborado com o intuito de mostrar os relacionamentos entre os objetos. É um complemento do diagrama de sequência.

O processo de elaboração do diagrama de comunicação é realizado da mesma forma que o do diagrama de sequência. Assim, ele é elaborado com base nas ações lógicas descritas em determinados casos de uso especificados no diagrama de casos de uso.

Os símbolos utilizados no diagrama de comunicação são: objetos representados por retângulos e mensagens numeradas representadas por setas e linhas, para mostrar os relacionamentos entre os objetos. É importante ressaltar que essa linha representa uma mensagem ou várias entre os relacionamentos.

Os fluxos de mensagens utilizados são síncronos ou assíncronos, conforme já explicado. Entretanto, os assíncronos não são utilizados com muita frequência, por o diagrama de comunicação não especificar as sequências deles.

A figura 19 ilustra o diagrama de comunicação elaborado com base nas ações lógicas do caso de uso Efetuar saque. Observe que esse diagrama de comunicação é muito simples. Isso porque ele não deve ser complexo, mas deve apresentar os relacionamentos entre os objetos e as mensagens indicadas de forma objetiva e precisa. A leitura do diagrama de comunicação deve ser realizada com base no diagrama de caso de uso e no seu complemento, o diagrama de sequência.

Geralmente, o diagrama de comunicação é elaborado para os casos de uso que apresentam muitos objetos, de forma a esclarecer melhor os relacionamentos entre eles.

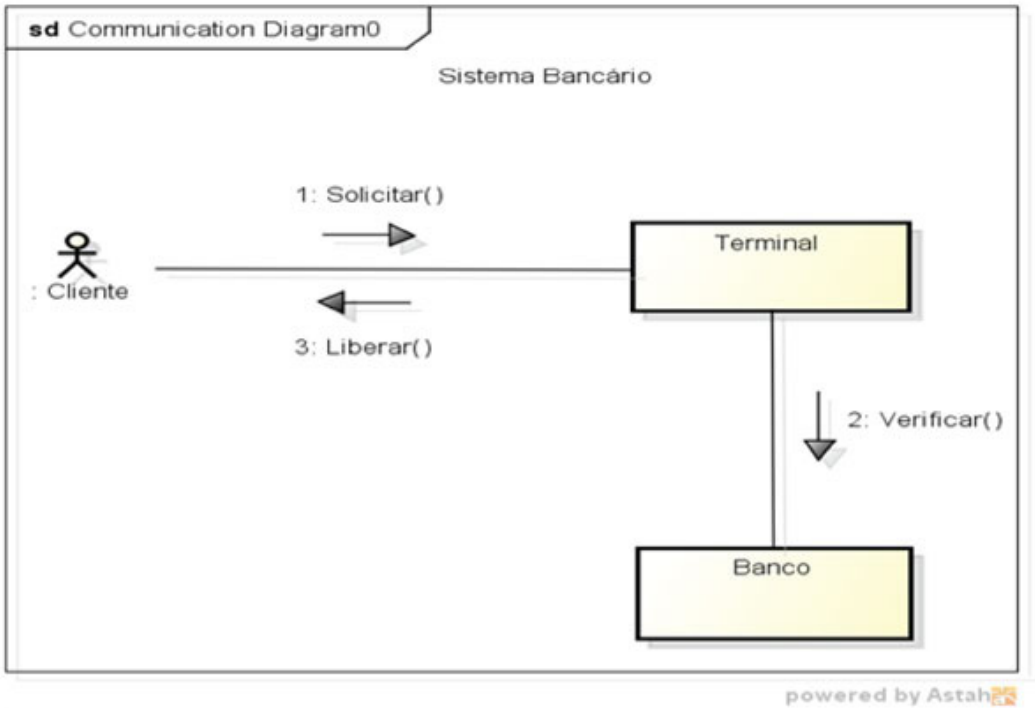

Figura 19: Diagrama de comunicação: caso de uso Efetuar saque.

3. Exercícios do Módulo 3

1) Na mais recente versão da UML, o diagrama de colaboração é chamado de diagrama de comunicação.

a. Verdadeiro

b. Falso

2) O fluxo de mensagens assíncrono requer um retorno do remetente.

a. Verdadeiro

b. Falso

3) <<include>> e <<extend>> são estereótipos utilizados no diagrama de caso de uso.

a. Verdadeiro

b. Falso

4) Autodelegação é uma técnica utilizada em algoritmos para mostrar que uma operação chama a si própria.

a. Verdadeiro

b. Falso

5) Faça as respectivas associações:

1. Diagrama de caso de uso.
2. Diagrama de sequência.
3. Diagrama de comunicação.

a. ( ) Mostrar os relacionamentos entre objetos.

b. ( ) Mostrar como ocorre o fluxo de mensagens entre os objetos ao longo do tempo.

c. ( ) Mostrar a funcionalidade do sistema.

6) O modelo de _____ descreve as interações dentro de um _____. Ele descreve como os _____ interagem para produzir resultados úteis.

7) Um caso de _____ é a especificação de um conjunto de _____ executado tipicamente por um _____ que entrega um _____ observável que é de valor para um ou mais _____ ou outros interessados no sistema.

8) Diagrama de _____ é um diagrama que apresenta os _____ de mensagens, numa ordem _____, entre os _____ pertinentes ao respectivo _____, num determinado _____ de tempo.

9) Os objetos são desenhados como um retângulo ao topo de uma linha vertical tracejada e projetada para baixo, que recebe o nome de linha de vida. Além dos objetos, geralmente, uma instância de ator é desenhada como a primeira linha de vida. A descrição representa os símbolos utilizados no diagrama de:

a. Comunicação.

b. Caso de Uso.

c. Sequência.

d. Colaboração.

MÓDULO 4 – MODELAGEM DE ESTADO E DE IMPLEMENTAÇÃO

Neste módulo, são apresentadas as modelagens de estado e de implementação. Na modelagem de estado, veremos o diagrama de estado e o diagrama de atividade. Na modelagem de implementação, são mostrados dois diagramas: o de componentes e o de implantação. Todos foram elaborados com base no exemplo do sistema bancário.

1. Modelagem de estado

1.1. Representação de um diagrama de estado

De acordo com Blaha e Rumbaugh (2006), o modelo de estados consiste em vários diagramas de estados, um para cada classe com comportamento temporal importante para uma aplicação. O diagrama de estados é um conceito padrão da ciência da computação, uma representação gráfica para máquinas de estado finito que relaciona eventos e estados. Os eventos representam os estímulos externos, e os estados, os valores dos objetos.

Um sistema responde com base em estímulos externos. Dependendo deles, ocorrem mudanças em seu comportamento. O diagrama de estado mostra como um objeto se comporta quando recebe eventos externos.

As máquinas de estado são muito utilizadas na computação em aplicações de inteligência artificial. Elas avaliam os aspectos dinâmicos de uma construção.

Um diagrama de estado é composto pelos seguintes elementos: estado, transição, estado inicial e estado final.

- **Estado**

O símbolo de estado, denominado estado simples, é representado por um retângulo com bordas arredondadas contendo um nome de estado. Em outro símbolo, chamado de estado composto ou máquina de estado, o retângulo é maior, pois nele existirão outros estados.

É também possível dividir a máquina de estado por regiões. Estas são separadas por linhas tracejadas, na vertical ou na horizontal, dentro daquela. Os estados dentro de cada região são considerados um subestado de um estado composto. As figuras 20 e 21 exemplificam os símbolos descritos.

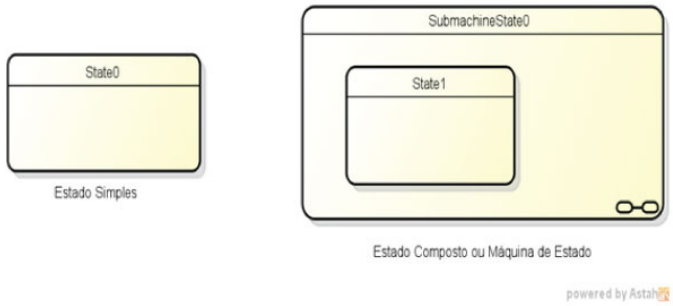

Figura 20: Estado simples e estado composto (máquina de estado).

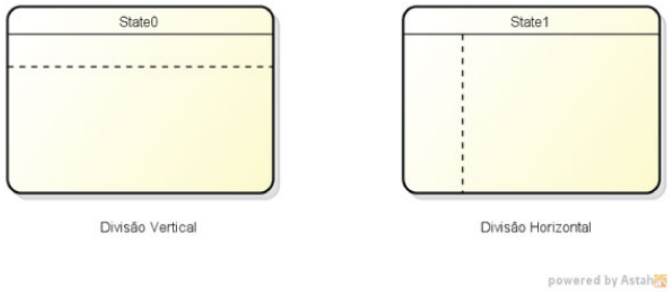

Figura 21: Máquinas de estado divididas por regiões na forma vertical e horizontal.

