



1

Análise e Projeto de Sistemas II

Prof. William Pelissari

William.pelissari@faculdadefacec.edu.br

wrpelissari@gmail.com

2

➤ Nome do Docente

➤ William Roberto Pelissari

➤ Formação

➤ Mestre em Desenvolvimento de Tecnologia

➤ Experiência

➤ Coordenador de área na Virtual Age

➤ Coordenador e professor de Pós-Graduação

➤ Coordenador de Graduação

➤ Professor de Graduação

➤ E-mail Institucional

➤ ads@faculdadefacec.edu.br

3

➤ O que esperar da disciplina?

➤ Qual a sua importância para o profissional formando em ADS?

➤ Carga horária 120h

➤ Aulas práticas

➤ Normas

➤ Avaliações

➤ Tipo(s): Teórica, Teórica e Prática – Peso 7 para parte teórica e 3 para prática (Regra)

➤ De 5 a 15 questões (Dissertativas e Objetivas)

➤ Utilização de meios fraudulentos – consequência

➤ Bimestrais

➤ 2^a. oportunidade – mediante pagamento de taxa

➤ Substitutiva – mediante pagamento de taxa

➤ Exame

2022

4

➤ Ementa:

Histórico e evolução das metodologias de orientação a objetos.

Análise de Requisitos.

Aspectos de Linguagem de Modelagem Unificada.

Modelos e Diagramas OO.

Ferramentas automatizadas.

Estudo de caso.

2022

5

➤ Plano de Ensino:

- Conceito de Análise e Projeto de Sistemas
- Revisão de Análise de Requisitos
- Conceito de Programação Orientação ao Objeto
- Conceitos e Paradigmas da OO
- Diagramas
 - Diagrama de Classe
 - Diagrama de Caso de Uso
 - Diagramas de Interação
 - Diagrama de Sequência
 - Diagrama de Colaboração
 - Diagrama de Estado
 - Diagrama de Atividade
 - Diagramas de implementação
 - Diagrama de Componente
 - Diagrama de implantação

2022

6

Referências

- PRESSMAN, Roger. **Engenharia de software.** São Paulo: McGraw-Hill., 2006
- BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro: Campus, 2007.
- TONSING, Sérgio Luiz. Engenharia de Software – Análise e Projeto de Sistemas. 2^a Edição. Rio de Janeiro: Ed. Ciência Moderna, 2008.
- LARMAN, Graig. Utilizando UML e Padrões. 3^a. Ed. Bookman. 2008.
- SOMMERSVILLE, Ian. Engenharia de Software. 6^a ed. Ed. Addison Wesley. 2003.

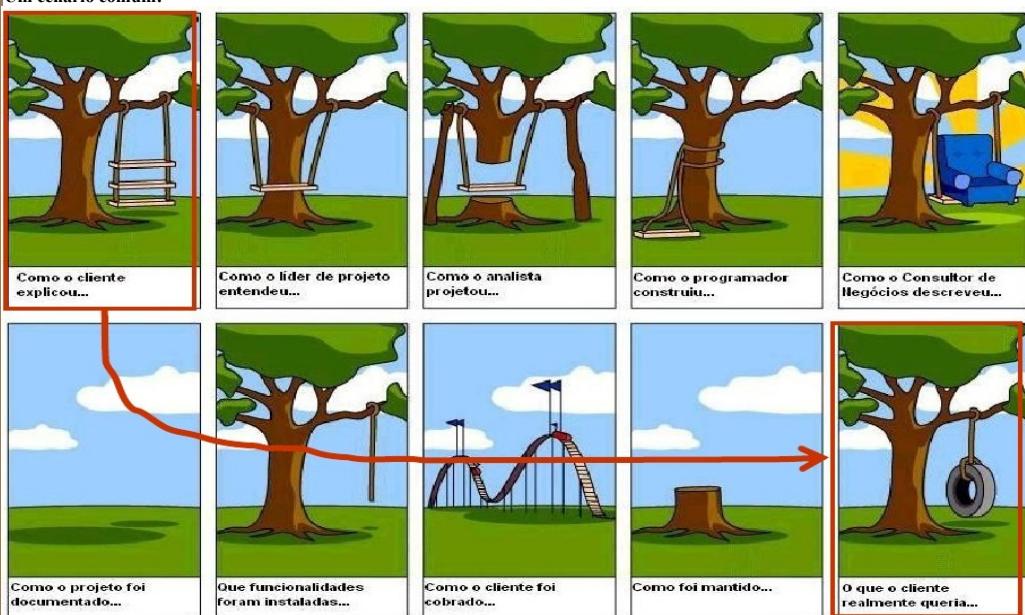
7



8

Analise de Requisitos de Software

Vamos relembrar
Um cenário comum:



Prof.William Pelissari

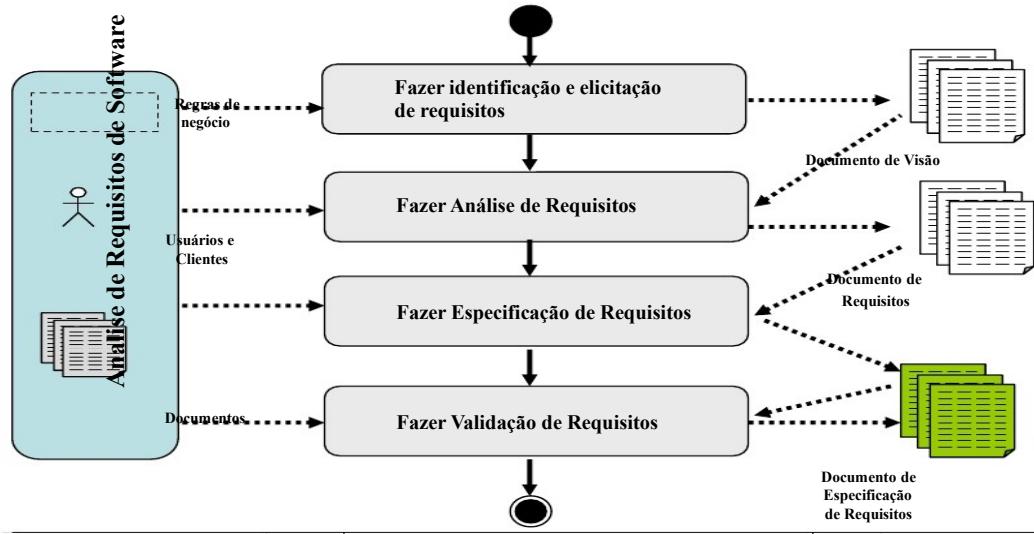
Análise e Desenvolvimento de Sistemas

9

9

Requisitos de Software

Requisitos. Road Map



Prof.William Pelissari

10

Sistemas de Informação

- Combinação de PESSOAS, DADOS, PROCESSOS, INTERFACES, REDES DE COMUNICAÇÃO, TECNOLOGIAS...
- Com o objetivo de melhorar o processo de negócio de uma organização!!

ADICIONAR VALOR

11

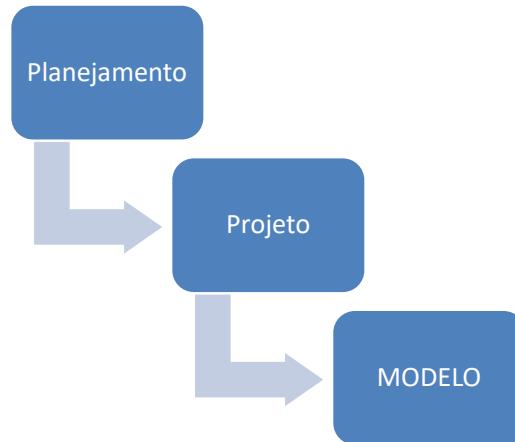
Por que Análise?

- Característica intrínseca de um sistema de informação é a complexidade de seu desenvolvimento, que aumenta a medida que cresce o tamanho do sistema (BEZERRA, 2007).

NECESSITA de PLANEJAMENTO!

12

Modelagem de Sistemas de Informação



13

MODELO

- Razões para utilizar modelos (BEZERRA, 2007):
 - Gerenciamento da complexidade
 - Comunicação entre pessoas envolvidas
 - Redução dos custos no desenvolvimento
 - Previsão do comportamento futuro do sistema

14

Modelo

- Gerenciamento da complexidade
 - Utilização de modelos onde cada qual descreve uma perspectiva do sistema.
 - Através dos modelos é possível realizar estudos e prever comportamentos.
 - Modelos baseiam-se no “*Princípio da Abstração*”.

15

Modelo

- Comunicação entre pessoas envolvidas
 - Grande quantidade de atividades.
 - Resultam em informação
 - Essas informações correspondem aos modelos.
 - Utilizados para promover a difusão de informação relevantes ao sistema dentre os indivíduos envolvidos em sua construção.

16

Modelo

- Redução dos custos no desenvolvimento
 - Correções a erros são menos custosas quando detectada e realizada na fase inicial.
 - Modelos de sistemas são mais baratos de construir que os sistemas. Consequentemente, erros identificados sobre modelos tem um impacto menos desastroso.

17

Modelos

- Representados por diagramas
 - Os desenhos gráficos que modelam os sistemas apresentam uma representação concisa do sistema.
 - A modelagem de sistemas de informação consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se várias perspectivas diferentes e complementares.

18

A Análise e Projeto de Sistemas e a Engenharia de Software

- A análise e projeto de sistemas dentro da Engenharia de Software.



19

O Analista de Sistemas

- “O provérbio ‘possuir um martelo não torna alguém um arquiteto’ é particularmente verdadeiro em relação à tecnologias de objetos. Conhecer uma linguagem orientada a objetos (como Java) é um primeiro passo necessário, mas insuficiente, para criar sistemas orientado a objetos. Saber pensar em termos de objetos é crucial.” (LARMAN, 2008).

20

Atividades da Análise de Sistemas

- Análise é o estudo de um problema, que antecede à tomada de uma ação... Estudo de alguma área de trabalho ou de uma aplicação levando quase sempre à especificação de um novo sistema (Tonsing, 2008).

21

Atividades da Análise de Sistemas

- Consiste em:
 - Métodos e técnicas de investigação e especificação da solução dos problemas.
 - Sua origem ocorre nos requisitos levantados.
- Suas atividades vão desde o “simples” relacionamento humano até o mais complexo pensamento abstrato.
- Necessita documentar todas as soluções encontradas para um problema, utilizando especificações técnicas de acordo com um método.

22

O Analista de Sistemas

- O analista de sistemas é o elo entre os usuários e o corpo técnico de codificação dos programas (programador).
- Suas atividades:
 - Desenhar soluções, empregando um método.
 - Analisar e projetar sistemas eficientes – PRINCIPAL TAREFA.
 - Descobrir o que um sistema terá que fazer.

23

Capacidades

- Compreender conceitos abstratos, reorganizá-los em divisões lógicas e sintetizar ‘soluções’ baseadas em cada divisão.
- Absorver fatores pertinentes de fontes conflitantes ou confusas.
- Entender os ambientes do usuário/cliente.
- Aplicar elementos do sistema de hardware e/ou software aos elementos usuário/cliente.
- Comunicar-se **bem** nas formas **escrita** e **verbal**.

24

Características de um analista



Figura 30 Características desejáveis em um analista do sistemas.

25

UML

- Linguagem de Modelagem Unificada;
- A UML não é A/POO é apenas uma **notação padrão de diagramação**;
- Falaremos mais detalhadamente nas próximas aulas!!

26

POO – Princípios e Padrões

- Como as responsabilidades devem ser atribuídas as classes de objetos?
- Como os objetos devem interagir?
- Quais classes devem fazer o que?
- Para responder essas questões vamos trabalhar sobre a clássica metáfora de POO

Projeto guiado por responsabilidades!

27

Desenvolvimento Iterativo

- Iterativo - Diz-se do processo que se repete diversas vezes para se chegar a um resultado e a cada vez gera um resultado parcial que será usado na vez seguinte.

(<http://pt.wiktionary.org/wiki/iterativo>)

28

Desenvolvimento Iterativo

- Como um desenvolvedor ou uma equipe de desenvolvimento deve proceder para desenvolver um software, considerando as inúmeras atividades possíveis para as etapas de eliciar, analisar, projetar e implementar de um sistema?

29

Processo de desenvolvimento

- É um conjunto de atividades, e resultados associados, parcialmente ordenadas, com a finalidade de obter um produto de software.

SOMMERVILLE, 2003

30

Processos de desenvolvimento

- Modelos de Processos
 - Ágil
 - Iterativo
 - UP
 - RUP
 - Espiral
 - Cascata
 - XP
 - Scrum

31

Processos de desenvolvimento

- Em nossas aulas utilizaremos uma abordagem ágil ao Processo Unificado (PU), utilizado como exemplo de processo de desenvolvimento iterativo.
- O PU ágil não invalida sua aplicação em outros métodos ágeis como SCRUM e outros.

32

O que será visto na disciplina



Figura 1.1 Tópicos e habilidades abordadas.

33

Orientação a Objetos

34

Objetivos

- Exemplificar os elementos que compõem o paradigma Orientado a Objetos.
- Apresentar a UML e seus diagramas.

35

Orientação a Objetos

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados **objetos**.

Cada objeto é responsável por realizar tarefas específicas.

É através da interação entre objetos que uma tarefa computacional é realizada.

36

Orientação a Objetos

Um sistema de software orientado a objetos consiste de objetos em colaboração com o objetivo de realizar as funcionalidades deste sistema.

Cada objeto é responsável por tarefas específicas.

É através da cooperação entre objetos que a computação do sistema se desenvolve.

37

Conceitos e Princípios da OO

- Conceitos
 - Classe
 - Objeto
 - Mensagem
- Princípios
 - Encapsulamento
 - Polimorfismo
 - Generalização (Herança)
 - Composição

38

Classes, objetos e mensagens

- O mundo real é formado de coisas.
- Na terminologia de orientação a objetos, estas coisas do mundo real são denominadas *objetos*.
- Seres humanos costumam agrupar os objetos para entendê-los.
- A descrição de um grupo de objetos é denominada ***classe de objetos***, ou simplesmente de ***classe***.

39

O que é uma classe?

- Uma classe é um molde para objetos. Diz-se que um objeto é uma instância de uma classe.
- Uma classe é uma **abstração** das características **relevantes** de um grupo de coisas do mundo real.
 - Na maioria das vezes, um grupo de objetos do mundo real é muito complexo para que *todas* as suas características e comportamento sejam representados em uma classe.



Representante



Cliente

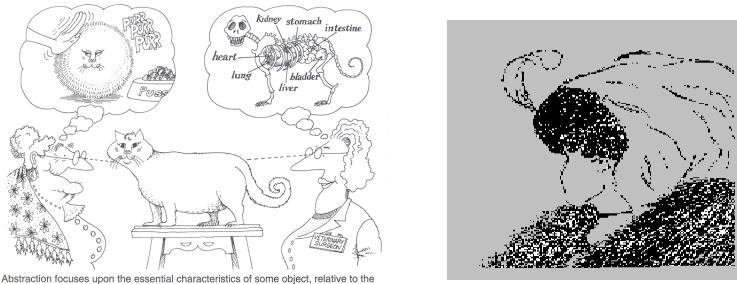


Produto

40

Abstração

- Uma abstração é qualquer modelo que inclui os aspectos relevantes de alguma coisa, ao mesmo tempo em que ignora os menos importantes. *Abstração depende do observador.*



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

41



42

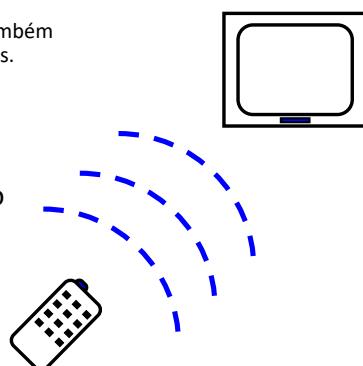
Mensagens

- Para que um objeto realize alguma tarefa, deve haver um estímulo enviado a este objeto.
- Pense em um objeto como uma entidade ativa que representa uma abstração de algo do mundo real
 - Então faz sentido dizer que tal objeto pode responder a estímulos a ele enviados
 - Assim como faz sentido dizer que seres vivos reagem a estímulos que eles recebem.
- Independentemente da origem do estímulo, quando ele ocorre, diz-se que o objeto em questão está recebendo uma **mensagem**.
- Uma mensagem é uma requisição enviada de um objeto a outro para que este último realize alguma operação.

43

Encapsulamento

- Objetos possuem **comportamento**.
 - O termo comportamento diz respeito a que operações são realizadas por um objeto e também de que modo estas operações são executadas.
- De acordo com o encapsulamento, objetos devem “esconder” a sua complexidade...
- Esse princípio aumenta qualidade do SSOO, em termos de:
 - Legibilidade
 - Clareza
 - Reuso



44

Encapsulamento

- O *encapsulamento* é uma forma de restringir o acesso ao comportamento interno de um objeto.
 - Um objeto que precise da colaboração de outro para realizar alguma tarefa simplesmente envia uma mensagem a este último.
 - O método (maneira de fazer) que o objeto requisitado usa para realizar a tarefa não é conhecido dos objetos requisitantes.

45

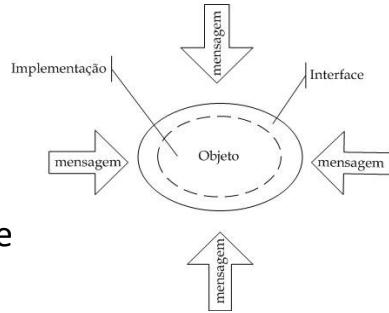
Encapsulamento

- Na terminologia da orientação a objetos, diz-se que um objeto possui uma *interface*.
 - A interface de um objeto é o que ele conhece e o que ele sabe fazer, sem descrever *como* o objeto conhece ou faz.
 - A interface de um objeto define os serviços que ele pode realizar e consequentemente as mensagens que ele recebe.

46

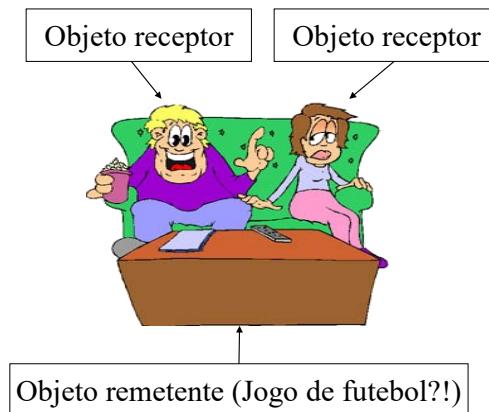
Encapsulamento

- Uma interface pode ter várias formas de **implementação**.
- Mas, pelo princípio do encapsulamento, a implementação utilizada por um objeto receptor de uma mensagem não importa para um objeto remetente da mesma.



47

Polimorfismo



48

Polimorfismo

- É a habilidade de objetos de classes diferentes responderem a mesma mensagem de diferentes maneiras.

49

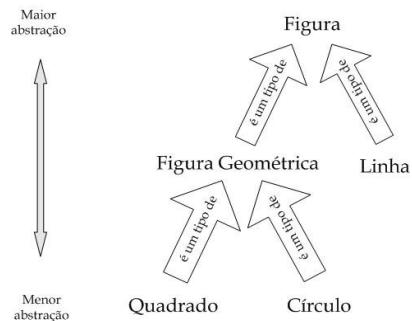
Generalização (Herança)

- A herança pode ser vista como um nível de abstração acima da encontrada entre classes e objetos.
- Na herança, classes semelhantes são agrupadas em hierarquias.
 - Cada nível de uma hierarquia pode ser visto como um nível de abstração.
 - Cada classe em um nível da hierarquia herda as características das classes nos níveis acima.