Segundo Lima (2011),

> um estado simplesmente representa uma ação executada, uma condição satisfeita ou uma situação estática de espera em que um objeto se encontra durante sua existência ou durante o processo de execução de alguma atividade no sistema.

Os diagramas de estados são muito utilizados para a validação de modelagem de classes. Para cada classe, poderá haver um diagrama de estado, com o objetivo de analisar todos os estados possíveis de cada objeto.

Na segunda parte do símbolo de estado, pode-se identificar uma *atividade do*. De acordo com Blaha e Rumbaugh (2006), uma atividade é o comportamento real que pode ser invocado por diversos efeitos. Efeito é uma referência a um comportamento executado em resposta a um evento. Assim, uma atividade do é uma atividade que continua ao longo de um período de tempo. Ela só pode ocorrer dentro de um estado e não pode estar conectada a uma transição. A figura 22 exemplifica isso.

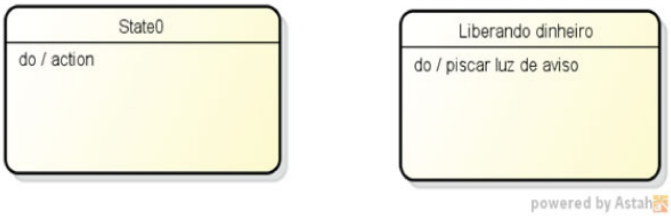

Figura 22: Notação UML e exemplo da atividade do, respectivamente.

- **Transição**

A transição é representada por uma seta e indica o relacionamento entre estados. A transição indica a execução de uma ou mais ações específicas: quando estiver no primeiro estado e entrar no segundo ou quando um evento específico ocorrer e determinadas condições forem satisfeitas. A figura 23 ilustra um exemplo de transição.

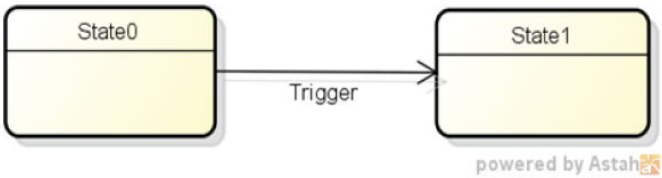

Figura 23: Transição de um estado origem para um estado destino.

O rótulo a ser descrito numa transição tem a seguinte sintaxe: *Evento(Argumentos) [condição] / Ação*, sendo que:

- **Evento**: é uma ocorrência num período de tempo. Os eventos podem ser do tipo:
  - ◊ Evento de sinal: representado por <<signal>> e indicado acima de uma classe.
  - ◊ Evento de mudança: representado pela palavra-chave *when* e seguido por uma expressão booleana. Exemplo: *when*(valor_do_saque > saldo_na_conta_corrente).

◊ Evento de tempo: representado pela palavra-chave *after* e seguida por uma expressão entre parênteses, indicando a duração de tempo. Exemplo: after(5 minutos). É utilizada juntamente com o evento de mudança.

- **Argumentos**: parâmetros do evento (valores).
- **Condição**: é uma expressão booleana representada entre colchetes. A transição somente se executará se a condição assumida pela expressão for verdadeira. A condição é denominada condição de guarda.
- **Ação**: é uma ação executada durante a transição.

Assim como nos estados, é possível especificar atividades em transição. As atividades *entry* e *exit* indicam entrada e saída, respectivamente. Vale ressaltar que elas podem ser utilizadas em transições e também em estados. A figura 24 exemplifica as atividades *entry* e *exit*, primeiramente nos estados e depois em transições.

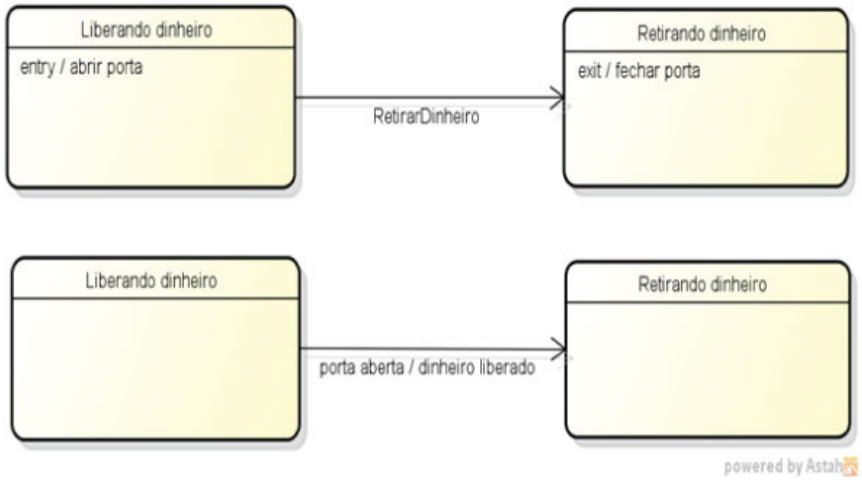

Figura 24: Atividades *entry* e *exit* em estados e em transições.

- **Estado inicial e estado final**

O estado inicial é representado por um círculo preenchido na cor preta (fechado) e indica que o objeto está inerte. O estado final é representado por um círculo aberto (branco) e, dentro deste, outro preenchido na cor preta (fechado). Ele indica o último estado de um objeto. Os estados inicial e final são considerados pseudoestados. Os exemplos representados pela figura 25 indicam o estado inicial e o estado final de um diagrama de estado.

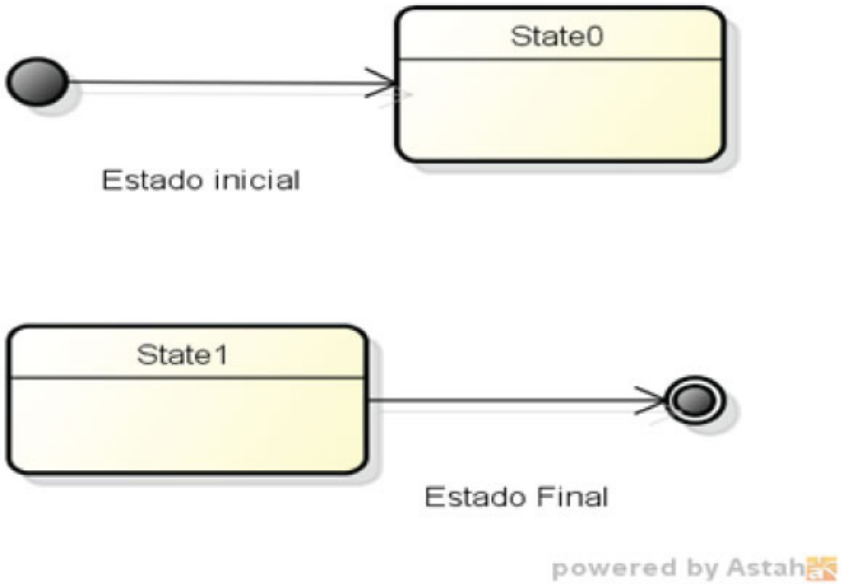

Figura 25: Estado inicial e estado final de um diagrama de estado.

O diagrama de estados possui duas estruturas importantes que podem ser utilizadas na modelagem de estados: a primeira é chamada *fork* (bifurcação), que permite a subdivisão de um estado em dois ou mais, concorrentes. A segunda estrutura é chamada *join* (junção), que une dois ou mais estados num único. A figura 26 apresenta um exemplo dessas duas estruturas.