50

Herança

- A herança facilita o compartilhamento de comportamento entre classes semelhantes.
- As diferenças ou variações de uma classe em particular podem ser organizadas de forma mais clara.



51

Coesão e Acoplamento

- Os benefícios deste princípio são:
 - Facilitar a manutenção e modificação de uma determinada unidade.
 - Permitir a substituição de unidade por outra.
 - Diminuir a comunicação entre as unidades mais abstratas, o que pode significar um baixo tráfego na rede.



52

Coesão e Acoplamento

- O Princípio da Responsabilidade Única é um mantra a ser seguido, quando o assunto é fraco acoplamento e alta coesão.
- Quem faz tudo não faz nada. Diz o ditado.

Na coesão, qual é a função dos objetos abaixo:



53

Coesão e Acoplamento

- Coesão e acoplamento, são princípios de engenharia de software muito antigos, que embora sejam simples, muitas das vezes são ignorados. Por conta disto, os projetos de software acabam sendo prejudicados por um design ruim.
- O que estes princípios significam e como eles podem nos ajudar a ter um design maduro e eficiente em nosso software?

54

Coesão e Acoplamento

- São princípios de engenharia de software muito utilizados.
- Para ter uma arquitetura madura e sustentável, leva-los em conta, pois cada um tem um propósito específico que visa melhorar o design do software.
- O problema é não conhecer a diferença entre eles e não obter os benefícios de colocá-los em prática no desenho da arquitetura de um software.

55

Coesão e Acoplamento

- A Coesão está ligada ao princípio da responsabilidade única, (introduzido por Robert C. Martin no inicio dos anos 2000), e diz que:
 - uma classe deve ter apenas uma única responsabilidade e realizá-la de maneira satisfatória.
 - uma classe não deve assumir responsabilidades que não são suas .
- Uma vez ignorado este princípio, aparecem problemas como: dificuldades de manutenção e de reuso.

56

Coesão e Acoplamento

- Observe o exemplo:

```
public class Programa
{
    public void ExibirFormulario() {
        //implementação
    }

    public void ObterProduto() {
        //implementação
    }

    public void gravarProdutoDB {
        //implementação
    }
}
```

- **Esta classe não está coesa,** ou seja, **ela tem responsabilidades demais**, e o que é pior, **responsabilidades que não são suas.**
- Como visto no exemplo, a classe Programa tem responsabilidades que não são suas, como: obter um produto e gravá-lo no banco de dados.

57

Coesão e Acoplamento

- Observe outro exemplo:

```
public class Programa
{
    public void MostrarFormulario() {
        //Implementação
    }

    public void BotaoGravarProduto( ) {
        Produto.gravarProduto();
    }
}
```

- Neste exemplo, **há uma clara separação de responsabilidades**, o que contribui para um design desacoplado e organizado.
- **O formulário não assume o papel de cadastrar o produto, ele pede a quem tem a responsabilidade para que faça tal tarefa.**
- **A classe deve ser responsável por exercer uma única responsabilidade e fazer outras classes cooperarem quando necessário.**

58

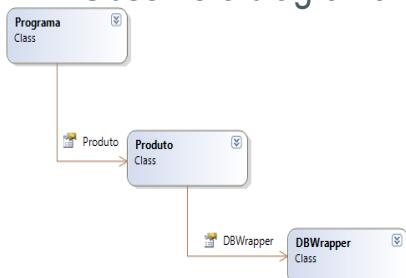
Coesão e Acoplamento

- O acoplamento significa o quanto uma classe depende da outra para funcionar.
- Quanto maior for esta dependência entre ambas, estas classes estão fortemente acopladas.
- O forte acoplamento também traz muitos problemas, problemas semelhantes aos de um cenário pouco coeso.

59

Coesão e Acoplamento

- Observe o diagrama:

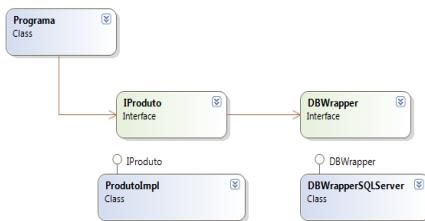


- Como pode ser observado na cadeia de classes, o forte acoplamento na mesma torna muito custosa a sua manutenção e o seu gerenciamento, pois **qualquer mudança afetará toda a cadeia de classes**.
- **A solução é a chamada Inversão de Controle.** É uma maneira de mudar este quadro. Deve inverter o controle e utilizar o padrão de injeção de dependência, para diminuir o acoplamento e evitar futuros problemas.

60

Coesão e Acoplamento

- Observe as alterações:

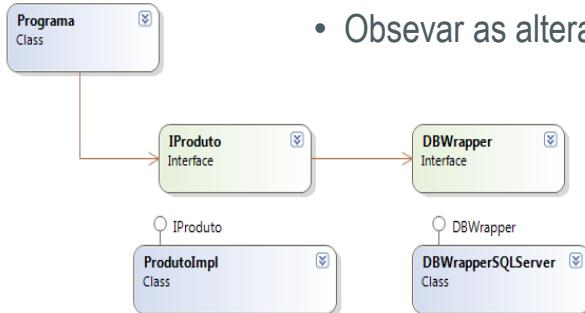


- Perceba como a dependência está na direção oposta, ou seja, não é mais de implementações concretas que estão baseados os relacionamentos entre as classes. Perceba que utilizar abstrações, mantém um cenário preparado para os impactos que as possíveis mudanças poderiam trazer.

61

Coesão e Acoplamento

- Observe as alterações:

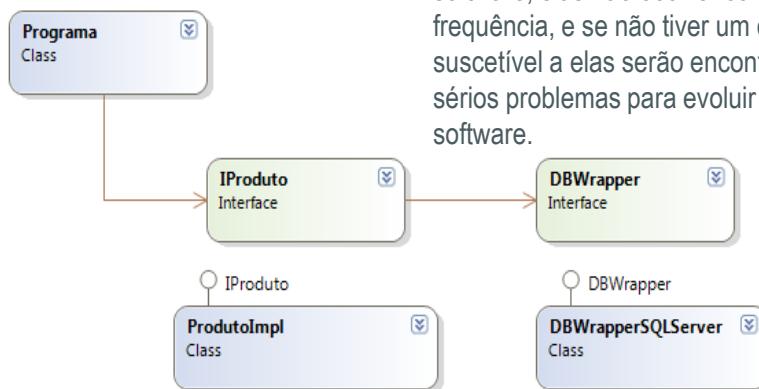


- Perceba que a classe Produto se relaciona com uma abstração(interface) de DBWrapper, ou seja, tem uma classe chamada DBWrapperSqlServer que implementa esta interface, caso o banco de dados for trocado e passar a ser oracle, basta adicionar uma classe DBWrapperOracle para atender a mudança de banco de dados.

62

Coesão e Acoplamento

- Observe as alterações:



63

Coesão e Acoplamento

- No mundo “real”, nem sempre é possível ter um cenário ideal, que é com um baixo acoplamento e uma alta coesão.
- É necessário saber tomar a decisão correta, pois determinadas decisões não poderão ser revertidas, dependendo da fase em que estiver o projeto ou o tipo de decisão tomada.
- **Ter classes** com responsabilidades claras e um baixo acoplamento, embora não seja fácil de serem construídas, nos traz benefícios como baixo impacto em uma possível manutenção, gerenciamento e mudança no negócio facilitados.

64

Coesão e Acoplamento

- Também não se deve esquecer que as aplicações evoluem, mudam e, muitas das vezes, se transformam.
- Se as melhores práticas forem ignoradas na hora de desenhar uma arquitetura, há possibilidades de ter sérios problemas.
- E para auxiliar, existe o princípio da responsabilidade única e da inversão de controle, cujo objetivo é obter um cenário de responsabilidades claras entre as classes e um baixo acoplamento.

65

Coesão e Acoplamento

- Em resumo:
 - Um dos princípios básicos para organização da estrutura e comportamento é **manter uma alta coesão e um baixo acoplamento entre as unidades**.
 - Uma unidade com alta coesão mantém unidades dependentes entre si agrupadas em uma unidade maior. Isto implica que unidades dependentes e com alto grau de comunicação e dependência entre si estejam localizadas numa mesma unidade mais abstrata.

66

O que é UML?

- UML - Linguagem de Modelagem Unificada
 - É uma linguagem visual para especificar, construir e documentar os artefatos dos sistemas.

67

O que é UML

- Mentores: Booch, Rumbaugh e Jacobson
 - “Três Amigos” - IBM Rational (www.rational.com)



Grady Booch



Ivar Jacobson



James Rumbaugh

68

O que é UML?

- UML **é**...
 - uma linguagem visual.
 - independente de linguagem de programação.
 - independente de processo de desenvolvimento.
- UML **não é**...
 - uma linguagem programação (mas possui versões!).
 - uma técnica de modelagem.

69

O que é UML?

- Um processo de desenvolvimento que utilize a UML como linguagem de modelagem envolve a criação de diversos documentos.
 - Estes documentos, denominados **artefatos de software**, podem ser textuais ou gráficos.
- Os artefatos gráficos produzidos de um sistema OO são definidos através dos **diagramas da UML**.

70

UML

- Três modos de aplicar a UML.

- UML como rascunho**

- Diagramas incompletos e informais (frequentemente rascunhados a mão), criados para explorar partes difíceis do problema ou espaço de solução, explorando o poder das linguagens visuais.

71

UML

- Três modos de aplicar a UML.

- UML como planta de software**

- Diagramas de projetos relativamente detalhados, utilizados para:
 - Engenharia reversa: para visualizar e entender melhor códigos existentes
 - Geração de código: para engenharia avante.

72

UML

- Três modos de aplicar a UML.

- **UML como linguagem de programação**

- Especificação executável completa de um sistema de software em UML. Código executável será automaticamente gerado.
 - Este uso requer um modo prático e detalhado de diagramar.
 - Necessita diagrama todo o comportamento (ou a lógica) do sistema em diagrama de estado e/ou interação.
 - Ainda está em desenvolvimento em termos de teoria, ferramentas robustas e usabilidade.

73

Modelagem Ágil

- A modelagem ágil enfatiza a UML como rascunho;
- Trata-se de um modo comum de aplicar a UML.
 - Alto retorno no investimento de tempo.

74

Manifesto Ágil

Manifesto para o desenvolvimento ágil de software

Descobrindo maneiras melhores de desenvolver software,
passamos a valorizar:

Indivíduos e interação entre eles mais que processos e ferramentas

Software em funcionamento mais que documentação abrangente

Colaboração com o cliente mais que negociação de contratos

Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

75

Diagramas da UML

- Um diagrama na UML é uma apresentação de uma coleção de **elementos gráficos** que possuem um significado predefinido.
 - No contexto de desenvolvimento de software, correspondem a desenhos gráficos que seguem algum padrão lógico.

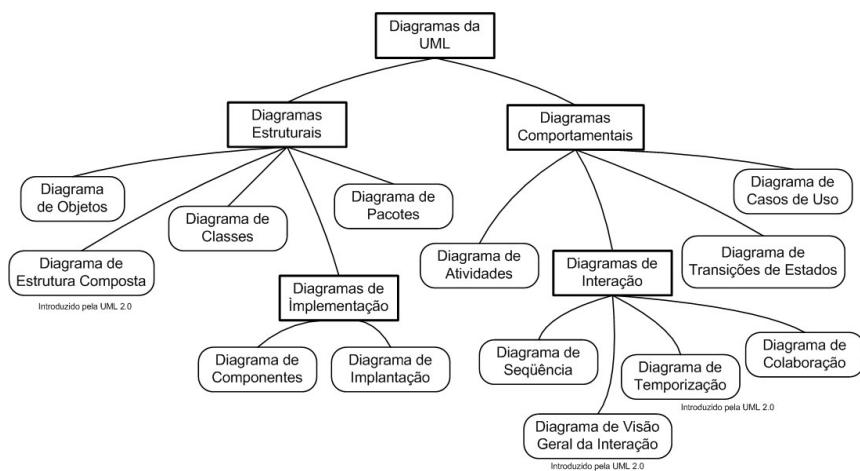
76

Diagramas da UML

- Um processo de desenvolvimento que utilize a UML como linguagem de modelagem envolve a criação de diversos documentos.
 - Estes documentos, denominados **artefatos de software**, podem ser textuais ou gráficos.
- Os artefatos gráficos produzidos no desenvolvimento de um SSOO são definidos através dos **diagramas da UML**.

77

Diagramas da UML 2.0



78

Perspectivas

- **Perspectiva Conceitual:** os diagramas são interpretados como descrevendo coisas em uma situação do mundo real ou domínio de interesse.

79

Perspectivas

- **Perspectiva de especificação (software):** os diagramas (usando a mesma notação da perspectiva conceitual) descrevem abstrações de software ou componentes com especificações e interfaces, mas nenhum comprometimento com a implementação. Por exemplo, não especifica uma classe em C# ou em Java.

80

Perspectivas

- **Perspectiva de implementação (software):** os diagramas descrevem implementações de software em uma tecnologia particular, tal como Java.

81

Perspectivas para aplicar UML

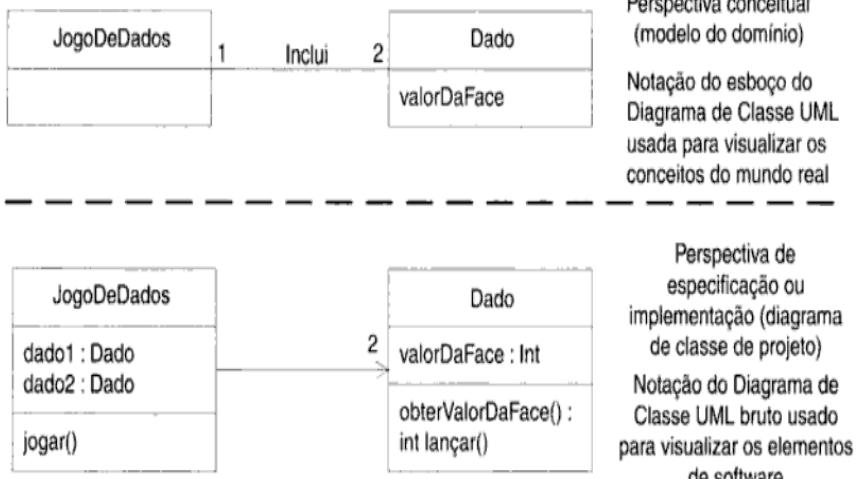


Figura 1.6 Diferentes perspectivas em UML.

Fonte: LARMAN (2008)

82

UML the Unified Modeling Language



1

83

Elementos gráficos

- Há três tipos de relacionamentos topológicos importantes:
 - conexão normalmente de linhas para formas 2-d
 - retenção de símbolos para formas 2-d com limites
 - anexo visual: um símbolo que está próximo a outro em um diagrama
- Há quatro tipos de elementos gráficos de construção:
 - ícones
 - símbolos 2-d
 - caminhos
 - texto

2

84

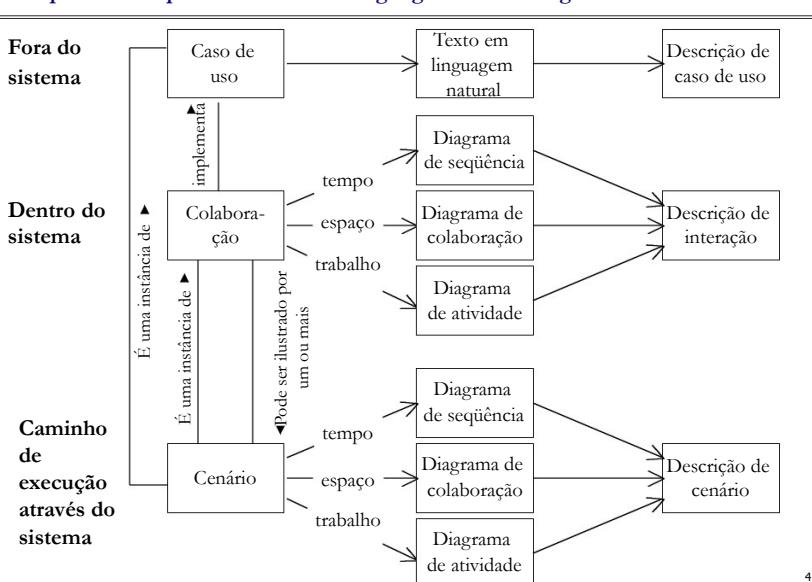
Diagramas da UML

- Diagrama de classe
- Diagrama de caso de uso
- Diagramas de interação:
 - Diagrama de seqüência
 - Diagrama de colaboração
- Diagrama de estado
- Diagrama de atividade
- Diagramas de implementação:
 - Diagrama de componente
 - Diagrama de implantação

3

85

Perspectiva Tipo de modelo Linguagem de modelagem Modelo



4

86



Notações genéricas da UML

5

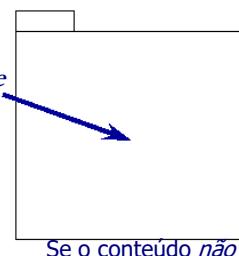
87

Pacote

- Mecanismo de propósito geral para organizar elementos de modelo em grupo
- São porções principais resultantes da subdivisão em limites de coesão lógica funcional
- Se for possível dar um nome a um grupo de classes, isso pode indicar a existência de um bom agrupamento para pacote
- Um pacote pode estar aninhado em outro pacote



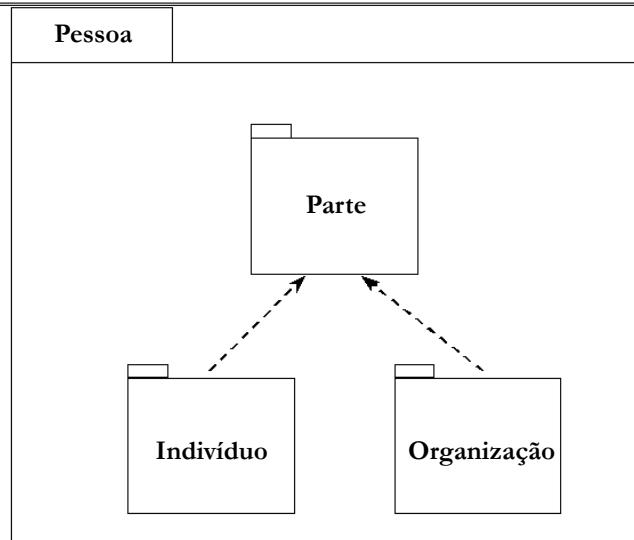
ou



6

88

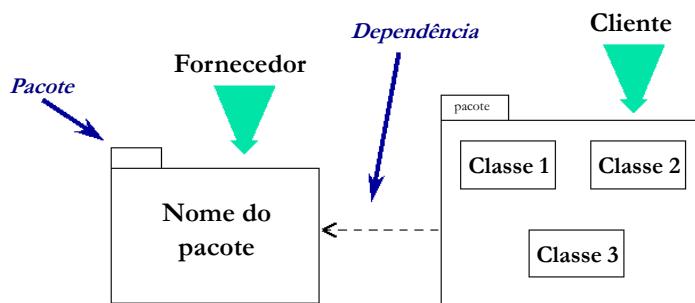
Pacote



7

89

Dependência de pacotes



Se uma classe em um pacote precisa utilizar uma classe em outro pacote pode-se desenhar uma dependência (dependências não são transitivas)