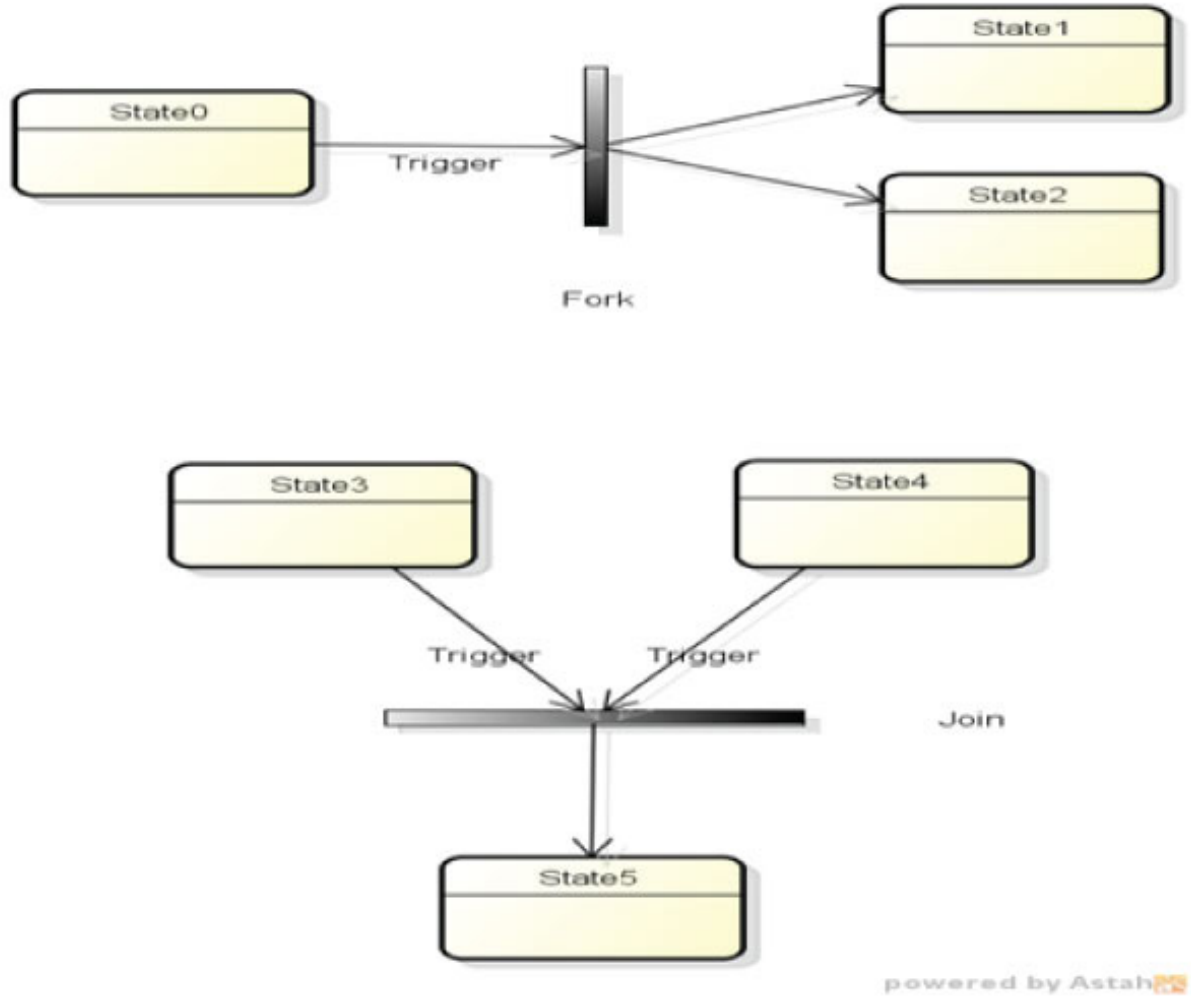

Figura 26: Estruturas do diagrama de estado: fork e join.

Na figura 12, o diagrama de classes apresenta uma classe chamada Conta. Com base nela, foi construído o diagrama de estado referente ao sistema bancário apresentado na figura 27.

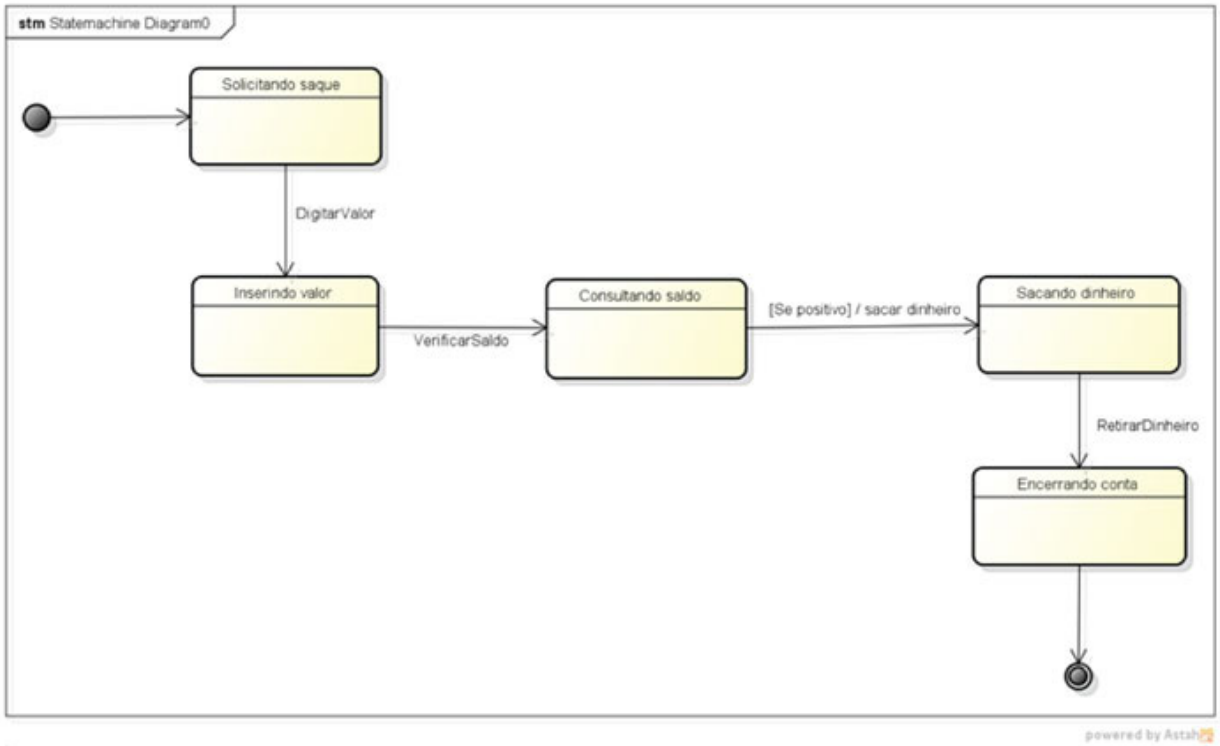

Figura 27: Diagrama de estado do sistema bancário.

1.2. Representação de um diagrama de atividades

O diagrama de atividades é um diagrama que mostra a sequência lógica de um sistema, como um algoritmo. Entretanto, nele, é possível manipular processos paralelos. O principal foco do diagrama de atividades são as operações.

Segundo o OMG (2014), um diagrama de atividades

> especifica a coordenação de execuções de comportamentos, usando um modelo de fluxo de controle e de dados. O fluxo de execução é modelado como nós de atividade conectados por extremidades. Um nó (nodo ou *node*) pode ser a execução de um comportamento subordinado, como um cálculo computacional, uma chamada para uma operação ou manipulação de conteúdos de objetos. Nós de atividade também incluem construções de fluxo de controle, como sincronização, decisão e controle de concorrência. Atividades podem invocar hierarquias de outras atividades, resolvidas em ações individuais.

De acordo com Furlan (1998), um diagrama de atividades é elaborado com os seguintes propósitos:

- Capturar o funcionamento interno de um objeto.
- Capturar o trabalho (ações) que será desempenhado quando uma operação for executada.
- Mostrar como um processo de negócio funciona em termos de atores, fluxos de trabalho, organização e objetos.
- Mostrar como uma instância de caso de uso pode ser realizada em termos de ações e mudanças de estado de objetos.
- Mostrar como um conjunto de ações relacionadas pode ser executado e como afetará objetos ao redor.

Os símbolos utilizados em um diagrama de atividades são:

- **Atividade**: representada por um retângulo com bordas arredondadas.
- **Transição**: representada por um fluxo de uma atividade para outra.
- **Decisão/merge**: representada por um losango, pode ter diversas saídas.
- **Bifurcação e junção (*fork* e *join*)**: representados por barras de sincronização.
- **Entrada**: representada por um círculo preenchido na cor preta (fechado).

- **Saída**: representada por um círculo aberto (branco) e, dentro deste, outro preenchido na cor preta (fechado).
- **Partições**: são representadas por um grande retângulo, que pode ser vertical ou horizontal.

A utilização da bifurcação e da junção de atividades é muito importante no diagrama de atividades, pois mais de uma atividade podem ocorrer ao mesmo tempo nesses casos. Esse fato é o que o diferencia de um fluxograma.

A bifurcação (*fork*) é uma atividade subdividida em duas ou mais concorrentes, e a junção (*join*) são duas ou mais atividades que se juntam numa única.

O losango representa tanto uma decisão quanto um merge. Quanto há uma entrada e diversas saídas, ele representa uma decisão. Nesse caso, cada saída é representada por uma condição de guarda, entre colchetes. Quando há diversas entradas e uma única saída, ele representa um merge e é utilizado para agregar diversos fluxos de controle em um só.

O diagrama de atividades permite também o uso de partições (swimlanes). Elas indicam diferentes ações que podem ser executadas por objetos ou entidades diferentes e podem ser verticais ou horizontais. É uma forma lógica de organização das atividades. Por exemplo: a partição Cliente indica as atividades relacionadas ao cliente, e a partição Conta indica as relacionadas à conta. Entretanto, as atividades não são paralelas e distintas, mas integradas entre as partições, com o objetivo de demonstrar os relacionamentos entre elas.

A figura 28 apresenta um exemplo do diagrama de atividades com base no sistema bancário.

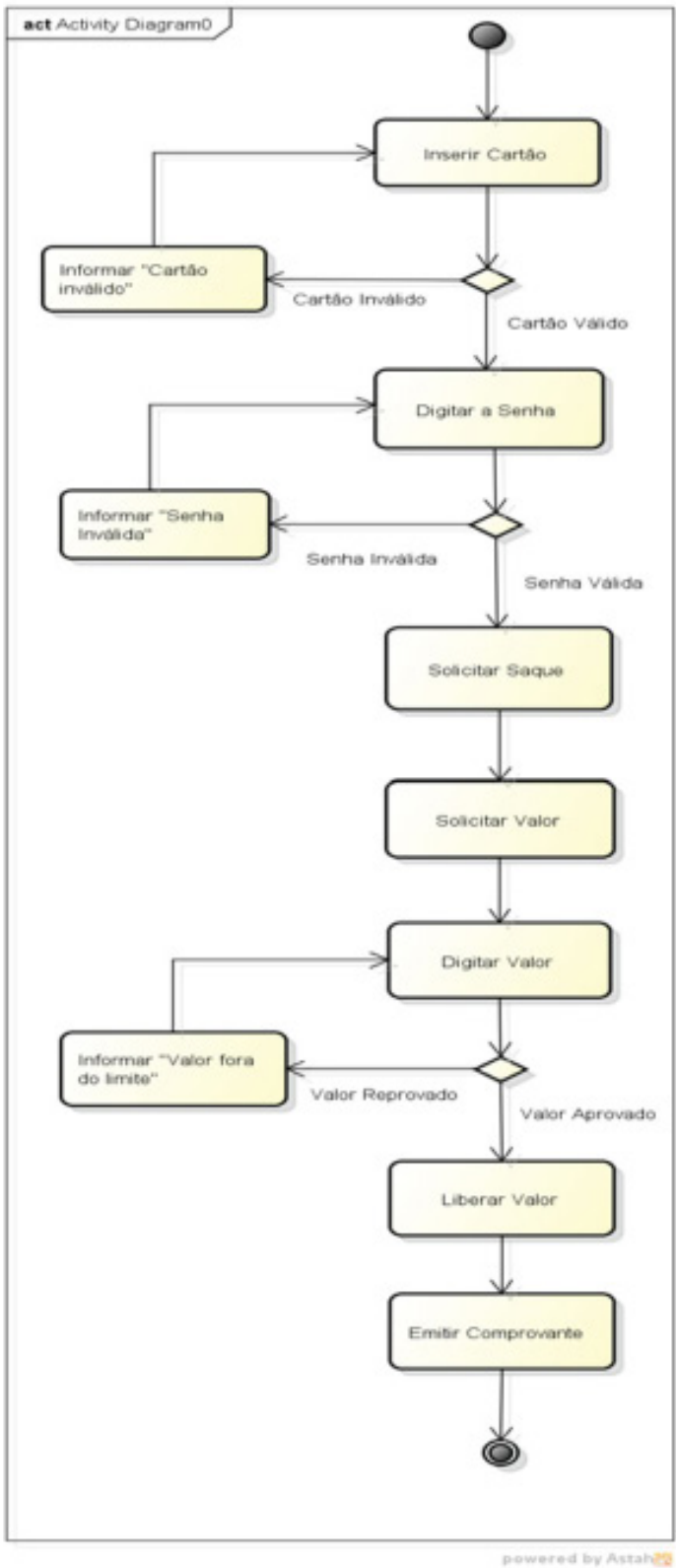

Figura 28: Diagrama de atividade do sistema bancário.

2. Modelagem de implementação

2.1. Representação de um diagrama de componentes

O diagrama de componentes representa todos os *softwares* que serão utilizados no sistema. Ele especifica os relacionamentos e as dependências entre os componentes de *software*. O diagrama de componentes é parte da especificação da arquitetura do sistema. Por meio dele, é possível especificar os componentes de vários ambientes da arquitetura, tais como ambiente de desenvolvimento, ambiente de produção, ambiente de testes etc.

A UML fornece alguns tipos de estereótipos que podem ser utilizados em diagramas de componentes: <<component>>, <<executable>>, <<library>>, <<document>>, <<interface>>, <<database>>, <<file>> etc.

O diagrama de componente é formado por componentes, interfaces e relacionamentos.

Um componente é qualquer arquivo que faça parte do desenvolvimento de um sistema. É um conjunto de interfaces. Os principais elementos utilizados como componentes em um diagrama de componentes são: código-fonte, bibliotecas, interfaces, base de dados, tabelas, arquivos e documentos complementares.

A figura 29 representa o símbolo de um componente.

Figura 29: Componente.

A interface é um conjunto de operações que executa os serviços de uma classe ou componente. O relacionamento entre um componente e uma interface dá-se por meio de uma realização (implementação da interface) ou de uma dependência (utilização da interface).

A figura 30 representa a notação UML para a interface esperada (*required interface*) e para a interface fornecida (*provided interface*).

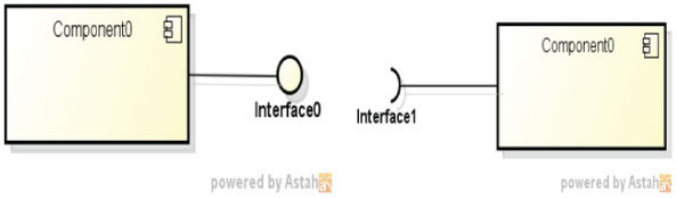

Figura 30: Provided interface e required interface, respectivamente.

As interfaces mostradas podem ser ligadas por meio do conector <<assembly>>, que une dois componentes. Isso significa que um componente fornece serviços para outro.

A figura 31 apresenta uma parte do diagrama de componentes do sistema bancário.

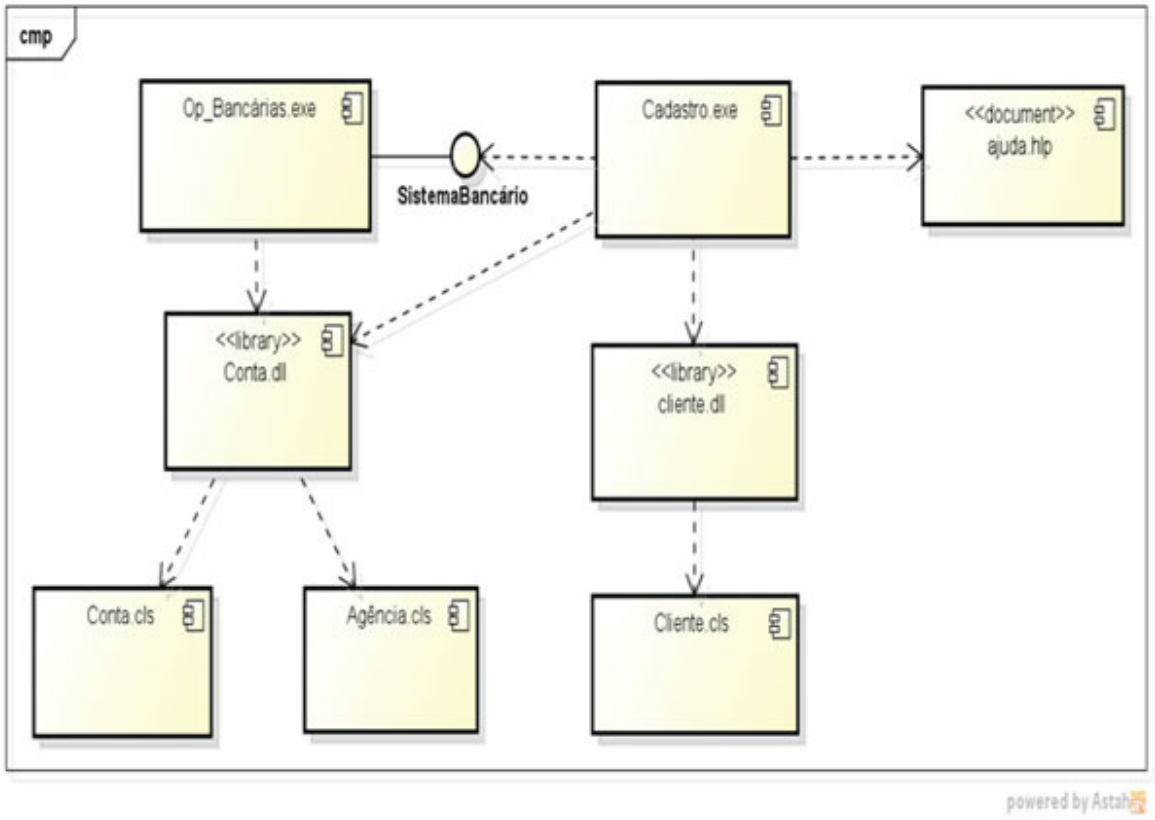

Figura 31: Diagrama de componentes do sistema bancário.