8

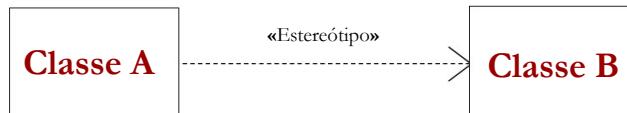
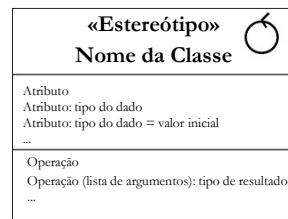
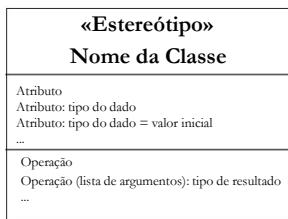
90

- Palavra sugerida por Rebecca Wirfs-Brock para criar uma metaclassificação de elementos na UML
- Introduz novos elementos no metamodelo para permitir que usuários estendam a capacidade de modelagem da linguagem
- Todo elemento na UML pode ter no máximo um estereótipo
- Estereótipos com semântica predefinida na UML são:
 - Classe e estereótipos de objeto;
 - Evento
 - Exceção
 - Interface
 - Metaclasse
 - Utilitário
 - Estereótipos de tarefa
 - Processo: tarefa de impacto
 - Thread: tarefa mais simples



9

91



10

92

- Um comentário colocado em um diagrama sem qualquer conteúdo semântico
- Aparece em diagramas particulares e pode ser anexada a zero ou mais elementos de modelagem através de linhas tracejadas
- É mostrada como um retângulo com canto curvado no canto direito superior contendo um texto arbitrário

Este diagrama foi elaborado
para preparação do livro
Modelagem de Objetos

11

93



Diagrama de Classe

12

94

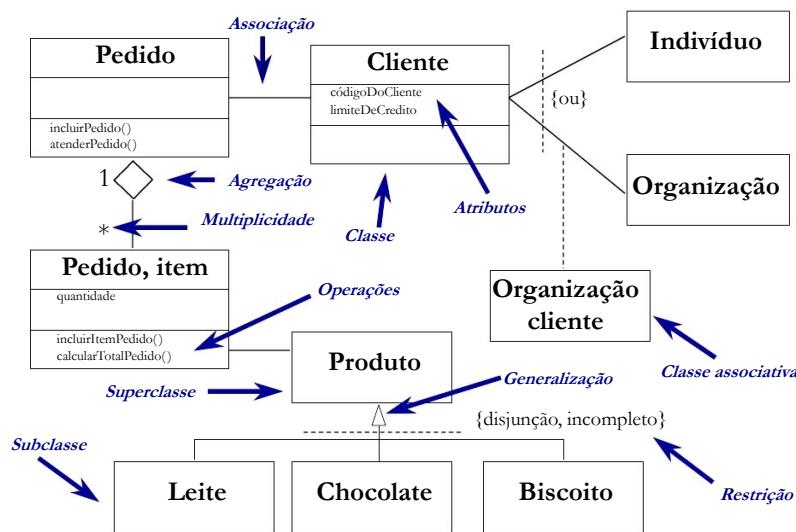
Diagrama de classes

- Essência da UML resultado de uma combinação de diagramas propostos por OMT, Booch e vários outros métodos
- Estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus conteúdos e relações
- Relacionamentos no diagrama de classes
 - **Generalização/especificação**
 - Indica relacionamentos superclasse/subclasse, também conhecidos como herança ou classificação (i.e. uma enfermeira é uma pessoa)
 - **Agregação**
 - Usada para denotar relacionamentos todo/parte (i.e. um item de compra é parte de um pedido)
 - **Associação**
 - Utilizada para denotar relacionamentos entre classes não correlatas (i.e. um cliente pode alugar várias fitas de vídeos)

13

95

Diagrama de classe

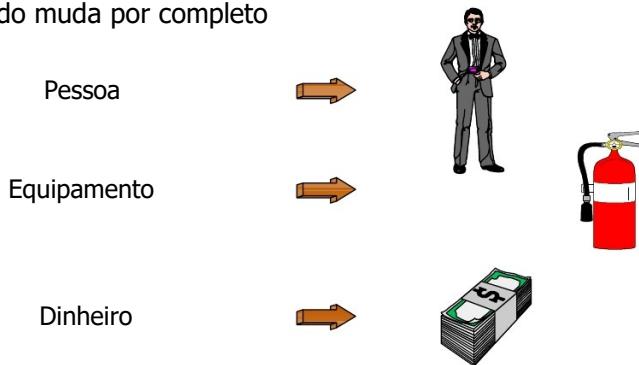


14

96

FACER Faculdade Objeto

- Uma unidade real ou abstrata, individualizada e identificável que modela um conceito presente na realidade humana, ocupando espaço físico (mundo físico) ou lógico (na memória)
- A identidade de um objeto é uma propriedade que o distingue de todos os demais, sendo preservada até mesmo quando seu estado muda por completo



15

97

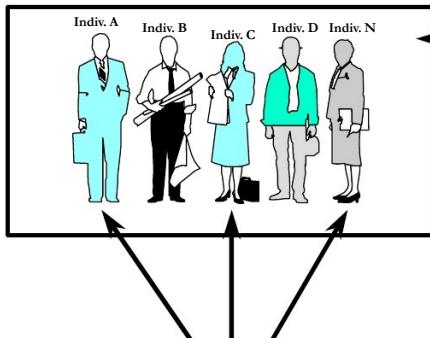
FACER Faculdade Classe

- Classe é a representação de um conjunto de coisas reais ou abstratas que são reconhecidas como sendo do mesmo tipo por compartilhar as mesmas características de atributos, operações, relações e semântica
- Uma descrição que relaciona algum número de objetos
- Retrato para um grupo arbitrário de objetos relacionados, mas não necessariamente idênticos

Nome da classe
atributo
atributo: tipo do dado
atributo: tipo do dado = valor inicial
...
operação
operação (lista de argumentos): tipo de resultado
...

16

98

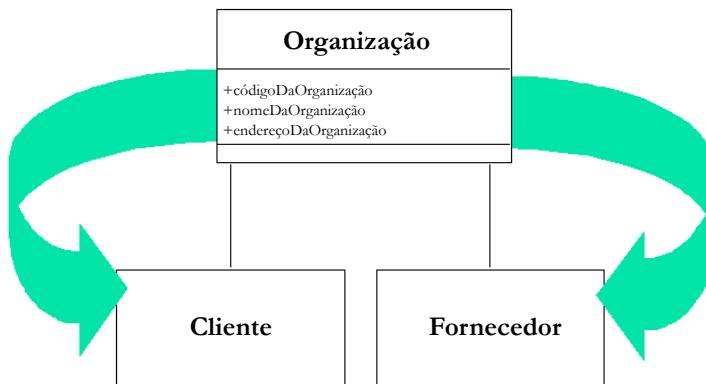
Classe Indivíduo

Classe = grupo de objetos similares que compartilham atributos e comportamento semelhantes

Objetos = ocorrências de uma classe

17

99



18

100

- Representam normalmente os substantivos no domínio de aplicação
- Classes grandes são mais difíceis de entender e reutilizar. Devemos criar classes de objetos que possuam propósitos bem definidos para ajudar na reutilização por outras aplicações
- Classes menos complicadas são mais fáceis de entender e manter, e se uma classe não puder ser explicada em algumas orações ou no máximo alguns parágrafos, ou ainda ter muitas operações ou se tornar confusa, são fortes candidatas a subdivisão em classes menores

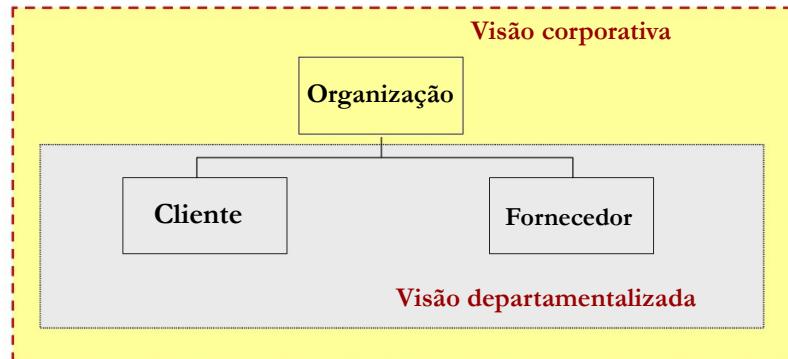
19

101

- Algo tangível e/ou visível, tal como pessoas, equipamentos, minerais
- Coisas intangíveis e/ou simbólicas, tais como qualidade, planejamento, responsabilidades
- Alguma coisa que possa ser concebida intelectualmente ou direciona-se ação ou pensamento
- Dispositivos ou sistemas exteriores com os quais a aplicação interage
- Eventos temporais ou externos

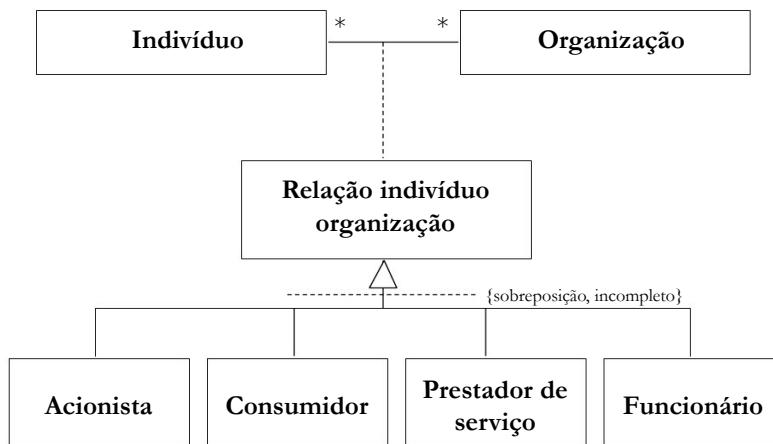
20

102



21

103



22

104

- Uma classe é desenhada como sendo uma caixa retangular com três compartimentos:
 - nome de classe no compartimento superior
 - uma lista de atributos no compartimento intermediário (pode ser suprimida)
 - uma lista de operações no compartimento inferior (pode ser suprimida).
- As linhas de separação dos compartimentos não pode ser suprimida se um dos comportamentos de atributos e operações estiver descrito
- Se um compartimento é suprimido, nenhuma conclusão pode ser assumida sobre a presença ou ausência desse elemento

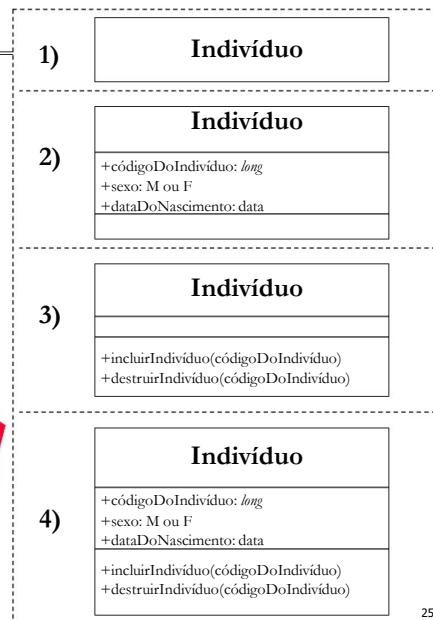
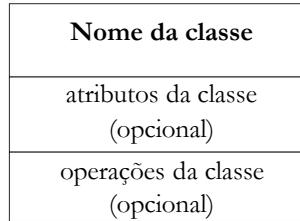
23

105

- Apresentar reticências (...) ao final de uma lista ou de uma seção delimitada de uma lista indica que existem elementos adicionais no modelo que satisfazem a condição de seleção mas isso não é mostrado naquela lista
- Dependendo da ferramenta de modelagem sendo empregada, podem ser fornecidos compartimentos adicionais como uma extensão à teoria para mostrar propriedades definidas pelo usuário, como por exemplo, regras empresariais, responsabilidades, variações, eventos manipulados entre outros

24

106

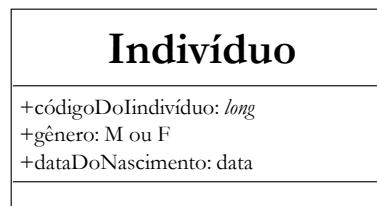


25

107

Atributo

- Estrutura de dado componente de uma classe
- É a menor unidade que em si possui significância própria através de um princípio de atomicidade
- Armazena um valor simples em uma célula
- Sintaxe padrão é:
 - Visibilidade NomeDoAtributo: TipoDeExpressão = ValorInicial {Propriedade}



26

108

Visibilidade de atributo

- + visibilidade pública (valor default)

 - Significa que todos tem acesso. Qualquer atributo (ou operação) que siga esta palavra-chave pode ser acessado por operações declaradas dentro de outras classes. É parte da interface de uma classe e é visível por qualquer parte do ambiente de referência da classe

- # visibilidade protegida

 - Significa que qualquer atributo (ou operação) que siga a palavra-chave pode ser acessado através de métodos dentro da mesma classe e por operações de classes ao longo do pacote no qual a classe é definida

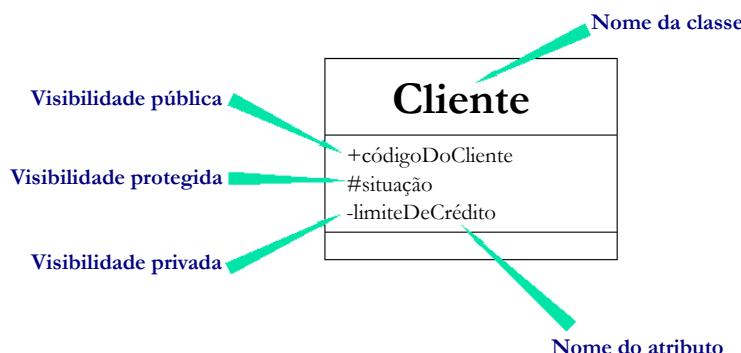
- - visibilidade privada

 - Declaração que é parte da interface de uma classe mas não é visível a quaisquer outras classes, ou seja, qualquer atributo (ou operação) que siga esta palavra-chave pode apenas ser acessado por operações declaradas dentro da mesma classe. Tudo, exceto declarações locais, é visível fora da classe; identificadores de componente privado são desconhecidos e inacessíveis

27

109

Visibilidade de atributo



28

110

- O marcador de visibilidade pode ser suprimido, mas sua ausência apenas indica que sua visibilidade não é mostrada e não que seja indefinida
- Alguns tipos adicionais de visibilidade podem ser definidos para suportar determinadas linguagens de programação, como é o caso da visibilidade de implementação para C++
- Quando o atributo for precedido por "/" indica que é um atributo derivado (informação), ou seja, um atributo que pode ser computado a partir de outro, mas é mostrado para elucidar melhor o modelo sem que agregue conteúdo semântico

29

111

- NomeDoAtributo
 - É uma sequência de caracteres de identificação. O nome do atributo tipicamente começa com letra minúscula
- TipoDeExpressão
 - É uma especificação que depende da linguagem de programação e do tipo de implementação de um atributo. A UML evita especificar regras para construção de tipos de expressão pois as linguagens de programação apresentam uma variedade de regras para declaração de variáveis e parâmetros. Com isso, a UML deixa indefinida a sintaxe real das expressões de tipo, passando a ser uma responsabilidade da ferramenta em uso verificar e analisar gramaticalmente tais expressões
- ValorInicial
 - É uma expressão que também dependente da linguagem de programação utilizada para o valor inicial de um objeto criado recentemente. O valor inicial é opcional (o sinal igual também é omitido)
- Propriedade
 - Possui características aplicáveis ao elemento, mas é opcional. Refere-se a um valor nomeado que denota uma característica de um elemento com impacto semântico. Certas propriedades são predefinidas na UML, outras podem ser definidas pelo usuário. Algumas propriedades úteis são:

30

112

- Propriedades do atributo
 - **Descrição do atributo**
 - Uma descrição sucinta e autocontida que representa claramente o seu propósito e alcance
 - **Tipo de dado e tamanho do atributo**
 - int, short, long, float, double, byte, boolean e char
 - **Tipo de atributo**
 - Estático (único valor compartilhado por todos os objetos da classe), constante (valor imutável), atributo normal (contém um valor que é variável/modificável), matriz (do tipo itemDeCompra[10], atributo final (atributo constante que não pode ser mudado -- usado em Java)
 - **Domínio de valores**
 - Valor mínimo e valor máximo (finito ou infinito) e restrição

31

113

- Para individualizar e identificar cada objeto de uma classe, é assinalado um identificador (identidade ao objeto) que será o ponto de referência para sua distinção de todos os demais dentro da classe
- O identificador (ou OID – Object Identifier) é atribuído automaticamente a objetos pelo gerenciador de banco de objetos para identificar unicamente uma ocorrência em particular
- Os identificadores também atuam como atributos de referência para representar associações entre objetos, da mesma forma como fazem as chaves estrangeiras em modelos relacionais
- Uma diferença entre atributo de referência e chave estrangeira é o fato de que este está associado a valores visíveis ao usuário, enquanto aquele não está

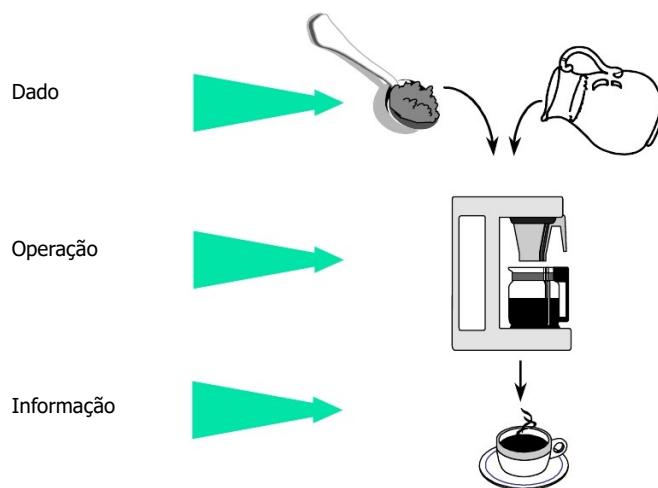
32

114

Informação	Dados
Idade: 40 anos	Data do nascimento, Data do dia
Valor total fatura: R\$ 1.000,00	Preço unitário, Quantidade
13 km/l	Km percorrido, Métrica de distância, Consumo , Métrica de volume
Quente	Medição, Métrica de temperatura, Faixa de temperatura
Longe	Medição, Métrica de distância, Faixa de distância

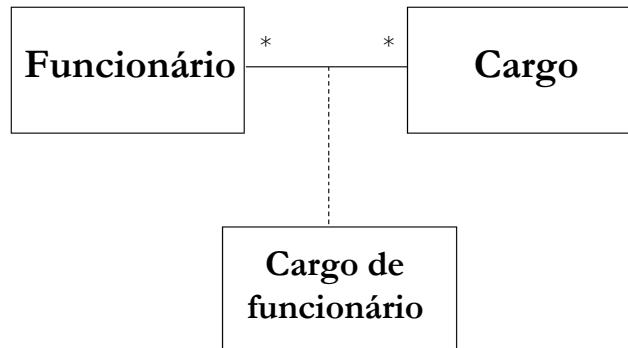
33

115



34

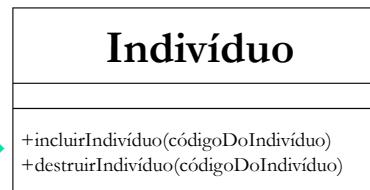
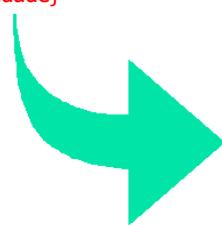
116



35

117

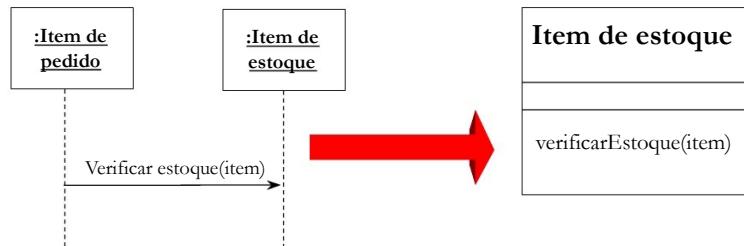
- Um serviço de classe ou comportamento resultante de um procedimentos algorítmico
- São executadas operações sempre que um objeto recebe uma mensagem de outro objeto
- As operações estão relacionadas aos verbos associados aos substantivos para fornecer as ações de tratamento de dados em resposta a evento
- A sintaxe padrão é:
 - Visibilidade NomeDaOperação (Parâmetro) : ExpressãoDeTipoDeRetorno {Propriedade}



36

118

- O comportamento de uma classe é representado por suas operações
- Operações podem ser encontradas examinando-se diagramas de interação



37

119

Visibilidade

- Definida da mesma forma que para atributos, ou seja:
 - + visibilidade pública (valor default)
 - # visibilidade protegida
 - - visibilidade privada
- Nome da classe**
- Nome do atributo**
- Visibilidade pública**
- Visibilidade protegida**
- Visibilidade privada**
-
- ```

classDiagram
 class Cliente {
 +obterLimiteDeCrédito()
 #bloquearCliente()
 -cancelarCliente()
 }

```
- The diagram shows a UML class named `Cliente` with three operations: `+obterLimiteDeCrédito()`, `#bloquearCliente()`, and `-cancelarCliente()`. Three green arrows point to the class from the left, each labeled with a visibility level: `Visibilidade pública` (public), `Visibilidade protegida` (protected), and `Visibilidade privada` (private). A green arrow also points from the bottom right to the class, labeled `Nome do atributo` (attribute name).