2.2. Representação de um diagrama de implantação

O diagrama de implantação (*deployment*) representa a arquitetura física (normalmente *hardware*) do sistema. Ele modela os recursos de infraestrutura, redes (servidores) e artefatos de sistema. É importante que os analistas entendam a composição dos elementos de *hardware* e dos ambientes de execução de *software* de um sistema.

O diagrama de implantação é um gráfico de nós, ou *nodes*. Nó é um elemento de *hardware* conectado por vias de comunicação. Ele representa um recurso de *hardware*, por exemplo, um servidor de aplicação, estações de trabalho, roteadores ou impressoras. Cada nó pode conter instâncias de nós ou de componentes.

A figura 32 ilustra o símbolo de um nó e de uma instância de um nó.

Figura 32: Nó e instância de um nó.

Além de nós, o diagrama de implantação é constituído também por artefatos e componentes. Os componentes são os mesmos utilizados no diagrama de componentes e com as mesmas características.

De acordo com o OMG (2013), “um artefato definido pelo usuário representa um elemento concreto no mundo

físico". Artefatos podem ser programas executáveis, códigos de programas, tabelas, documentos de texto etc. A figura 33 exemplifica o símbolo de um artefato.

Figura 33: Artefato.

O diagrama de implantação do sistema bancário, apresentado na figura 34, especifica quatro nós: Servidor, Servidor de BD, Computador e Impressora, sendo que o nó Servidor é composto pelos mesmos componentes indicados no diagrama de componentes do sistema.

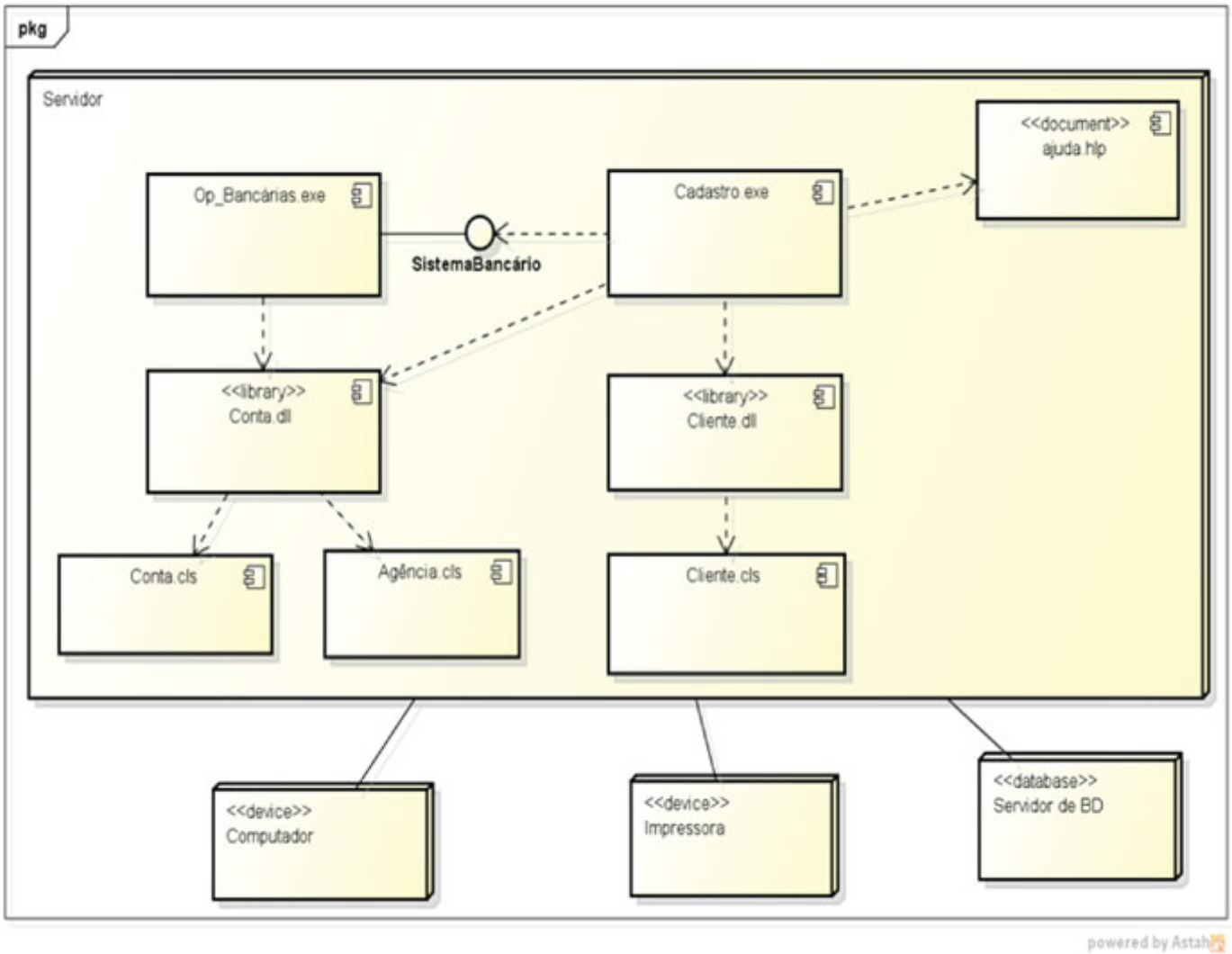

Figura 34: Diagrama de implantação do sistema bancário.

3. Exercícios do Módulo 4

1) O diagrama que mostra a sequência lógica de um sistema e permite a manipulação de processos paralelos é chamado de:

a. Diagrama de estado.
b. Diagrama de atividade.
c. Diagrama de componentes.
d. Diagrama de implantação.

2) O símbolo de um nó ou node que representa um recurso de *hardware* é utilizado no diagrama de:

a. Estado.
b. Atividade.
c. Componentes.
d. Implantação.

3) Um evento no diagrama de estado é uma ocorrência num período de tempo. Os eventos podem ser do tipo:

a. Sinal, mudança e tempo.
b. Do, entry e exit.
c. Transição, decisão e merge.
d. Argumentos, condição e ação.

4) Coloque "V" para verdadeiro e "F" para falso:

a. ( ) O mesmo símbolo de componentes utilizado no diagrama de componente é também utilizado no diagrama de implantação.

b. ( ) O relacionamento entre um componente e uma interface dá-se por meio de uma realização ou dependência.

c. ( ) A interface esperada (provided interface) e a interface fornecida (required interface) são conectadas por meio do conector <<assembly>>.

d. ( ) O diagrama de componentes permite o uso de partições (swimlanes) que indicam diferentes ações que podem ser executadas por objetos ou entidades diferentes.

e. ( ) A bifurcação (fork) indica duas ou mais atividades que se unem numa única atividade e junção (join) indica uma atividade subdividida em duas ou mais atividades concorrentes.

f. ( ) O símbolo de um estado inicial é representado por um círculo preenchido na cor preta (fechado) e o símbolo de um estado final é representado por um círculo aberto (branco) e, dentro deste, um círculo preenchido na cor preta (fechado).

5) O diagrama de _____ é um conceito padrão da _____ (uma representação gráfica para _____ de estado finito), que relaciona _____ e _____. Os eventos representam os _____ e os estados representam os _____.

6) Estado _____, estado _____ (máquina de estado) e _____ (estados subdivididos por _____) são formas de representar os _____ num diagrama de estado.

7) Segundo o OMG, "um artefato definido pelo usuário representa um elemento concreto no mundo físico". Os artefatos podem ser programas executáveis, códigos de programas, tabelas, etc. Os artefatos são utilizados no diagrama de:

a. Estado.
b. Atividade.
c. Comunicação.
d. Implantação.

MÓDULO 5 - PADRÕES DE PROJETO (DESIGN PATTERNS)

Neste módulo, você conhecerá os principais conceitos sobre padrões de projeto, ou *design patterns*. Ele não se propõe a apresentar a parte prática da aplicação dos padrões de projeto com utilização da linguagem Java, que será discutido em disciplina específica.

1. Padrões de projeto

1.1. Fundamentos dos padrões de projeto

Em engenharia de *software*, estudamos as técnicas, os métodos e as ferramentas necessários para o desenvolvimento de *software*. Nesses conteúdos, também são abordados assuntos relacionados à reutilização de códigos. Os padrões de projeto baseiam-se no reúso e na facilidade que essa reutilização proporciona no desenvolvimento de projetos orientados a objetos.

Além disso, os padrões de projeto abordam também aspectos comportamentais que podem ser aplicados em outros projetos. O analista deve projetar *software* orientado a objetos reutilizável, a fim de atender a outros requisitos semelhantes.

Segundo Deitel e Deitel (2005), os padrões de projeto beneficiam os desenvolvedores de um sistema nos seguintes aspectos:

- Na construção de um *software* confiável com arquitetura testada.
- Na promoção da reutilização de projetos em futuros sistemas.
- Na identificação de equívocos comuns e armadilhas que ocorrem ao se construírem sistemas.
- Na criação de projetos de sistemas independentemente da linguagem em que eles, em última instância, serão implementados.
- No estabelecimento de um vocabulário comum de projeto entre os desenvolvedores.
- No encurtamento da fase de projeto no processo de desenvolvimento de um *software*.

De acordo com Christopher Alexander (1977 apud GAMMA et al., 2000), "cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira".

O autor Christopher Alexander é o precursor dos trabalhos relacionados a padrões (patterns). Ele é um arquiteto e matemático que ficou muito conhecido, na década de 1970, pelos trabalhos desenvolvidos na criação de padrões geométricos e matemáticos na área de arquitetura.

Na década de 1990, quatro autores - Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides -, lançaram o primeiro catálogo de padrões de projeto, chamado livro GoF (*Gang of Four*). Desde então o desenvolvimento de projetos de *software* fundamenta-se em terminologia e soluções baseadas nesse catálogo.

Para Gamma et al. (2000), padrões de projetos são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto num contexto particular. Segundo os autores, um padrão tem quatro elementos essenciais:

- **Nome do padrão**: é uma referência que identificará o problema de projeto, suas soluções e consequências.
- **Problema**: descreve o problema e o contexto com o intuito de saber quando aplicar o padrão.
- **Solução**: descreve os elementos que compõem o projeto, seus relacionamentos, suas responsabilidades e colaborações.
- **Consequências**: são os resultados e a análise das vantagens e desvantagens (*trade-offs*) da aplicação padrão.

Existem outros catálogos de padrões, tais como Padrões JEE, Padrões Microsoft, Padrões SOA etc. Eles são referências em seus respectivos campos de atuação.

Os padrões de projeto são representados por gráficos. Os objetos e classes são desenhados, e os seus relacionamentos, apresentados. Após essa notação gráfica, os códigos são especificados e explicados.

Além dessa notação gráfica, é necessária uma descrição dos padrões de projeto. Nela são especificadas decisões, análise de custo-benefício, exemplos etc. Gamma et al. (2000) descrevem uma estrutura uniforme para essas informações:

- **Nome e classificação do padrão**: nome do padrão e indicação da categoria dele.
- **Intenção e objetivos**: curta declaração das intenções e objetivos do padrão.
- **Também conhecido como**: outros nomes também utilizados para o padrão, se existirem.
- **Motivação**: cenário do problema e da solução de projeto por meio do padrão.
- **Aplicabilidade**: situações de aplicação do padrão de projeto.
- **Estrutura**: representação gráfica das classes do padrão. Podem-se utilizar diagramas de interação.
- **Participantes**: classes e objetos que participam do padrão de projeto e suas responsabilidades.
- **Colaborações**: como os participantes colaboram para executar suas responsabilidades.
- **Consequências**: indicação das consequências da aplicação do padrão, como custo-benefício.
- **Implementação**: informações relativas à implementação do padrão de projeto, como linguagem de programação, dentre outras.

- **Exemplo de código**: fragmentos ou blocos de códigos da implementação do padrão.
- **Usos conhecidos**: exemplos do padrão encontrados em sistema reais.
- **Padrões relacionados**: descrição dos padrões relacionados ao padrão utilizado.

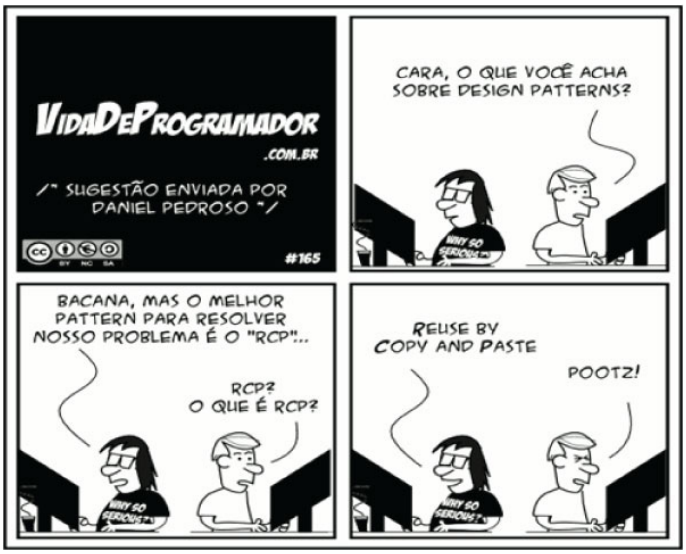

Figura 35. Tirinha de humor sobre *design patterns*. Fonte: http://vidadeprogramador.com.br/2011/07/08/design-patterns/.

1.2. Catálogo dos padrões de projeto GoF

O catálogo de padrões GoF está dividido em três categorias: criação, estrutural e comportamental. Os padrões de criação são utilizados para criar objetos. Os padrões estruturais preocupam-se com a composição de objetos e classes. Já os padrões comportamentais lidam com a interação de objetos e classes e suas responsabilidades.

Além dessas categorias, Gamma et al. (2000), organizam o catálogo GoF em 23 padrões, conforme relação, em ordem alfabética, descrita a seguir.

1. ***Abstract factory***: fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
2. ***Adapter***: converte a interface de uma classe em outra esperada pelos clientes. O *adapter* permite que certas classes trabalhem em conjunto, pois, de outra forma, seria impossível, por causa de sua interfaces incompatíveis.
3. ***Bridge***: separa uma abstração da sua implementação, de modo que as duas possam variar independentemente.
4. ***Builder***: separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.
5. ***Chain of responsibility***: evita o acoplamento do remetente de uma solicitação ao seu destinatário, dando a mais de um objeto a chance de tratar a solicitação. Encadeia os objetos receptores e passa a solicitação ao longo da cadeia até que um deles a trate.
6. ***Command***: encapsula uma solicitação como um objeto, permitindo que se parametrize clientes com diferentes solicitações, se enfileire ou registre (*log*) solicitações e se suporte operações que podem ser desfeitas.
7. ***Composite***: compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. O *composite* permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.
8. ***Decorator***: atribui responsabilidades adicionais a um objeto dinamicamente. Os *decorators* fornecem uma alternativa flexível a subclasses para extensão da funcionalidade.
9. ***Facade***: fornece uma interface unificada para um conjunto de interfaces em um subsistema. O *façade* define uma interface de nível mais alto que torna o subsistema mais fácil de usar.
10. ***Factory method***: define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. O factory method permite a uma classe postergar (*defer*) a instanciação às subclasses.
11. ***Flyweight***: usa compartilhamento para suportar grandes quantidades de objetos, de granularidade fina, de maneira eficiente.
12. ***Interpreter***: dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças nesta linguagem.
13. ***Iterator***: fornece uma maneira de acessar sequencialmente os elementos de um objeto agregado sem expor sua representação subjacente.
14. ***Mediator***: define um objeto que encapsula a forma como um conjunto de objetos interage. O *mediator* promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros, permitindo que você varie suas interações independentemente.
15. ***Memento***: sem violar a encapsulação, captura e externaliza um estado interno de um objeto, de modo que o mesmo possa, posteriormente, ser restaurado para esse estado.
16. ***Observer***: define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.
17. ***Prototype***: especifica os tipos de objetos a serem criados usando uma instância prototípica e cria novos objetos copiando este protótipo.
18. ***Proxy***: fornece um objeto representante (*surrogate*), ou um marcador de outro objeto, para controlar o acesso ao mesmo.

19. ***Singleton***: garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela.
20. ***State***: permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado sua classe.
21. ***Strategy***: define uma família de algoritmos, encapsula cada um deles e torna-os intercambiáveis. O *strategy* permite que o algoritmo varie independentemente dos clientes que o utilizam.
22. ***Template method***: define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses. O *template method* permite que as subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.
23. ***Visitor***: representa uma operação a ser executada sobre os elementos da estrutura de um objeto. O *visitor* permite que você defina uma nova operação sem mudar as classes dos elementos sobre os quais opera.

A tabela 5 apresenta a subdivisão dos 23 padrões distribuídos nas três categorias do catálogo GoF.

Tabela 5: Espaço dos padrões de projeto.