38

120

- NomeDaOperação
  - É uma seqüência de identificação da operação em si
- Parâmetro
  - É uma lista de valores separada por vírgula de parâmetros formais. Trata-se de um especificação de uma variável que pode ser mudada, passada ou devolvida, podendo incluir um nome, tipo e direção. Parâmetros são usados para operações, mensagens e eventos
- ExpressãoDeTipoDeRetorno
  - É uma especificação dependente de linguagem de programação sobre o tipo de implementação do valor retornado pela operação. Se o tipo de retorno é omitido a operação não devolve um valor

39

121

- Propriedade
  - Indica valores de propriedade que se aplicam ao elemento, sendo opcional (as chaves são omitidas se nenhuma propriedade for especificada)
  - Algumas propriedades podem ser:
    - classificação
    - precondição
    - pós-condição
    - tipo de exceção
    - concorrência
    - transformação
    - estereótipo

40

122

## Características de propriedade de operação

- Classificação declara a categoria da operação, por exemplo, modificador, seletor, de interação, construtor, destrutor:
  - um modificador é uma operação que altera o valor ou estado de um objeto
  - um seletor é uma operação que tem acesso, mas não pode alterar, o valor ou estado de um objeto
  - de interação é uma operação que permite visitar as partes de um objeto
  - um construtor é uma operação que cria e/ou inicializa um objeto
  - um destrutor é uma operação que destrói/desabilita um objeto

41

123

## Características de propriedade de operação

- Precondição declara uma condição que deve ser satisfeita para uma transformação correta; se a Precondição falha, uma exceção pode ser levantada para indicar que a operação não poderia ser executada corretamente
- Pós-condição declara uma condição que assegura que a transformação aconteceu corretamente; se a pós-condição falha, uma exceção pode ser levantada para indicar que aconteceu uma transformação com erro;
- Tipo de exceção é um tipo ou classe da exceção que é levantada pela operação
- Concorrência declara se a operação é uma operação seqüencial ou simultânea; uma operação seqüencial é um membro de uma classe seqüencial; uma operação simultânea é um membro de uma classe simultânea
- Transformação declara a fórmula ou expressão da operação;
- Estereótipo declara o tipo descritivo da operação, por exemplo, modificador, de acesso etc

42

124

## Características de propriedade de operação

- Propriedades dependentes de linguagem de programação:
  - Uma operação virtual indica que a operação pode ser anulada em uma subclasse
  - Uma operação virtual pura indica que a classe é uma classe abstrata em C++
  - Uma operação estática é uma operação de classe que pode modificar um valor de atributo estático; é invocada através de chamada de classe e não objeto
  - Uma operação de const indica que a operação não vai modificar algum valor de atributo
  - Um procedimento é uma operação que pode ou não devolver um valor (uma função é uma operação que devolve um valor)
  - Uma operação amiga pode ser acessada por membros de outra classe, por exemplo uma classe amiga (C++)
  - Uma operação inline tem código ampliado ao ponto de uma chamada para melhoria de desempenho (C++)
  - Uma operação de evento manipula evento especial em Delphi, por exemplo, um evento definido pelo usuário
  - Uma operação de sobreposição fornece uma operação de substituição em uma subclasse que sobrepõe uma operação em um superclasse (Delphi)
  - Uma operação final não pode ser anulada em uma subclasse (Delphi)
  - Uma operação nativa é implementada fora de Java
  - Uma operação sincronizada (synch) pode ser acessada simultaneamente, fornecendo um bloqueio para assegurar que apenas um de cada vez invoque a operação (Java)

43

125

## Ao definir uma operação pergunte para si mesmo:

- A operação está fazendo apenas uma coisa?
  - Lembre-se que operações menores são mais reutilizáveis
- As responsabilidades estão com as classes corretas?
  - Um bom ajuste aumenta a probabilidade para subclasses futuras utilizarem novamente os serviço

44

126

- Associação é uma relação que descreve um conjunto de vínculos entre elementos de modelo
- Quando duas classes, ou mesmo uma classe consigo própria, apresenta interdependência onde determinada instância de uma delas origina ou se associa a uma ou mais instâncias da outra, dizemos que elas apresentam uma associação
- São desenhadas como linhas sólidas entre pares de classes
- O fim de uma associação onde se conecta uma classe é chamado de papel da associação; a maioria da informação interessantes sobre uma associação é anexada a seus papéis

45

127

- Associação unária
  - Quando há um relacionamento de uma classe para consigo própria, também conhecido como associação reflexiva
  - Na associação unária são conectados ambos os fins à mesma classe, mas os dois fins são distintos

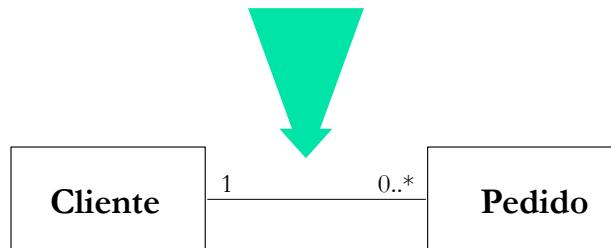


46

128

- Associação binária

- Quando há duas classes envolvidas na associação de forma direta de uma para a outra
- Uma associação binária é desenhada como um caminho sólido que conecta dois símbolos de classe, sendo que tal caminho pode consistir em um ou mais segmentos conectados



47

129

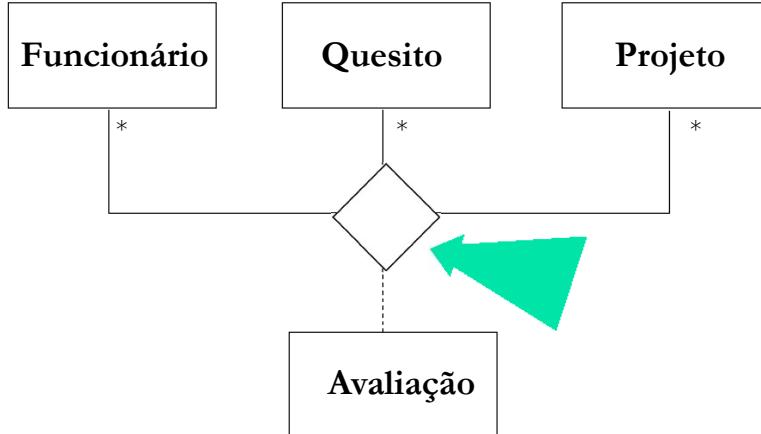
- Associação n-ária

- É uma associação entre 3 ou mais classes, mas uma única classe pode aparecer mais de uma vez
- Associações ternárias ou superiores são mostradas como diamantes conectados ao símbolo de classe através de linhas com um caminho do diamante para cada classe participante
- Um símbolo de classe de associação pode ser anexado ao diamante por uma linha tracejada
- Normalmente as linhas são desenhadas dos pontos no diamante ou o ponto central de um lado o que indica uma associação n-ária que tem atributos, operações e associações de "e/ou". Uma associação entre duas classes (associação binária) é um caso particular de associação n-ária

48

130

## Associação ternária

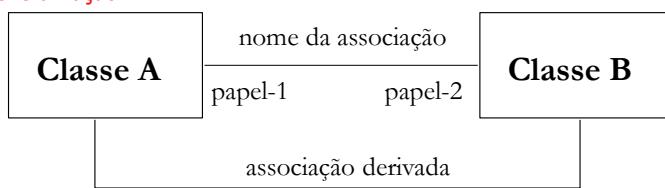


49

131

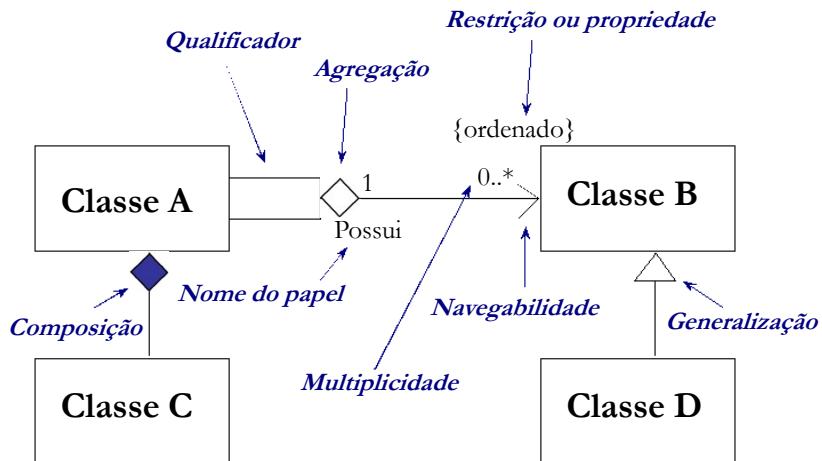
## Papéis em associações

- O fim de uma associação onde se conecta uma classe é chamado de papel da associação
  - A maioria dos detalhes interessantes sobre uma associação é tida em seus papéis que, além de um nome, podem apresentar informações de:
    - multiplicidade
    - ordenação
    - qualificador
    - agregação/composição
    - navegabilidade
    - generalização



50

132



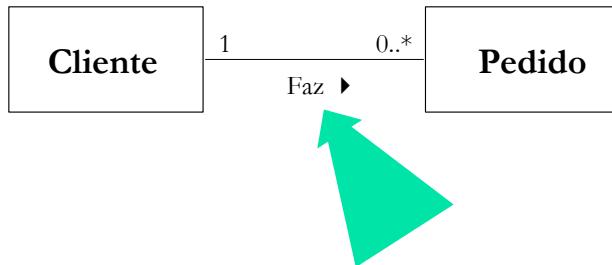
51

133

- A associação possui um nome próximo à linha que representa a associação, freqüentemente atribuído como um verbo, muito embora também possam ser utilizados substantivos
- Quando um diagrama de classe é construído, deve refletir o sistema que está sendo modelado, o que significa que os nomes de associação necessitam pertencer ao domínio do problema tal como ocorre com o nome das classes
- É possível utilizar uma associação navegável ao adicionar uma seta ao final da associação
- A seta indica que a associação somente pode ser utilizada na direção indicada, entretanto, associações podem ter dois nomes, um para cada direção
- A direção do nome é mostrada através de um triângulo sólido pequeno precedendo ou sucedendo o nome da associação, dependendo da direção, de forma que possa ser lida a associação de uma classe a outra
- Por exemplo, "um cliente faz pedido" (faz ▶ )

52

134



53

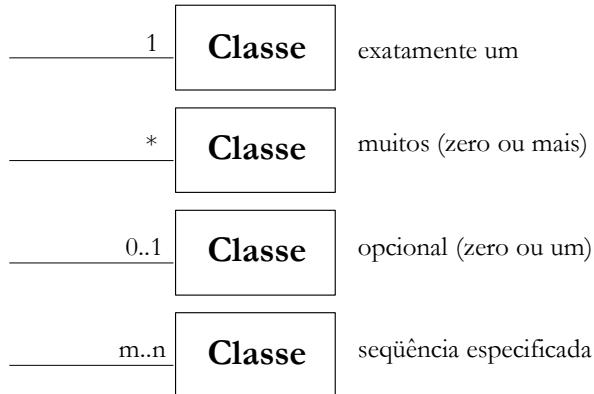
135

- Um dos aspectos chaves em associações é a cardinalidade de uma associação, chamada na UML de multiplicidade
- Especifica, por exemplo, em quantas companhias uma pessoa pode trabalhar, quantos dependentes pode ter um indivíduo etc,
- corresponde à noção de obrigatório, opcional, um-para-muitos, muitos-para-muitos similar ao enfoque de entidade/relacionamento
- A cardinalidade é especificada para cada extremidade na associação



54

136



55

137

- É mostrada como uma seqüência de intervalos inteiros separada por vírgula onde um intervalo representa um alcance de inteiros ou mesmo um intervalo infinito no formato limite inferior/límite superior
- Um asterisco (\*) pode ser usado para o limite superior denotando valor ilimitado
- Se a especificação de multiplicidade inclui somente asterisco, denota o alcance de inteiro ilimitado, o que é equivalente a \*..\* ou 0..\*
- Intervalos devem vir preferencialmente ordenados: "1..3,7,10" é preferível a "7,10,1.. 3"
- dois intervalos contíguos devem ser combinados em um único intervalo: "0..1" é preferível a "0,1"

56

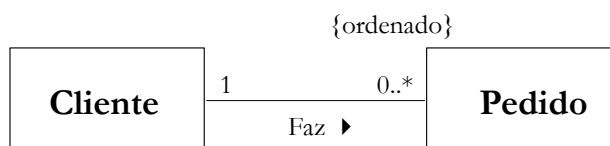
138

| Multiplicidade      | Significado                                     |
|---------------------|-------------------------------------------------|
| 0.. 1               | zero ou um                                      |
| 1                   | somente 1                                       |
| 0..*                | maior ou igual a zero                           |
| *                   | maior ou igual a zero                           |
| 1..*                | maior ou igual a 1                              |
| 1..15               | de 1 a 15, inclusive                            |
| 1..2,7..15,19,23..* | de 1 a 2, de 7 a 15, 19 e 23 e acima, inclusive |

57

139

- Se a multiplicidade é maior do que um, o conjunto de elementos relacionados pode ser ordenado ou não ordenado (este é o padrão e não necessita ser mostrado explicitamente)
- Os elementos são ordenados em uma lista, cuja especificação genérica inclui todos os tipos de ordenação podendo ser especificado por uma restrição de palavra-chave: {ordenado}

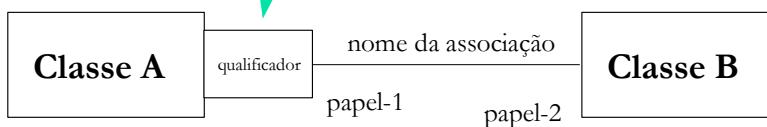


58

140

## Qualificador de associação

- Um qualificador é um atributo de associação ou estrutura de atributos de atributos cujos valores particionam um conjunto de objetos relacionados a um objeto por uma associação
- É mostrado como um retângulo pequeno, menor do que o retângulo fixo de classe, anexado ao fim do caminho de associação antes do símbolo da classe à qual se conecta



59

141

## Exemplo de associação qualificada



60

142



## Agregação

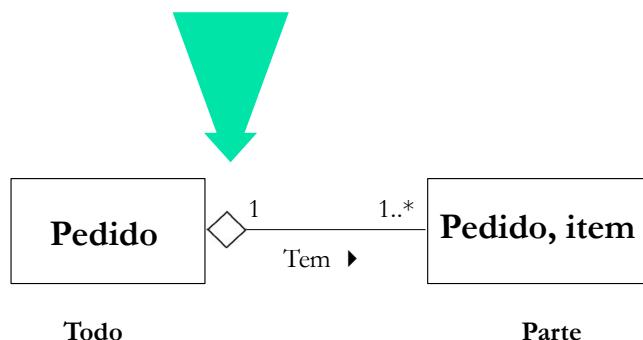
- É uma forma especial de associação utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro numa relação de todo/parte
  - Por exemplo, um pedido é composto por itens de pedido
- Indica que a vida das partes é dependente da vida do todo
- Os objetos partes não podem ser criados a menos que o objeto todo ao qual estão agregados seja criado
- Objetos partes não podem ser destruídos por qualquer objeto diferente do objeto de agregação que o criou em primeiro lugar, condição esta não verdadeira para objetos que emanam de um relacionamento de associação regular

61

143



## Agregação



62

144

## Agregação de composição

- Composição é uma forma de agregação com uma forte propriedade e vida coincidente da parte com o todo
- Um diamante vazio é anexado ao fim do caminho e próximo à classe que representa o todo no relacionamento para indicar agregação regular ou por-referência
- Se o diamante está cheio, significa uma forma forte de agregação conhecida como composição ou relacionamento por valor

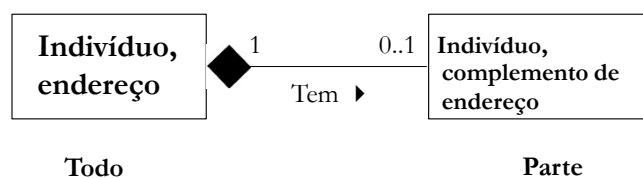


63

145

## Agregação de composição

- Na composição, ou agregação por valor (diamante cheio), o objeto todo declara uma instância real do objeto parte dentro do seu próprio corpo tornando o objeto parte fisicamente nele contido
- É semanticamente equivalente a um atributo, mas pode ser visualmente mais atraente quando a parte tem sua própria estrutura interna
- O fim de um objeto todo na agregação acarreta o fim de seus partes



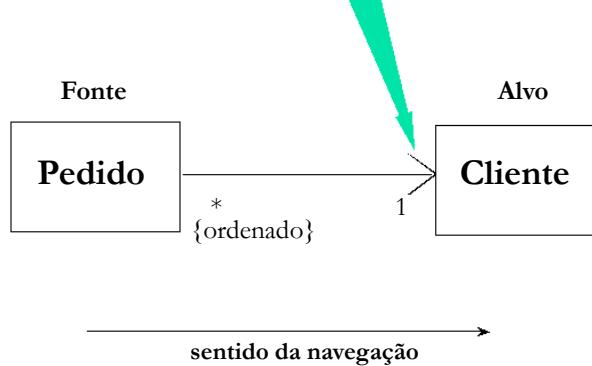
64

146

- Uma instância de uma classe pode navegar a instâncias de outra classe e vice-versa
- Navegabilidade é percebida freqüentemente por objetos que mantêm referências de algum tipo entre objetos associados
- Uma seta pode ser anexada ao fim do caminho para indicar que aquela navegação é suportada para a classe anexada à seta, assim como podem ser anexadas setas para zero, um, ou dois fins de caminho
- Em princípio, poderiam ser mostradas setas sempre que a navegação for suportada em uma determinada direção, mas na prática é às vezes conveniente suprimir algumas das setas e só exibir situações excepcionais