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	*Factory method*	*Adapter (class)*	*Interpreter*
				Template method
	Objeto	*Abstract factory*	*Adapter (object)*	*Chain of responsability*
		Builder	*Bridge*	*Command*
		Prototype	*Composite*	*Iterator*
		Singleton	*Decorator*	*Mediator*
			Façade	*Memento*
			Flyweight	*Observer*
			Proxy	*State*
				Strategy
				Visitor

Fonte: Adaptado de Gamma et al., 2000.

Há muitas maneiras de organizar os padrões de projeto: padrões utilizados em conjunto, padrões alternativos a outros, padrões por semelhança etc. A figura 36 representa uma forma chamada de “padrões relacionados”, que significa a forma como determinados padrões mencionam outros.

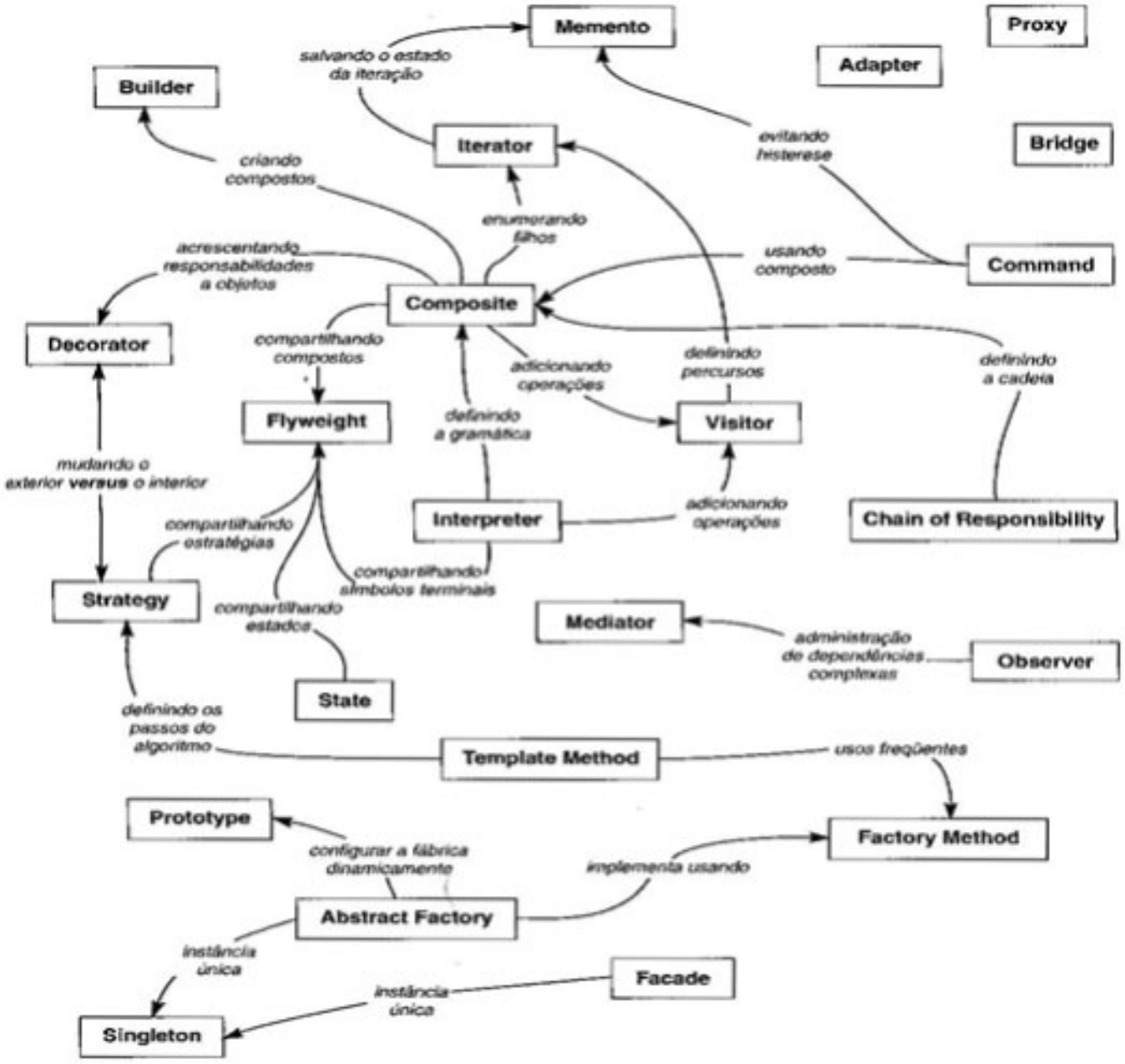

Figura 36: Relacionamentos entre padrões de projeto. Fonte: Gamma et al., 2000.

Considerações finais

Caro aluno, espero que você tenha obtido novos conhecimentos técnicos para aplicação na sua vida profissional. Esta disciplina trata de vários conceitos abstratos e, certamente, para entendê-los, é necessário transformá-los para o mundo real.

As primeiras aulas abordaram uma introdução e os principais conceitos relacionados à orientação a objetos. Esses conceitos são pré-requisitos para o entendimento dos diagramas UML. A notação UML deve ser muito bem compreendida, com o intuito de padronizar o desenvolvimento de projetos orientados a objetos.

A partir do módulo 2, iniciou-se a representação dos diagramas UML. Os exemplos dos diagramas foram baseados em um sistema bancário. O primeiro, chamado diagrama de classes, apresenta as classes relacionadas ao sistema bancário. Com base nelas, os demais foram desenvolvidos.

Vale ressaltar que não há uma sequência lógica para elaborar os diagramas. Eles são elaborados de acordo com as necessidades do ciclo de vida do sistema. Além disso, nem todos são elaborados a partir do diagrama de classes.

Conforme você deve ter notado, a modelagem UML referente ao sistema bancário não está completa. Para cada assunto, foi apresentado apenas um diagrama como exemplo. Outro detalhe importante são os casos especiais e as características de cada um. Nem sempre foi possível indicar todos em um único diagrama.

Eu sugiro, caro aluno, que você se mantenha atualizado em relação às mudanças da UML. Este material foi elaborado com base na versão 2.5. Portanto, é necessário manter toda a documentação relacionada aos projetos orientados a objetos, de forma atualizada e padronizada, com o intuito de melhoria na qualidade dos sistemas.

Prof. Claudinei

RESPOSTAS DOS EXERCÍCIOS

Módulo 1

1) Associe:

a. (1) Método OOSE (*4*) Grady Booch
b. (2) Método OMT (*2*) James Rumbaugh
c. (3) Método OOSA (*1*) Ivar Jacobson
d. (4) Método Booch (*3*) Shlaer e Mellor

2) UML significa Linguagem de Modelagem Unificada e é um método de especificação para ferramentas de modelagem.

a. Verdadeiro
b. *Falso*

3) Segundo Blaha e Rumbaugh (2006) a metodologia orientada a objetos possui os seguintes estágios:

a. Identidade, classificação, herança e polimorfismo
b. Análise, projeto e implementação
c. *Concepção do sistema, análise, projeto do sistema, projeto de classes e implementação*
d. Classes, estados e interações

4) O OMG é uma associação aberta internacional sem fins lucrativos fundada em 1989.

a. *Verdadeiro*
b. Falso

5) Indique abaixo a alternativa que não está relacionadaaos objetivos de Booch, Jacobson e Rumbaugh ao desenvolverem a UML:

a. Fornecer aos usuários de uma linguagem de modelagem visual expressiva e pronta para uso, de forma que eles possam desenvolver e intercambiar modelos significativos.
b. Integrar as melhores práticas.
c. Estimular o crescimento do mercado de ferramentas orientadas a objetos.
d. *Ser dependente das linguagens de programação e dos processos de desenvolvimento particulares.*

6) O desenvolvimento Orientado a Objetos dá uma ênfase maior à estrutura de dados e uma ênfase menor à estrutura de procedimentos.

a. *Verdadeiro*
b. Falso

7) A modelagem de um sistemas se dá por meio de três modelos. Qual dos itens listados abaixo não se constitui num modelo da modelagem de sistemas?

a. *Herança*
b. Classes
c. Estados
d. Interações

8) A orientação a objetos é uma *metodologia* utilizada para entender melhor os *requisitos do cliente* e solucionar os problemas com maior qualidade. A construção da modelagem orientada a *objetos* se dá por meio de objetos, que combina *estrutura de dados* e *comportamento*. É um modo de pensar e não uma técnica de programação.

Módulo 2

9) Um Diagrama de Classes é representado por um conjunto de classes com os seus respectivos atributos e operações.

a. *Verdadeiro*
b. Falso

10) Segundo Blaha e Rumbaugh (2006) uma classe descreve um grupo de objetos com as mesmas propriedades (atributos), comportamentos (operações), tipos de relacionamentos e semântica:

a. *Verdadeiro*
b. Falso

11) Em relação ao polimorfismo podemos afirmar que:

a. A mesma operação pode se comportar de forma diferente para todas as classes.
b. A mesma operação pode se comportar de forma igual para diferentes classes.
c. *A mesma operação pode se comportar de forma diferente para diferentes classes.*
d. A mesma operação pode se comportar de forma igual para todas as classes.