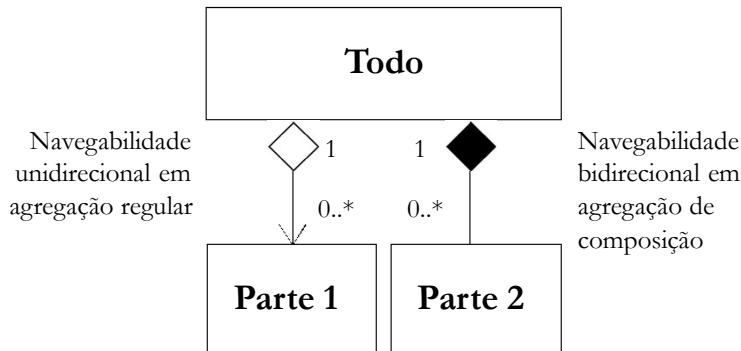
65

147



66

148



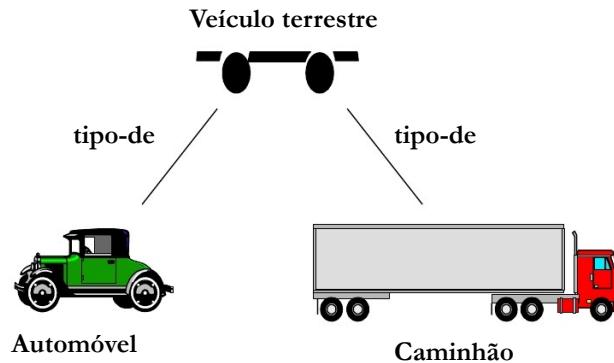
67

149

- Uma superclasse é criada para representar a essência da idéia sobre subclasses expressando os modos diferentes que determinadas essências podem ser percebidas
- A generalização é um relacionamento de taxinomia entre um elemento mais geral e um elemento mais específico que é completamente consistente com o primeiro elemento somando-o informação adicional especializada
- É usada para classes, pacotes, caso de usos e outros elementos
- Indica que uma classe mais geral, a superclasse, tem atributos, operações e associações comuns que são compartilhados por classes mais especializadas, as subclasses
- Por sua vez, as subclasses agregam atributos e operações particulares ao elemento de especialização que se referem: um objeto da classe é um tipo-de objeto da superclasse
- Por exemplo, uma superclasse [Veículo Terrestre] define atributos, operações e associações comuns às subclasses [Automóvel] e [Caminhão]

68

150



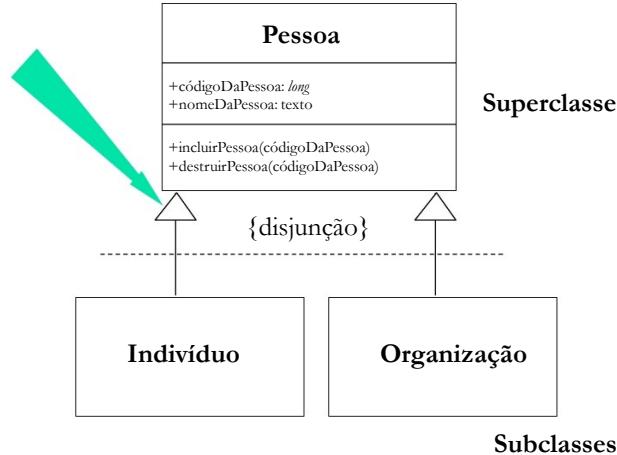
69

151

- É mostrada como uma linha sólida do elemento mais específico, uma subclasse, para o elemento mais geral, uma superclasse, com um triângulo vazio ao término do caminho que satisfaz o elemento mais geral
- Um grupo de caminhos de generalização para uma determinada superclasse pode ser mostrado como uma árvore com um segmento compartilhado, inclusive triângulo, para a superclasse, ramificando-se em caminhos múltiplos a cada subclasse
- Se vários arcos de generalização compartilham a mesma etiqueta, eles representam a mesma dimensão de generalização de superclasse; etiquetas diferentes definem dimensões independentes de generalização

70

152



71

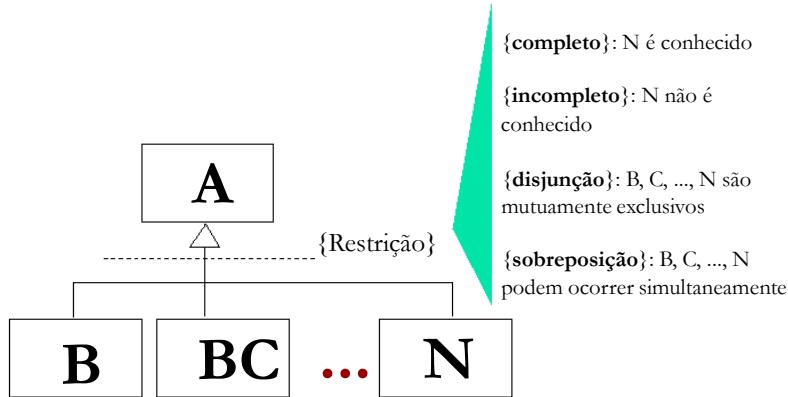
153

- Podem ser usadas restrições predefinidas para indicar condições semânticas entre subclasses
- Uma lista de palavras-chaves separada por vírgula é colocada em chaves próxima ao triângulo compartilhado, no caso de vários caminhos compartilharem um único triângulo, ou então próxima a uma linha pontilhada que cruza as linhas de generalização envolvidas
- As palavras-chaves seguintes são predefinidas na UML e podem ser usadas:

72

154

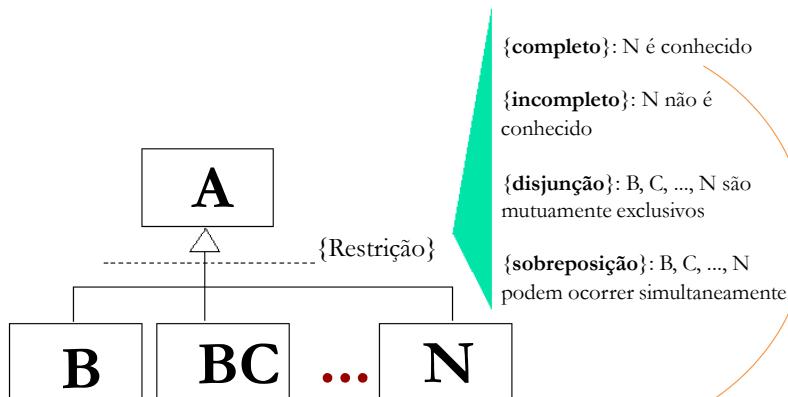
## Restrições para generalização/especificação



73

155

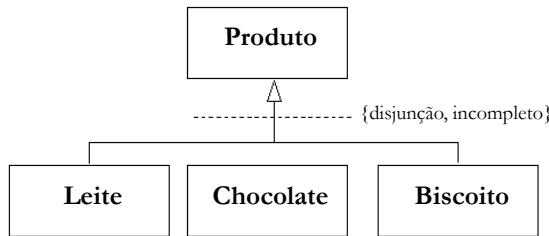
## Restrições para generalização/especificação



74

156

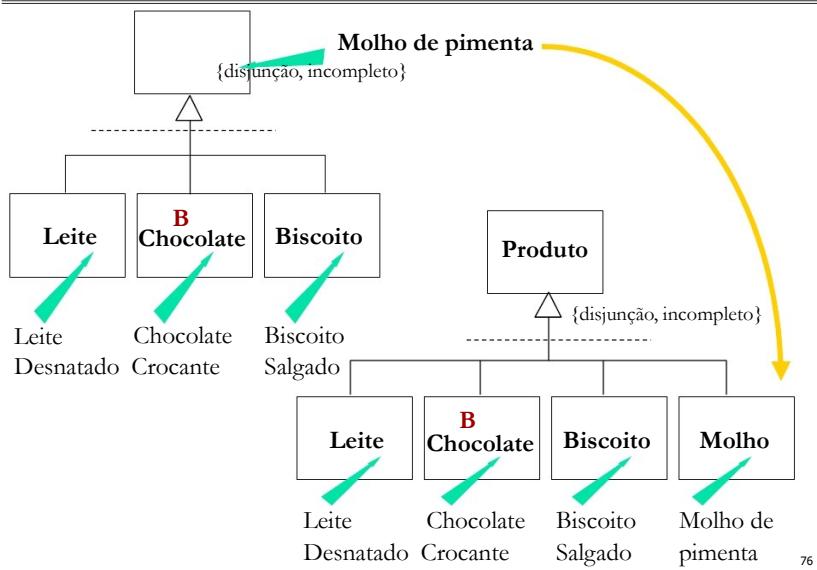
## Exemplo de generalização/especificação com restrição



75

157

## Nova subclasse



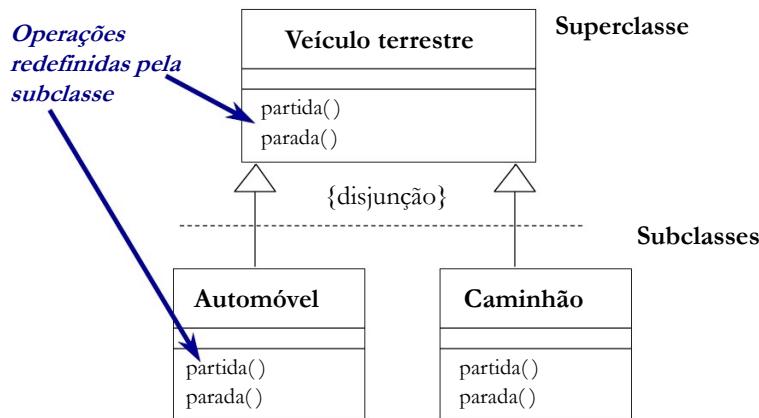
76

158

- Mecanismo de reutilização de atributos e operações pelo qual elementos mais específicos incorporam estrutura e comportamento de elementos mais gerais relacionados por comportamento
- Pode-se organizar tipos similares de classes de objetos em categorias chamadas hierarquia de classes, onde a classe de menor nível, que é uma especialização ou extensão de outra classe, pode utilizar serviços das classes superiores na hierarquia
- Permite a uma subclasse compartilhar atributos e operações definidas em sua superclasse

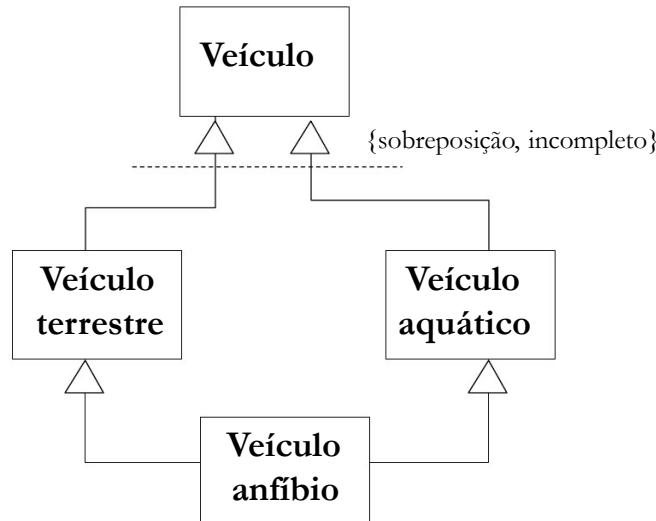
77

159



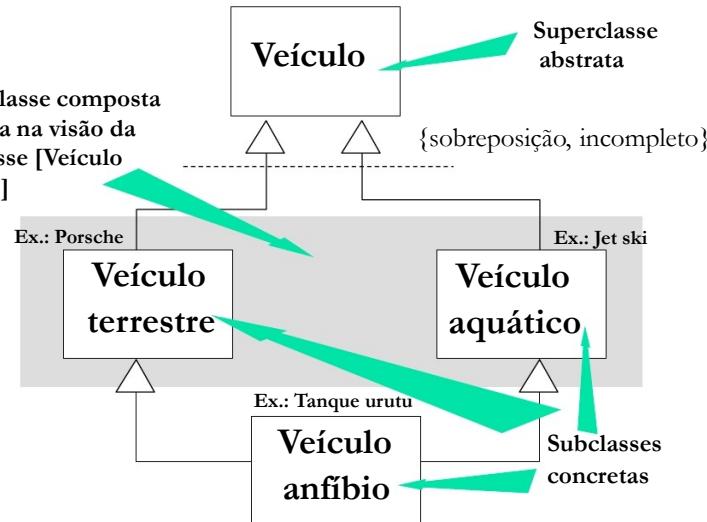
78

160



79

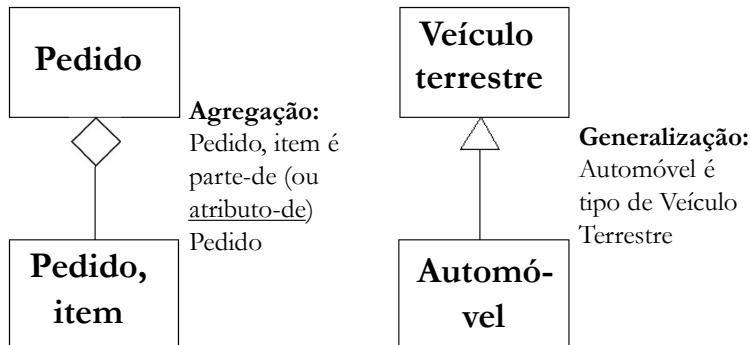
161



80

162

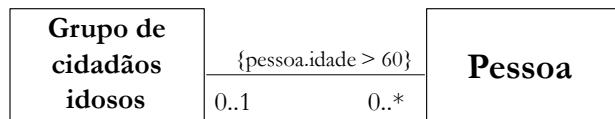
## Agregação versus generalização



81

163

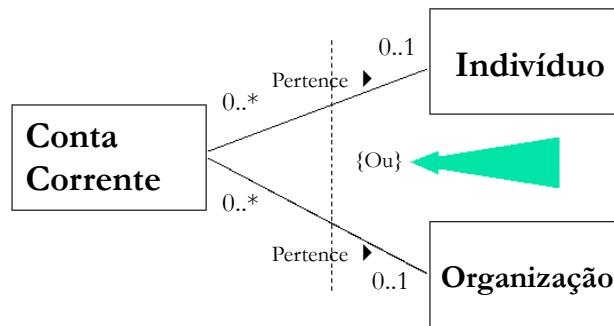
- Uma restrição é um relacionamento semântico entre elementos de modelo que especifica condições e proposições que devem ser mantidas como verdadeiras, caso contrário o sistema descrito pelo modelo é nulo, com consequências que estão fora do escopo da UML
- A restrição “ou” em associações, já é predefinida; outras restrições podem ser definidas normalmente pelos usuários em palavras cuja sintaxe, consistência e interpretação passam a ser de responsabilidade da ferramenta de modelagem em uso



82

164

## Restrição - Exemplos



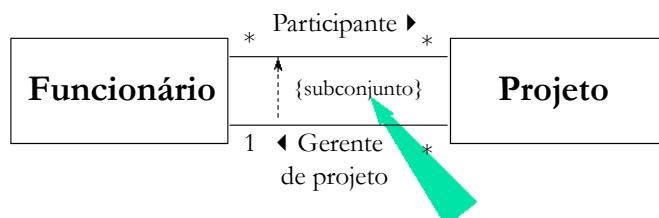
{Se nível de crédito for insuficiente então condição de pagamento deve ser à vista}



83

165

## Restrição - Exemplos

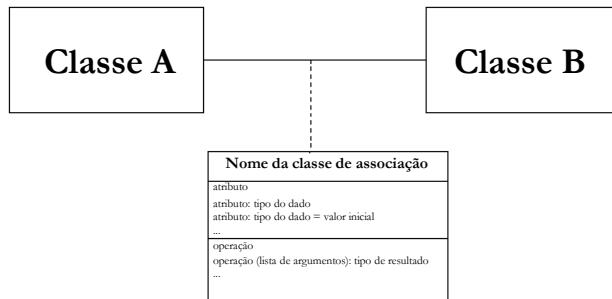


84

166

## Classe de associação

- Uma classe de associação é um elemento de modelagem que tem associação e propriedades de classe
- Pode ser vista tanto como uma associação que tem propriedades de classe como uma classe que tem propriedades de associação
- Embora seja desenhada como uma associação e uma classe, é apenas um único elemento do modelo e tem só um nome



85

167

## Classe de associação

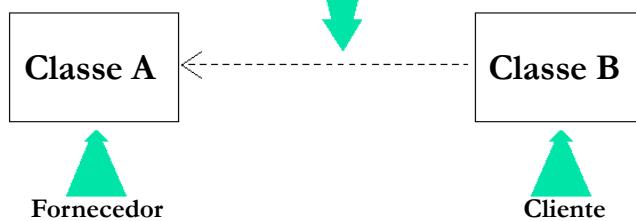
- Uma classe de associação é mostrada como um símbolo de classe anexado por uma linha tracejada a um caminho de associação
- Se uma classe de associação tem só atributos mas nenhuma operação ou outras associações, o nome pode ser exibido no caminho de associação e pode ser omitido do símbolo de classe de associação para enfatizar sua natureza de associação
- Se tem operações e outras associações, o nome pode ser omitido do caminho e pode ser colocado no retângulo de classe para enfatizar sua natureza de classe

86

168

## Dependência

- Indica a ocorrência de um relacionamento semântico entre dois ou mais elementos do modelo de forma que uma classe cliente é dependente de alguns serviços da classe fornecedora, mas não tem uma dependência estrutural interna com o fornecedor
- Uma mudança em um elemento (elemento independente) pode afetar outro elemento da dependência (elemento dependente)
- É mostrada como uma seta tracejada de um elemento de modelo para outro, do fornecedor para o cliente (classe apontada pela seta)



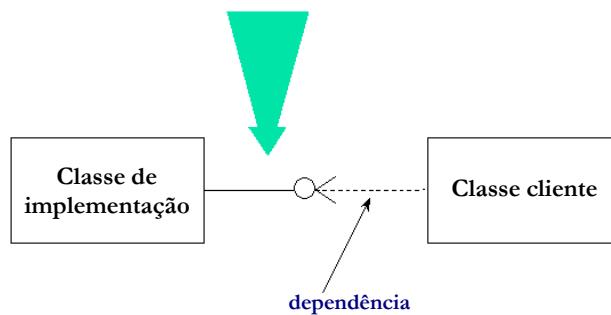
87

169

- Uma interface é um estereótipo de classe que designa a face externa de uma classe ou um pacote e consiste de um conjunto de operações e suas assinaturas juntamente com sua respectiva semântica dinâmica
- descreve os padrões legais de interação entre dois objetos
- A UML provê um ícone de estereótipo para mostrar interfaces em forma compacta.
- O fornecedor de uma interface (i.e., classe, pacote ou sistema inteiro que conforma o papel de "fornecedor") mostra um símbolo sólido de "pirulito" etiquetado pelo nome da interface
- O cliente de uma interface mostra uma seta de dependência tracejada ao círculo de interface

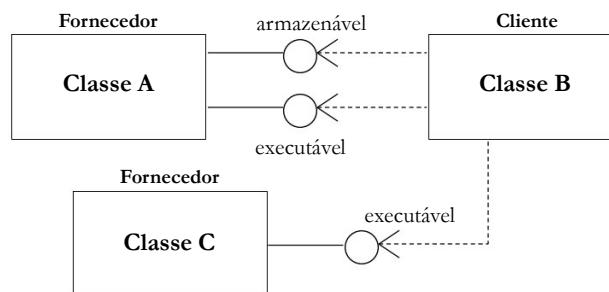
88

170



89

171



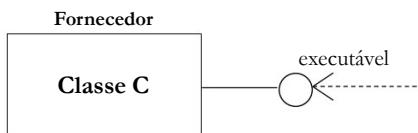
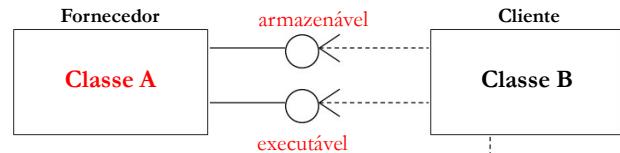
|                       |
|-----------------------|
| «interface»           |
| Executável            |
| {abstrato}            |
|                       |
| executar() {abstrato} |

|                       |
|-----------------------|
| «interface»           |
| Armazenável           |
| {abstrato}            |
|                       |
| carregar() {abstrato} |
| salvar() {abstrato}   |

- Classe A implementa a interface *executável* e *armazenável*
- Classe C implementa a interface *executável*
- Classe B usa a interface *executável* e *armazenável* de A e *executável* de C

90

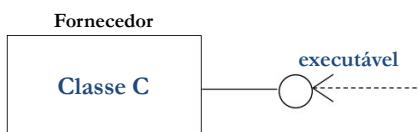
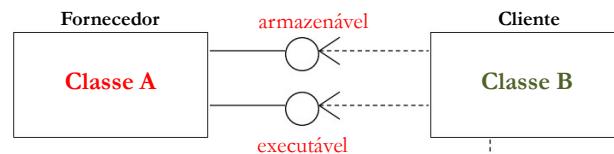
172



- |                                                |                                                 |                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| «interface»<br><b>Executável</b><br>{abstrato} | «interface»<br><b>Armazenável</b><br>{abstrato} | <ul style="list-style-type: none"> <li>• Classe A implementa a interface <b>executável</b> e <b>armazenável</b></li> <li>• Classe C implementa a interface <b>executável</b></li> <li>• Classe B usa a interface <b>executável</b> e <b>armazenável</b> de A e <b>executável</b> de C</li> </ul> |
| executar() {abstrato}                          | carregar() {abstrato}<br>salvar() {abstrato}    |                                                                                                                                                                                                                                                                                                  |

91

173



- |                                                |                                                 |                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| «interface»<br><b>Executável</b><br>{abstrato} | «interface»<br><b>Armazenável</b><br>{abstrato} | <ul style="list-style-type: none"> <li>• Classe A implementa a interface <b>executável</b> e <b>armazenável</b></li> <li>• Classe C implementa a interface <b>executável</b></li> <li>• Classe B usa a interface <b>executável</b> e <b>armazenável</b> de A e <b>executável</b> de C</li> </ul> |
| executar() {abstrato}                          | carregar() {abstrato}<br>salvar() {abstrato}    |                                                                                                                                                                                                                                                                                                  |

92

174

- Processo formal que examina os atributos de uma classe com o intuito de minimizar redundâncias de dados em objetos específicos
- Causa a simplificação dos atributos dentro da respectiva classe colaborando para a integridade e estabilidade do modelo

93

175

| Pedido                         |                   |                             |
|--------------------------------|-------------------|-----------------------------|
| Número do pedido: _____        |                   |                             |
| Código do cliente: _____       |                   |                             |
| Nome do cliente: _____         |                   |                             |
| Data do pedido: ____/____/____ |                   |                             |
| <i>Código do produto</i>       | <i>Quantidade</i> | <i>Descrição do produto</i> |
|                                |                   |                             |
|                                |                   |                             |
|                                |                   |                             |

Classe pedido preliminar

| Pedido                  |
|-------------------------|
| númeroDoPedido          |
| códigoDoCliente         |
| nomeDoCliente           |
| dataDoPedido            |
| códigoDoProduto[15]     |
| quantidadeDoProduto[15] |
| descriçãoDoProduto[15]  |
|                         |

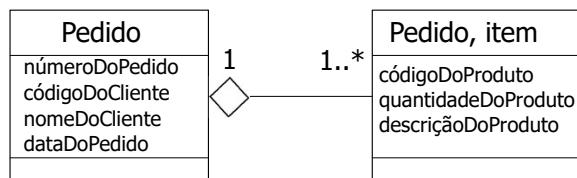
94

176

- DEFINIÇÃO:

- É a normalização da classe de forma que de seus atributos com o identificador seja única, isto é, para cada identificador há a ocorrência de um e somente um atributo de cada tipo

- Resultado da 1a forma normal:



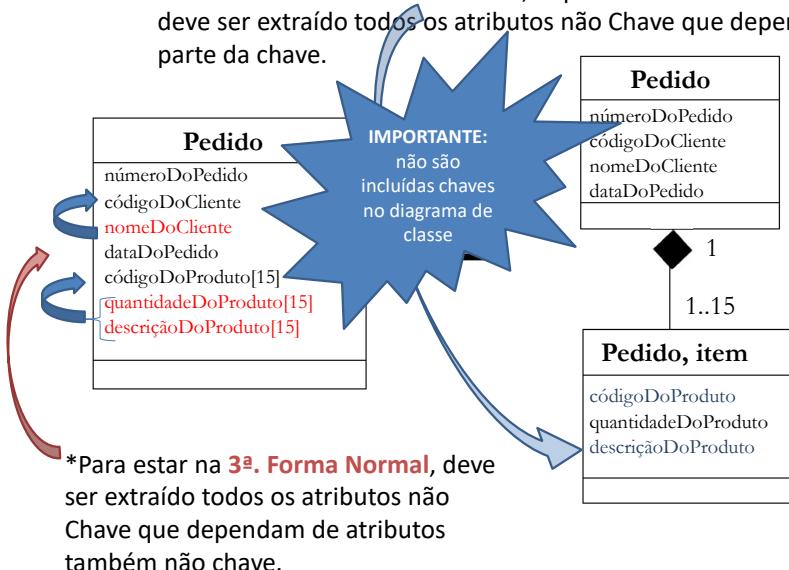
\*Para estar na 1<sup>a</sup>. Forma Normal, deve ser extraído todos os atributos Multivvalorados

95

177

## 1a forma normal como agregação de composição

\*Para estar na 2<sup>a</sup>. Forma Normal, dependência funcional parcial, deve ser extraído todos os atributos não Chave que dependam de parte da chave.



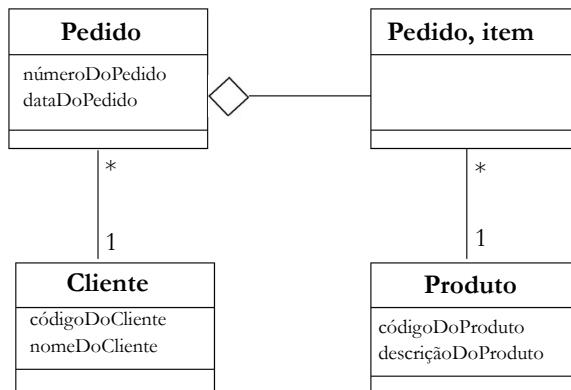
96

178

### 3a forma normal

- **DEFINIÇÃO:**

É a normalização da estrutura de atributos de forma que não contenha atributos associados que apresentem DEPENDÊNCIA TRANSITIVA com o identificador do objeto, ou seja, não possuam elemento intermediário de ligação entre os dois



97

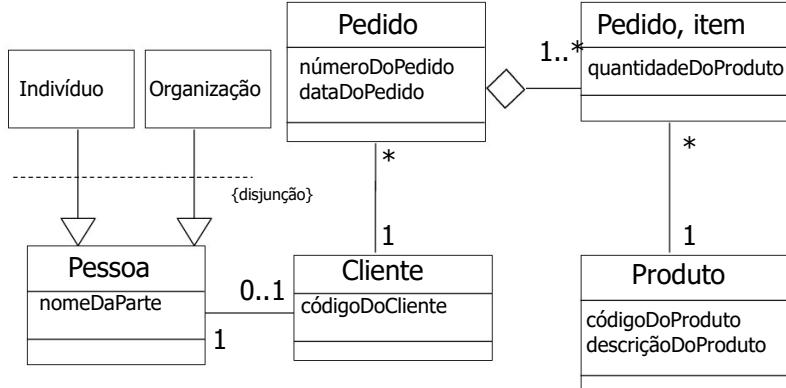
179

### Passos da normalização

- A primeira forma normal é obtida através dos seguintes passos:
  - Verificar se há ocorrências repetitivas de atributos dentro da classe [A] analisada
  - Destacar os atributos repetitivos e suas respectivas operações, criando uma nova classe [B] que absorverá esses itens
  - Estabelecer a associação de agregação regular e multiplicidade 1-\* entre as classes [A] e [B]
    - OBS: Caso o modelo de objetos esteja sendo implementado em um banco de objetos, não há, rigorosamente, a necessidade de se promover a primeira forma normal
- A terceira forma normal deve suceder a segunda forma normal sendo atingida através dos seguintes passos:
  - Verificar se a classe [C] analisada possui atributos que são dependentes de outros atributos nela contidos. Essa relação de dependência pode ser através de atributo próprio ou atributo de referência que não estejam presentes no identificador da classe [C]
  - Destacar os atributos com dependência transitiva e suas respectivas operações, incorporando-os na classe [D]
  - Eliminar os atributos obtidos por cálculo a partir de outros atributos da classe [C]

98

180



99

181

- **DEFINIÇÃO:**  
É a normalização da estrutura de atributos de forma que não contenha **DEPENDÊNCIA MULTIVALORADA** entre os atributos componentes do identificador
- **DEPENDÊNCIA MULTIVALORADA:**  
Dados três atributos A, B e C define-se dependência multivalorada quando A apresenta vários B e A apresenta vários C, entretanto, B e C são independentes entre si

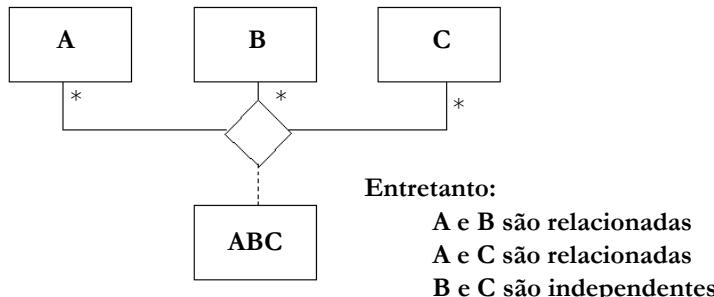
100

182

## 4a forma normal

---

Suponha a seguinte formação ternária de associações  
(deve haver pelo menos 3 classes envolvidas para a 4<sup>a</sup> FN):



101

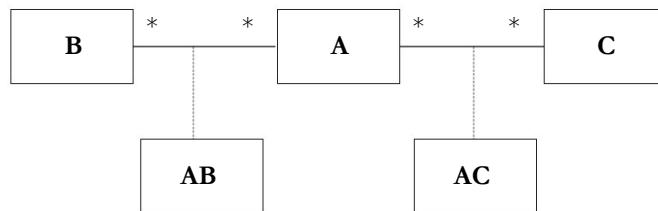
183

## Resultado da 4a forma normal

---

Portanto:

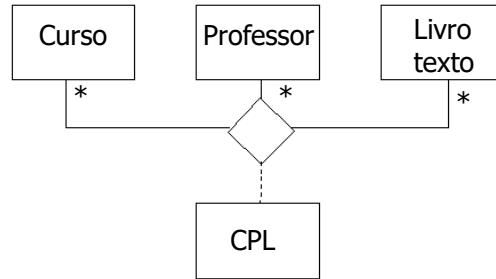
Aplica-se a 4<sup>a</sup> forma normal para transformar uma associação ternária em duas associações binárias, eliminando-se dependências multivaloradas, resultando em:



102

184

## Exemplo



103

185

## 4a forma normal - Exemplo

Antes...

| Curso      | Professor                   | Livro texto                                    |
|------------|-----------------------------|------------------------------------------------|
| Física     | Hertz, Heinrich Rudolf      | <i>Electromagnetic waves</i>                   |
| Física     | Maxwell, James Clerk        | <i>Electromagnetic waves</i>                   |
| Física     | Fizeau, Armand              | <i>The Speed of light</i>                      |
| Física     | Foucault, Jean Bernard Léon | <i>The Speed of light</i>                      |
| Física     | Doppler, Christian Johann   | <i>The color effect of double stars</i>        |
| Matemática | Babbage, Charles            | <i>Economy of Machines and Manufactures</i>    |
| Matemática | Boole, George               | <i>An Investigation of the Laws of Thought</i> |

Depois...

| Curso, Professor |                             |
|------------------|-----------------------------|
| Curso            | Professor                   |
| Física           | Hertz, Heinrich Rudolf      |
| Física           | Maxwell, James Clerk        |
| Física           | Fizeau, Armand              |
| Física           | Foucault, Jean Bernard Léon |
| Física           | Doppler, Christian Johann   |
| Matemática       | Babbage, Charles            |
| Matemática       | Boole, George               |

| Curso, Livro texto |                                                |
|--------------------|------------------------------------------------|
| Curso              | Livro texto                                    |
| Física             | <i>Electromagnetic waves</i>                   |
| Física             | <i>The Speed of light</i>                      |
| Física             | <i>The color effect of double stars</i>        |
| Matemática         | <i>Economy of Machines and Manufactures</i>    |
| Matemática         | <i>An Investigation of the Laws of Thought</i> |

104

186



## Diagrama de Caso de Uso

105

187

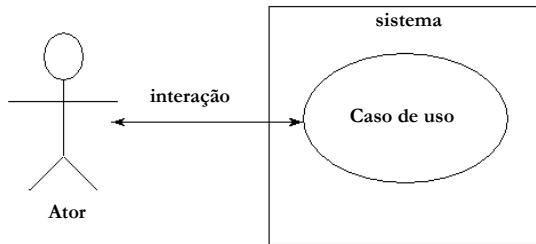
### Diagrama de caso de uso

- Fornecem um modo de descrever a visão externa do sistema e suas interações com o mundo exterior
- Representa uma visão de alto nível da funcionalidade intencional de um sistema mediante o recebimento de um tipo de requisição de usuário
- É um gráfico de atores, um conjunto de casos de uso incluído por um limite de domínio, comunicação, participação e associações entre atores, assim como generalizações entre casos de uso. Há quatro elementos básicos em um diagrama de caso de uso:
  - ator
  - caso de uso
  - interação
  - sistema

106

188

## Diagrama de caso de uso



107

189

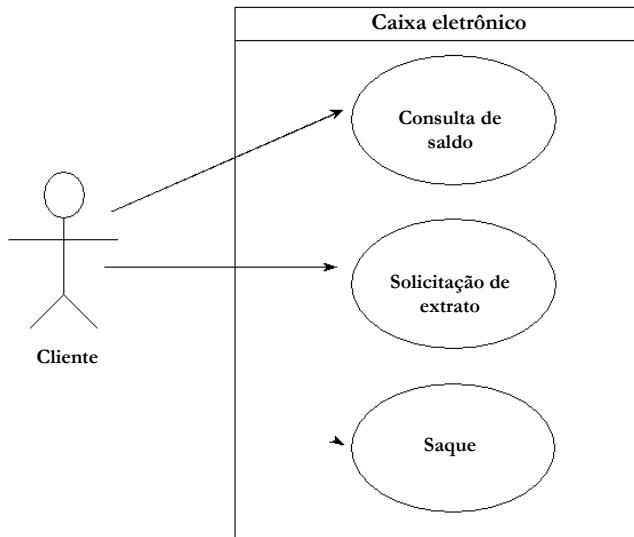
## Caso de uso

- Caso de uso é uma interação típica entre um usuário e um sistema
- É um modo específico de usar um sistema a partir de um ponto de vista segmentado de sua funcionalidade
- Representa uma seqüência completa de cenários de interação mostrando como eventos externos iniciais são respondidos no caso
  - Um cenário é uma narrativa de uma parte do comportamento global do sistema, sendo que uma coleção completa de cenários pode ser usada para especificar completamente um sistema

108

190

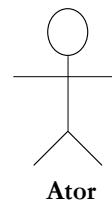
## Diagrama de caso de uso



109

191

- O mundo externo é representado por atores que desempenham papéis
- Um ator é um agente que interage com o sistema, um tipo de usuário ou categoria com papel definido, podendo incluir usuários humanos, máquinas, dispositivos ou outros sistemas
- Atores típicos são cliente, usuário, gerente, impressora, dispositivo de comunicação de rede etc
- A ênfase em papéis é importante: um ator pode representar muitos papéis e um papel (*forma de stick man*) pode ser representado por muitos atores
- Atores levam a cabo casos de uso mas não precisam ser seres humanos (ainda que sejam representados como figuras de stick man dentro de um diagrama de caso de uso)



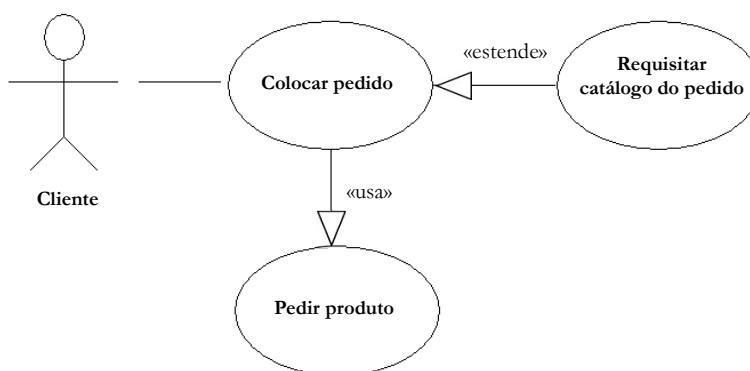
110

192

- As seguintes interações são importantes dentro de um diagrama de caso de uso:
  - **Comunicação**
    - Um ator comunica-se com o caso de uso, assim, cada participação sua é mostrada conectando-se o símbolo de ator ao símbolo de caso de uso por um caminho sólido
  - **Extensão**
    - Um relacionamento de um caso de uso para outro, especificando como o comportamento definido para o primeiro caso pode ser inserido no comportamento definido para o segundo
  - **Uso**
    - Um relacionamento de uso entre casos é mostrado por uma seta de generalização do caso de uso que faz o uso ao caso de uso que é usado