12) Quais são os tipos de visibilidade de um atributo?

a. Pública e privada.
b. Pública, privada e protegida.
c. *Pública, privada, protegida e pacote.*
d. Pública, privada, protegida, pacote e oculta.

13) Podemos dizer que a Composição é:

a. *Um caso especial de associação e uma forma de relacionamento todo/parte.*

b. Formada por um losango aberto.

c. É um relacionamento todo/parte, entretanto a parte poderá existir sem o todo.

d. Não é um relacionamento todo/parte, pois a composição é constituída por componentes que formam o composto.

14) Encapsulamento é o mesmo que:

a. Herança, ou seja, estabelece uma relação de ocultamento de informação.

b. *Ocultação de informação.*

c. Generalização, pois as informações são herdadas da superclasse para a subclasse.

d. Ocultar determinados detalhes da implementação.

7) Multiplicidade é uma notação *UML*. É uma simbologia utilizada em *ligações* e *associações*. Ela especifica o *número* de instâncias de uma *classe* que pode se relacionar a uma *única* instância de uma classe associada [Blaha, Rumbaugh, 2006].

8) Herança é a capacidade de *uma* ou mais *subclasses* herdarem os *atributos* e as *operações* da superclasse por meio do relacionamento de *generalização*.

Módulo 3

10) Na mais recente versão da UML, o Diagrama de Colaboração é chamado de Diagrama de Comunicação.

a. *Verdadeiro*

b. Falso

11) O fluxo de mensagens assíncrono requer um retorno do remetente.

a. Verdadeiro

b. *Falso*

12) <<include>> e <<extend>> são estereótipos utilizados no Diagrama de Caso de Uso.

a. *Verdadeiro*

b. Falso

13) Autodelegação é uma técnica utilizada em algoritmos para mostrar que uma operação chama a si própria.

a. *Verdadeiro*

b. Falso

14) Faça as respectivas associações:

1. Diagrama de Caso de Uso
2. Diagrama de Sequência
3. Diagrama de Comunicação

a. (*3*) Mostrar os relacionamentos entre objetos.

b. (*2*) Mostrar como ocorre o fluxo de mensagens entre os objetos ao longo do tempo.

c. (*1*) Mostrar a funcionalidade do sistema.

15) O modelo de *interações* descreve as interações dentro de um *sistema*. Ele descreve como os *objetos* interagem para produzir resultados úteis.

16) Um caso de *uso* é a especificação de um conjunto de *ações* executado tipicamente por um *sistema* que entrega um *resultado* observável que é de valor para um ou mais *atores* ou outros interessados no sistema.

17) Diagrama de *Sequência* é um diagrama que apresenta os *fluxos* de mensagens, numa ordem *lógica*, entreos *objetos* pertinentes ao respectivo *caso de uso*, num determinado *período* de tempo.

18) Os objetos são desenhados como um retângulo ao topo de uma linha vertical tracejada e projetada parabaixo, que recebe o nome de linha de vida. Além dos objetos, geralmente, uma instância de ator é desenhada como a primeira linha de vida. A descrição acima representa os símbolos utilizados no diagrama de

a. Comunicação.

b. Caso de Uso.

c. *Sequência*.

d. Colaboração.

Módulo 4

1) O Diagrama que mostra a sequência lógica de um sistema e permite a manipulação de processos paralelos é chamado de

a. Diagrama de Estado.

b. *Diagrama de Atividade.*

c. Diagrama de Componentes.

d. Diagrama de Implantação.

2) O símbolo de um nó ou node que representa um recurso de *hardware* é utilizado no Diagrama de

a. Estado.

b. Atividade.

c. Componentes.

d. *Implantação.*

3) Um evento no Diagrama de Estado é uma ocorrência num período de tempo. Os eventos podem ser do tipo:

a. *Sinal, Mudança e Tempo.*

b. Do, Entry e Exit.

c. Transição, Decisão e Merge.

d. Argumentos, Condição e Ação.

4) Coloque "V" para verdadeiro e "F" para falso:

a. (*V*) O mesmo símbolo de componentes utilizado no Diagrama de Componente é também utilizado no Diagrama de Implantação.

b. (*V*) O relacionamento entre um componente e uma interface dá-se por meio de uma realização ou dependência.

c. (*F*) A interface esperada (Provided Interface) e a interface fornecida (Required Interface) são conectadas por meio do conector <<assembly>>.

d. (*F*) O Diagrama de Componentes permite o uso de partições (swimlanes) que indicam diferentes ações que podem ser executadas por objetos ou entidades diferentes.

e. (*F*) A bifurcação (Fork) indica duas ou mais atividades que se unem numa única atividade e junção (Join) indica uma atividade subdividida em duas ou mais atividades concorrentes.

f. (*V*) O símbolo de um estado inicial é representado por um círculo preenchido na cor preta (fechado) e o símbolo de um estado final é representado por um círculo aberto (branco) e, dentro deste, um círculo preenchido na cor preta (fechado).

5) O diagrama de *estados* é um conceito padrão da *ciência da computação* (uma representação gráfica para *máquinas* de estado finito), que relaciona *eventos* e *estados*. Os eventos representam os *estímulos externos* e os estados representam os *valores dos objetos*.

6) Estado *Simples*, Estado *Composto* (Máquina de Estado) e *Subestado* (estados subdivididos por *regiões*) são formas de representar os *estados* num diagrama de Estado.

7) Segundo o OMG, "um artefato definido pelo usuário representa um elemento concreto no mundo físico". Os artefatos podem ser programas executáveis, códigos de programas, tabelas, etc. Os artefatos são utilizados no Diagrama de

a. Estado.

b. Atividade.

c. Comunicação.

d. *Implantação.*

REFERÊNCIAS BIBLIOGRÁFICAS

ASTAH. ***Astah reference* manual**. 2013. Disponível em: http://goo.gl/zUgoa8, acesso em: 9.fev.2014.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Elsevier, 2007.

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

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. Rio de Janeiro: Elsevier, 2012.

BLOOGS, Wendy. ***Mastering UML* com Rational Rose 2002: a bíblia**. Rio de Janeiro: Alta Books, 2002.

COCKBURN, Alistair. **Escrevendo casos de uso eficazes**. Porto Alegre: Bookman, 2005.

DEITEL, Harvey M.; DEITEL, Paul J. **Java: como programar**. São Paulo: Pearson Prentice-Hall, 2005.

FOWLER, M.; KENDALL, S. **UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos**. Porto Alegre: Bookman, 2005.

FURLAN, José D. **Modelagem de objetos através da UML: análise e desenho orientados a objetos**. São Paulo: Makron Books, 1998.

GAMMA, Erich et al. **Padrões de projeto: soluções reutilizáveis de *software* orientado a objeto**. Porto Alegre: Bookman, 2000.

GANG OF FOUR (GOF). ***Design patterns***. Disponível em: http://www.gofpatterns.com, acesso em: 5.fev.2014.

GUEDES, G. T. A. **UML 2: uma abordagem prática**. São Paulo: Novatec, 2011.

LARMAN, C. **Utilizando UML e padrões**. Porto Alegre: Bookman, 2007.

LIMA, Adilson S. **UML 2.3: do requisito à solução**. São Paulo: Érica, 2011.

MEDEIROS, Ernani S. **Desenvolvendo *software* com UML 2.0: definitivo**. São Paulo: Pearson Makron Books, 2004.

OBJECT MANAGEMENT GROUP (OMG). ***Introduction to OMG's Unified Modeling Language (UML)***. 2013. Disponível em http://www.omg.org/gettingstarted/what_is_uml.htm, acesso em: 9.fev.2014.

_____. ***UML: unified modeling language***. 2014. Disponível em http://www.uml.org, acesso em: 9.fev.2014.

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

PAULA FILHO, Wilson P. **Engenharia de *software*: fundamentos, métodos e padrões**. 3ª ed. Rio de Janeiro: LTC, 2009.

PETERS, J. F.; PEDRYC, W. **Engenharia de *software*: teoria e prática**. Rio de Janeiro: Campus, 2001.

PRESSMAN, R. S. **Engenharia de *software***. 7ª ed. São Paulo: McGraw-Hill, 2010.

SCHACH, Stephen. **Engenharia de *software*: os paradigmas clássico e orientado a objetos**. São Paulo: McGrawHill, 2008.

SILVA, R. P. **UML 2 em modelagem orientada a objetos**. Florianópolis: Visual Books, 2007.

SOMMERVILLE, Ian. **Engenharia de *software***. São Paulo: Addison-Wesley, 2007.