111

193



112

194

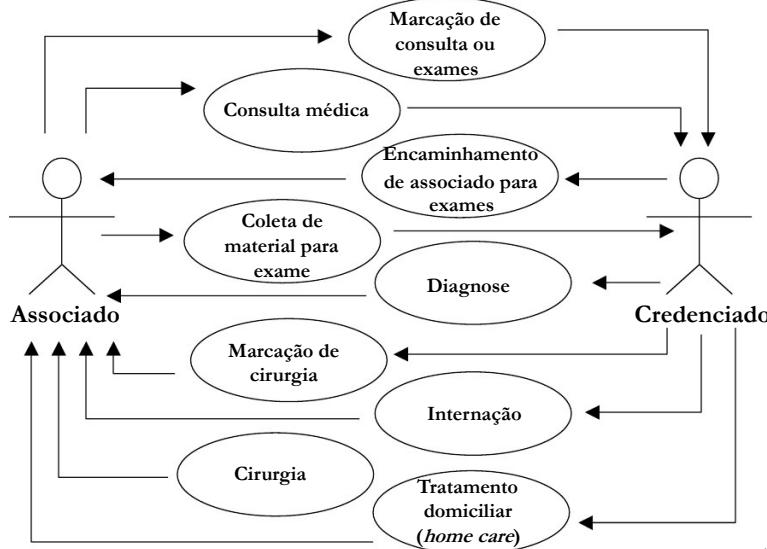
## Desenvolvimento de casos de uso

- Sugere-se os seguintes passos para se descrever casos de uso de sistemas:
  - identifique o limite de sistema em termos de seus agentes externos (atores)
  - identifique cada ator pelo papel que representa na interação com o sistema, por exemplo, cliente, gerente, caixa eletrônico
  - para cada ator, identifique os modos diferentes e fundamentais nos quais utiliza o sistema
  - para cada caso de uso, descreva a interação típica (cenário) do ator com o sistema

113

195

## Exemplo de casos de uso



114

196

| Nr. | Caso de uso                                | Quem inicia a ação | Descrição do caso de uso                                                                                                                                                                      |
|-----|--------------------------------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | Marcação de consulta ou exames             | Associado          | O associado entra em contato com o credenciado para marcar consultas ou exames                                                                                                                |
| 2   | Consulta médica                            | Associado          | O associado encaminha-se ao local da consulta e é atendido pelo credenciado                                                                                                                   |
| 3   | Encaminhamento de associado para exames    | Credenciado        | O credenciado encaminha o associado para a realização de exames laboratoriais                                                                                                                 |
| 4   | Coleta de material para exame              | Associado          | O associado encaminha-se ao credenciado para realizar coleta de material para exame laboratorial                                                                                              |
| 5   | Diagnose                                   | Credenciado        | O credenciado entrega o resultado do exame laboratorial para o associado                                                                                                                      |
| 6   | Marcação de cirurgia                       | Credenciado        | Após a diagnose, sendo necessária uma intervenção cirúrgica, o credenciado agenda cirurgia                                                                                                    |
| 7   | Internação                                 | Credenciado        | O credenciado faz internação do associado mediante a apresentação da ordem de internação ou com o pedido do mérito mediante cheque de caução, devolvido após a entrega da ordem de internação |
| 8   | Cirurgia                                   | Credenciado        | Credenciado efetuá cirurgia de associado, faz acompanhamento pós-operatório de recuperação e emite alta                                                                                       |
| 9   | Tratamento domiciliar ( <i>home care</i> ) | Credenciado        | O credenciado faz o atendimento e tratamento do associado em sua residência, quando não há a necessidade de internação hospitalar                                                             |

115

197

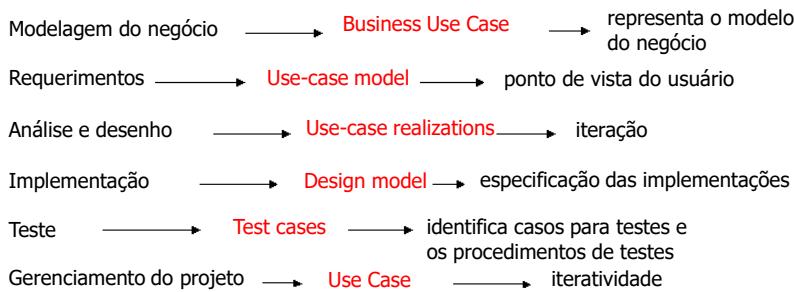
- As exigências de um sistema nem sempre são óbvias e fáceis de se encontrar
- Gerenciamento de requisitos é uma forma sistemática de:
  - Encontrar
  - Documentar
  - Organizar
  - Controlar as exigências de mudança em um sistema
- Proposta do RUP:
  - Analisar o problema
  - Focar nas necessidades do cliente (*stakeholders*)
  - Controlar o escopo do sistema
  - Refinar as exigências do sistema
  - Gerenciar das exigências de mudanças
  - Preparar casos de uso

116

198

## Gerenciamento de requisitos

- Antes de construir um sistema, é importante capturar os requisitos
- O RUP (Rational Unified Process) emprega casos de uso, como base para o desenvolvimento
- O caso de uso é parte integrante dos ciclos no desenvolvimento iterativo



117

199



## Diagramas de interação

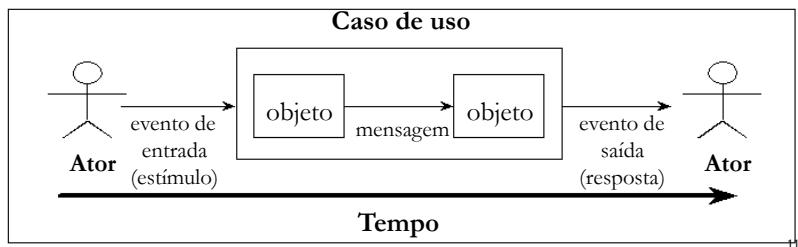
Diagrama de seqüência  
Diagrama de colaboração

118

200

## Diagrama de interação

- Diagrama de interação é um termo genérico que se aplica a vários tipos de diagramas que enfatizam interações de objeto
- Uma interação é uma especificação comportamental que inclui uma seqüência de trocas de mensagem entre um conjunto de objetos dentro de um contexto para realizar um propósito específico, tal como o implementação de uma operação
- Diagrama de interação tem aplicabilidade para cumprir com exigências de seqüência onde cada linha específica em um caso de uso é chamada cenário



201

## Diagrama de interação

- Diagrama de interação deve ser usado quando se deseja visualizar o comportamento de vários objetos dentro de um único caso de uso, a partir das mensagens que são passadas entre eles
- Diagrama de interação é apresentado em duas formas na UML:
  - Diagrama de seqüência
  - Diagrama de colaboração

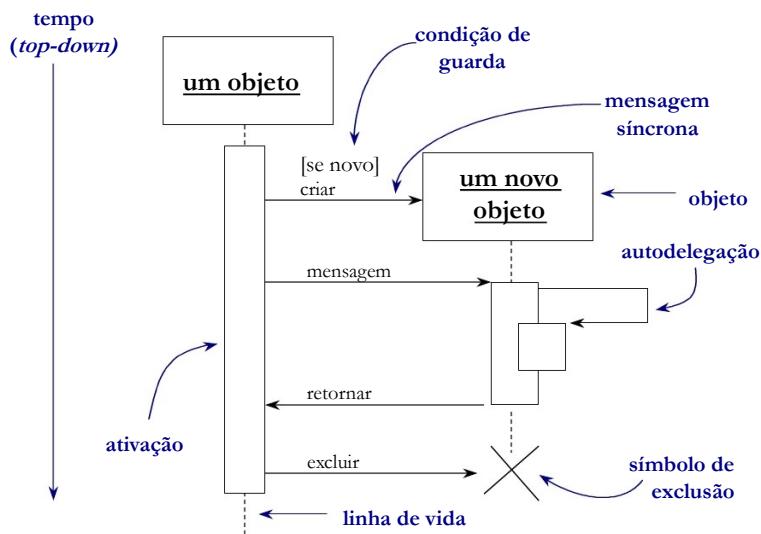
120

202

- Dentro de um diagrama de seqüência, um objeto é mostrado como um retângulo ao topo de uma linha vertical tracejada projetada para baixo
- Essa linha vertical é chamada de linha de vida do objeto e representa, tal como sugerido pelo próprio nome, o ciclo de vida de um objeto durante uma interação
- Cada mensagem é representada por uma linha com seta dirigida horizontalmente entre as linhas de vida de dois objetos
- A ordem na qual essas mensagens acontecem (fluxo de tempo) é mostrada de maneira top-down
- A menos que especificamente anotado, só a seqüência de mensagens é mostrada, não o tempo exato
- A UML permite somar anotações textuais a diagramas de seqüência quando o tempo é importante

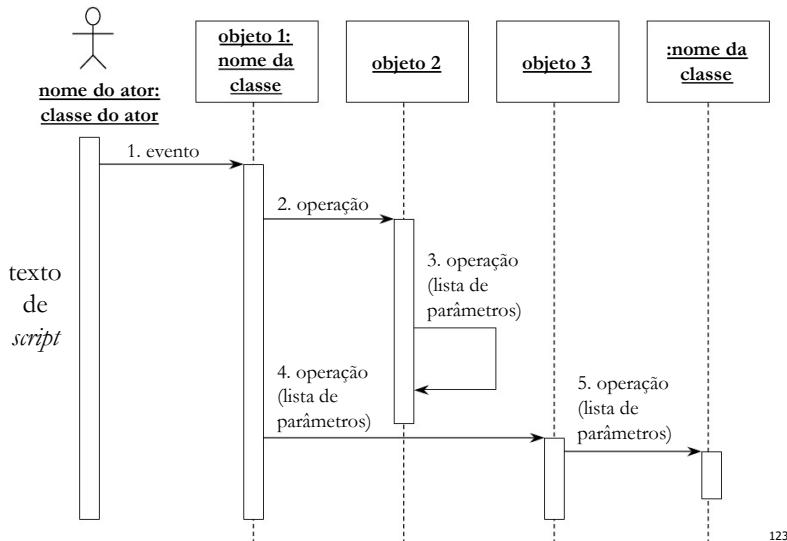
121

203



122

204



205

- O diagrama de seqüência tem duas dimensões: a dimensão vertical representa o tempo e a dimensão horizontal representa objetos diferentes
- O tempo procede de cima para baixo na página, sendo que não há qualquer significado ao ordenamento horizontal dos objetos
- Anotações textuais ao longo da extremidade esquerda do diagrama são opcionais e são chamadas *script*
- Cada declaração nos textos de script explica uma ou mais mensagens que são passadas no diagrama, podendo corresponder diretamente ao cenário real que está sendo modelado pelo diagrama de seqüência

124

206

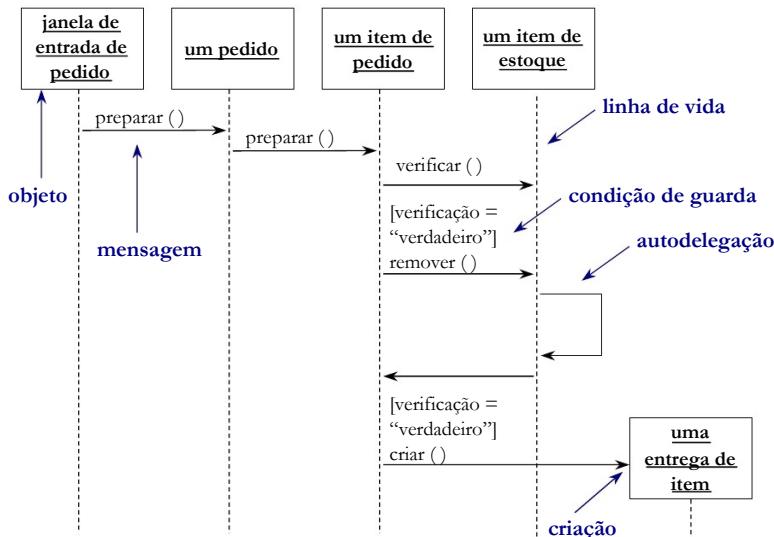
## Linha de vida do objeto

- Um objeto é mostrado como uma linha pontilhada vertical chamada linha de vida do objeto, que representa sua existência em um momento particular
- Se o objeto é criado ou eliminado durante o período de tempo mostrado no diagrama, então sua linha de vida começa ou termina em um ponto designado; caso contrário ocorre de maneira top-down
- Cada linha de vida representa um objeto distinto, podendo haver linhas de vida múltiplas para objetos diferentes dos mesmos tipos ou de tipos diferentes

125

207

## Diagrama de seqüência



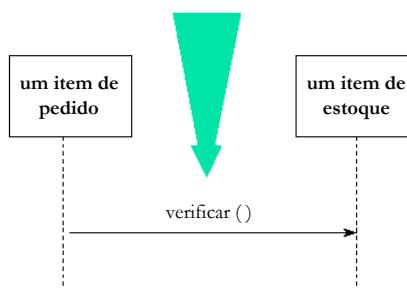
126

208

- Objetos necessitam comunicar-se entre si, e isto ocorre através da passagem de mensagens
- Objetos remetentes enviam mensagens para objetos destinatários, pedindo processamento, comunicando um evento ou qualquer outra comunicação que se tornar necessária no modelo para cumprir determinadas responsabilidades
- Este é o único meio para se obter informação de um objeto, desde que seus dados não sejam diretamente acessíveis: uma comunicação entre objetos carrega uma expectativa de ação a ser executada
- Mensagem é mostrada como uma seta sólida horizontal da linha da vida de um objeto para a linha da vida de outro objeto, etiquetada com o nome da mensagem (operação ou sinal) e seus valores de argumento. No caso de uma mensagem de um objeto para si mesmo, a seta pode começar e terminar no mesmo símbolo de objeto (autodelegação)

127

209



128

210

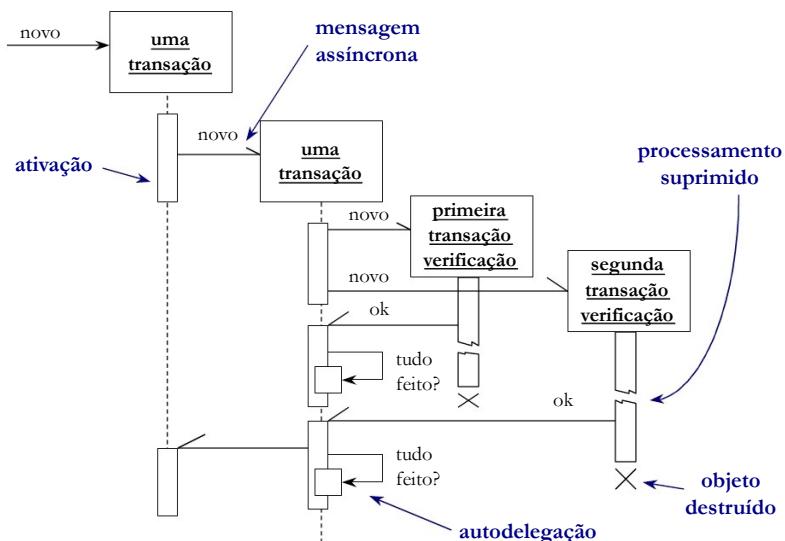
## Tipos de mensagem

- Ponta de flecha sólida preenchida →
  - Chamada de procedimento ou outro fluxo aninhado de controle. A seqüência aninhada inteira é completada antes das retomadas de seqüências exteriores. Pode ser usada com chamadas de procedimento comuns e também com objetos concorrentemente ativos quando um deles envia um sinal e espera pela conclusão de uma seqüência aninhada de comportamento
- Ponta de flecha fina →
  - Fluxo de controle plano. Cada seta mostra a progressão em seqüência ao próximo passo. Normalmente todas as mensagens são assíncronas
- Meia ponta de flecha fina ↘
  - Fluxo assíncrono de controle. Usado em vez da ponta da flecha fina para mostrar explicitamente uma mensagem assíncrona entre dois objetos em uma seqüência procedural
- Outras variações
  - Podem ser mostrados outros tipos de controle, como "empacado" ou "pausa", mas estes são tratados como extensões à UML

129

211

## Mensagem assíncrona



130

212

## Tipos de mensagem

- Mensagem de intervalo

- Uma mensagem de intervalo indica que o remetente esperará pelo destinatário estar pronto para a mensagem até um período fixo de tempo antes de abortar o processo de transmissão de mensagem e continuar com seu processo

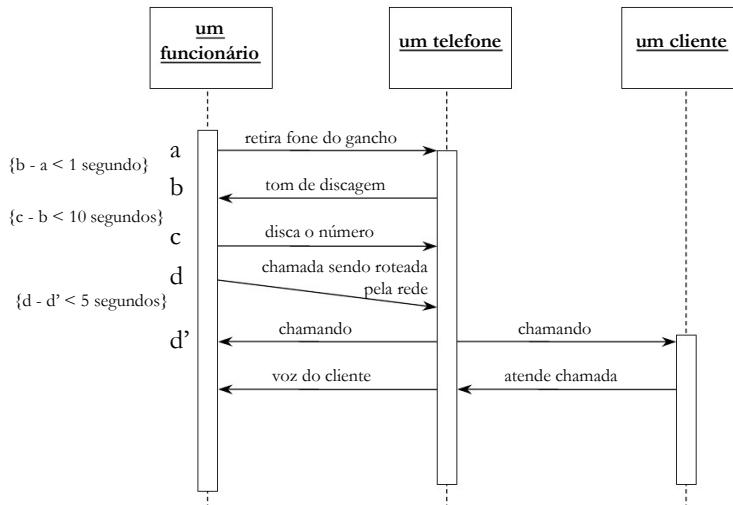
- Mensagem emperrada

- Uma mensagem emperrada significa que se o destinatário da mensagem não estiver imediatamente pronto para aceitar a mensagem, o remetente aborta a mensagem e continua

131

213

## Diagrama de seqüência com objetos concorrentes



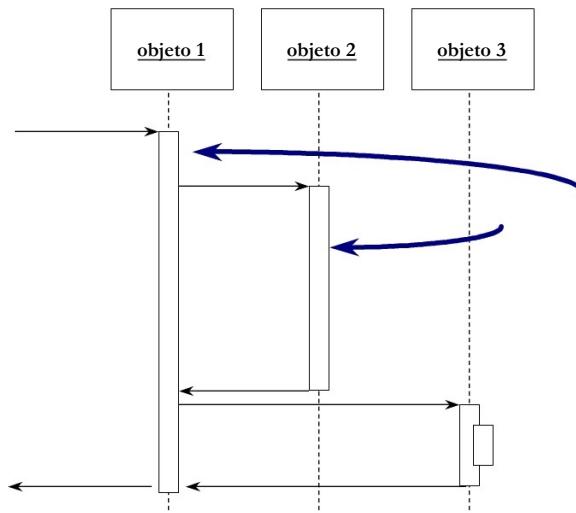
132

214

- É a execução de uma ação
- Uma ativação mostra o período de tempo no qual um objeto está executando diretamente uma ação ou através de um procedimento subordinado
- É exibida como um retângulo cujo topo é alinhado com seu tempo de iniciação e cuja parte inferior é alinhada com seu tempo de conclusão
- Em chamadas reflexivas para um objeto com uma ativação existente, o segundo símbolo de ativação é desenhado ligeiramente para a direita do primeiro, de forma que eles pareçam visualmente empilhados (podem ser aninhadas chamadas reflexivas a uma profundidade arbitrária)

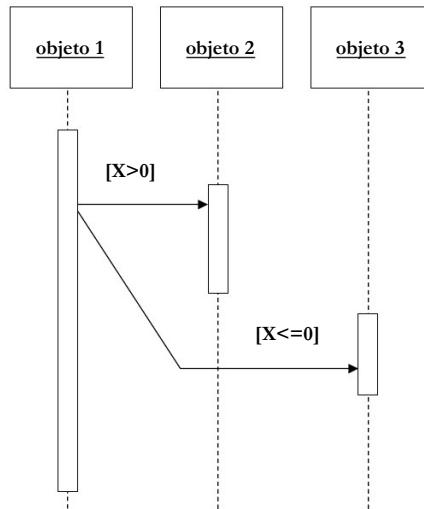
133

215



134

216

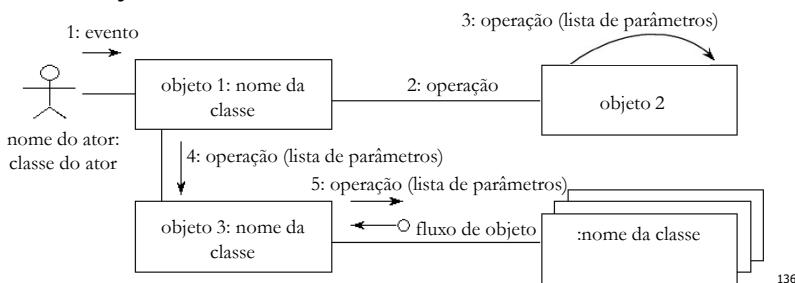


135

217

## Diagrama de colaboração

- Uma colaboração é uma visão de um conjunto de elementos de modelagem relacionados para um propósito particular em apoio a interações
- Um diagrama de colaboração mostra uma interação organizada em torno de objetos e seus vínculos formando uma base de padrões
- Diagrama de seqüência e diagrama de colaboração expressam informação semelhante mas mostram de modo diferente

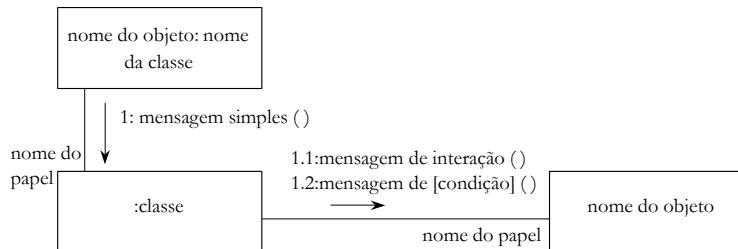


136

218

## Diagrama de colaboração

- Ao contrário de um diagrama de seqüência, um diagrama de colaboração mostra os relacionamentos entre os objetos, mas não mostra o tempo como uma dimensão separada
- A seqüência de tempo, conforme mencionamos, é indicada numerando-se as mensagens
- Dentro de um diagrama de colaboração, os objetos são mostrados como ícones, e tal como no diagrama de seqüência, setas indicam as mensagens enviadas dentro do caso de uso



137

219

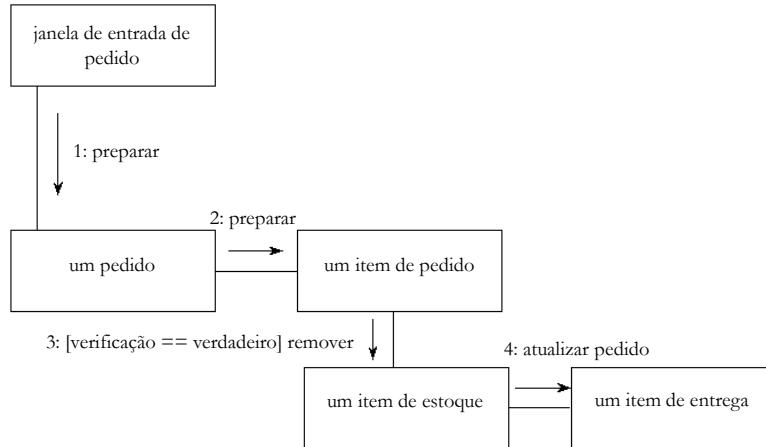
## Colaboração

- Um comportamento é implementado por conjuntos de objetos que trocam mensagens para realizar um propósito em um interação global
- Para entender os mecanismos usados em um desenho, é importante ver somente os objetos e as mensagens envolvidas realizando um ou mais propósitos subjacentes
- Tal construção é chamada de colaboração, uma unidade de modelagem que descreve um conjunto de interações entre tipos

138

220

## Diagrama de colaboração



139

221



## Diagrama de Estado

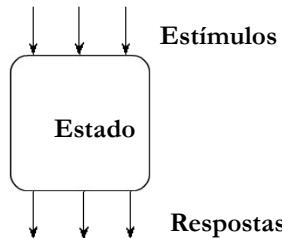
Estado  
Evento

140

222

## Máquina de estado

- Para cada objeto, a ordem das operações no tempo é tão importante que pode-se formalizar a caracterização do comportamento de um objeto em termos de uma máquina de estado finita equivalente
- Um máquina de estado finita pode ser pensada como uma caixa-preta que recebe um número finito de entradas (estímulos) e está preparada para tratá-las oferecendo respostas



141

223

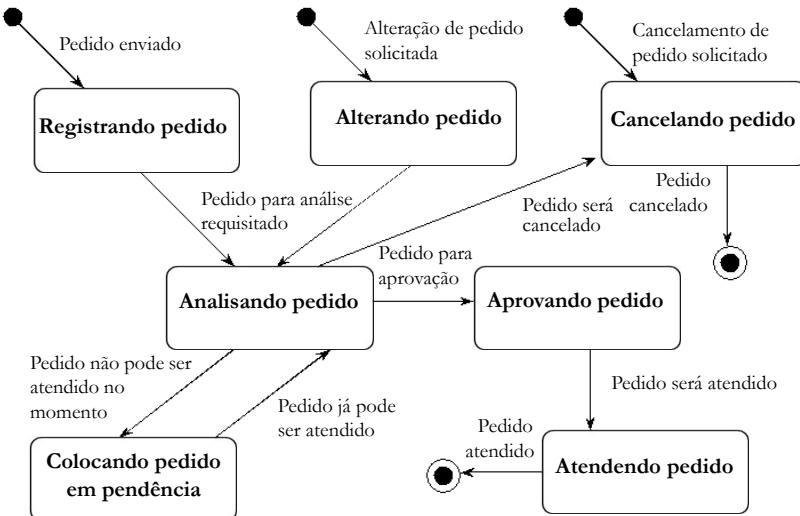
## Diagrama de estado

- Um sistema reage com respostas a estímulos externos ou temporais ocorridos como fatos concretos do mundo físico
- Uma desvantagem do diagrama de estado é ter de definir todos os possíveis estados de um sistema
- A UML propõe o emprego do diagrama de estado de maneira individualizada para cada classe, com o objetivo de tornar o estudo simples o bastante para se ter um diagrama de estado comprehensível

142

224

## Diagrama de estado para a classe Pedido



143

225

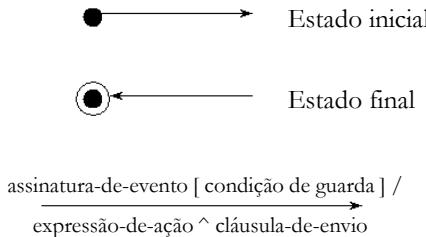
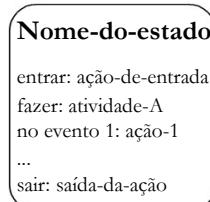
## Elementos do diagrama de estado

- Um diagrama de estado é um gráfico bipartido de estados e transições, conectados por retenção física e particionamento
- Um (pseudo) estado inicial é mostrado como um círculo sólido pequeno
- Um (pseudo) estado final é mostrado como um círculo que cerca um círculo sólido pequeno
- Uma transição é mostrada como uma seta sólida de um estado fonte para outro estado alvo, etiquetada por uma seqüência de transição

144

226

## Notações do diagrama de estado



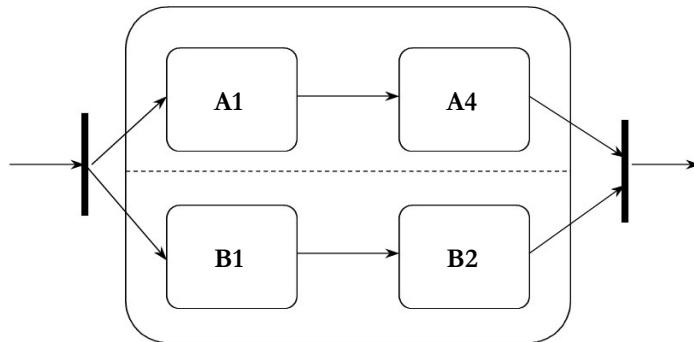
145

227

- Transição simples representa relacionamento entre dois estados indicando que um objeto no primeiro estado executará certas ações especificadas e entrará no segundo estado quando um evento relacionado ocorrer e forem satisfeitas condições preespecificadas
- Transição interna é uma transição que permanece dentro de um único estado em lugar de uma transição que envolve dois estados, representando uma ocorrência de um evento que não causa uma mudança de estado
- Transição complexa é equivalente a uma transição para seu estado inicial ou de cada uma de suas sub-regiões simultâneas. Indica uma transição que se aplica a cada um dos estados dentro da região de estado a qualquer profundidade sendo herdado pelos estados aninhados

146

228

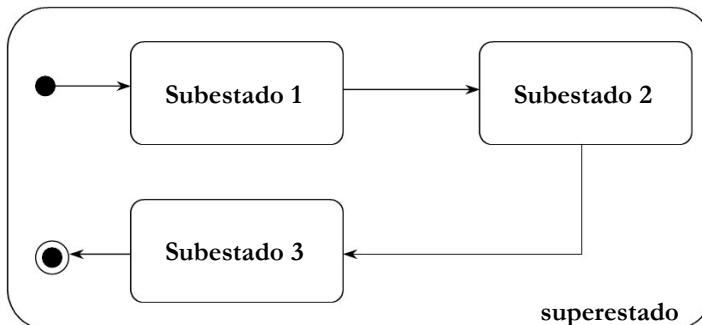


147

229

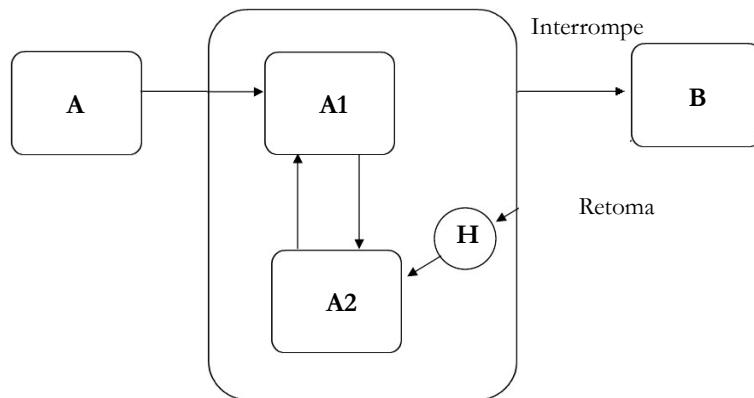
## Aninhamento de estado

- Estados aninhados permitem subdivisões de estados complexos em estruturas mais simples removendo transições duplicadas



148

230



149

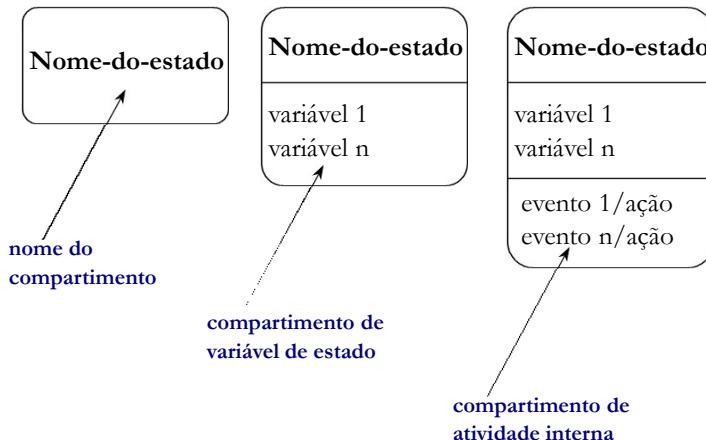
231

- Estado é uma condição ou situação durante a vida de um objeto no qual satisfaz alguma condição, executa alguma atividade em resposta a um evento ou espera pela ocorrência de algum evento
- Essa condição reflete o resultado das seqüências passadas de eventos caracterizado pela reação do sistema
- Como sugestão lingüística, o nome dos estados deve vir sempre no gerúndio, por exemplo: "Cotando material", "Administrando compra", "Preparando entrega", "Montando carga"

150

232

## Notação para estado

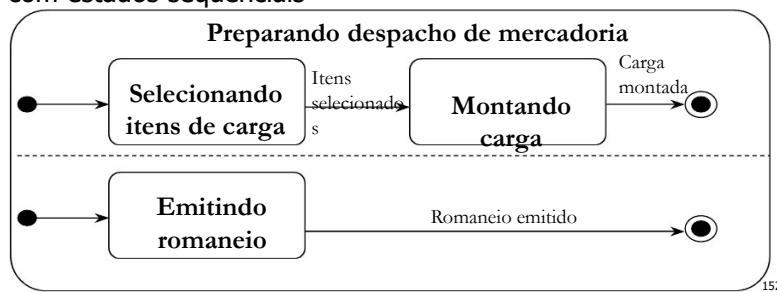


151

233

## Subestado simultâneo

- É um subestado que pode ocorrer simultaneamente com outro subestado contido no mesmo estado composto
- Uma expansão de um estado em subestados simultâneos pode ser mostrada ladrilhando-se a região gráfica por linhas tracejadas
- Cada subregião resultante é um subestado simultâneo, que terá um nome opcional e conterá um diagrama de estado aninhado com estados seqüenciais

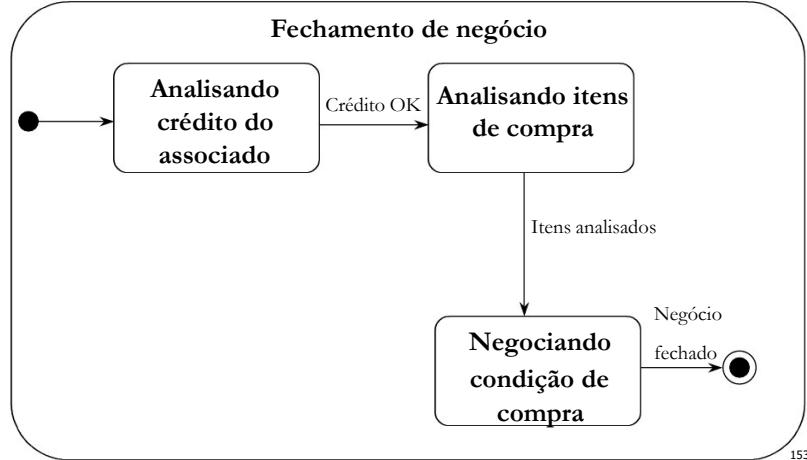


152

234

## Subestado seqüencial

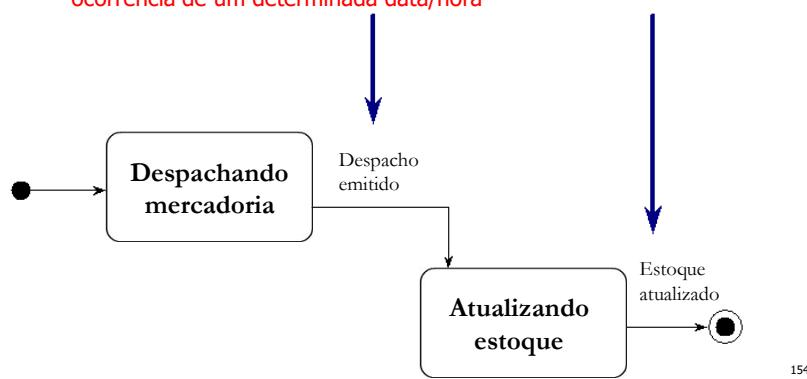
- É um subestado que não pode ocorrer simultaneamente com outros subestados simultâneos contidos no mesmo estado composto



235

## Evento

- É uma ocorrência significativa com localização no tempo e espaço que deve ser reconhecida e reagida pelo sistema em estudo. Existem os seguintes tipos de eventos:
  - Recibo de um sinal explícito de um objeto para outro
  - Passagem de um período designado de tempo depois de um evento ou a ocorrência de um determinada data/hora



236



## Diagrama de atividade

155

237

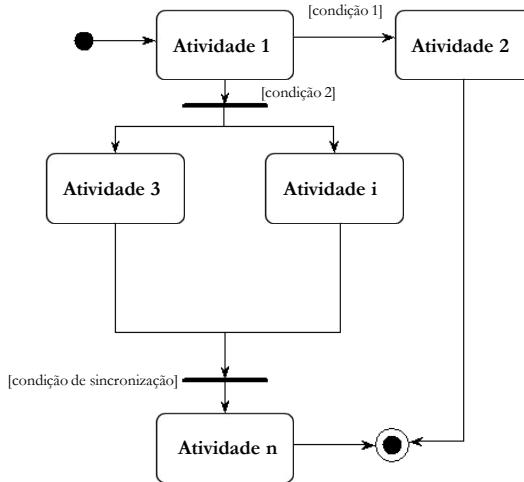
### Diagrama de ação

- Trata-se de um caso especial de diagrama de estado no qual tudo ou a maioria dos estados são estados de ação e a maioria das transições é ativada por conclusão de ações nos estados precedentes
- É útil quando se pretende descrever um comportamento paralelo ou mostrar como interagem comportamentos em vários casos de uso
- É semelhante aos fluxogramas
- Seu uso natural é modelar um passo na execução de um algoritmo
- Não devem apresentar transições internas ou de partida baseadas em eventos explícitos

156

238

## Diagrama de atividade



157

239

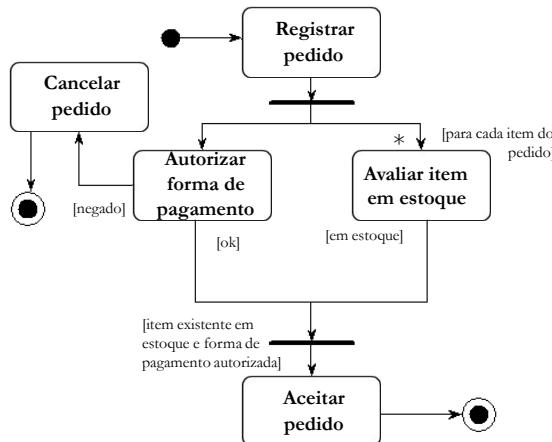
## Diagrama de atividade

- Em uma máquina de estado geral pode-se pensar em dois tipos de estados:
  - **Estado de atividade**
    - Um estado de atividade é apenas um caso especial de um estado normal e representa a execução de uma atividade com um evento implícito na sua terminação para ativar uma transição de saída
  - **Estado de espera**
    - Um estado de espera é um estado normal, tal como encontrado em uma máquina de estado de Harel. Representa um objeto que está esperando por algum evento externo acontecer, além do controle do objeto que possui o estado. Uma atividade representa uma notação conveniente para a primeira situação

158

240

## Diagrama de atividade



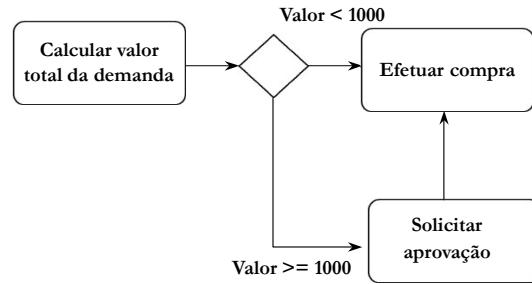
159

241

- Uma decisão é expressa quando são usadas condições de guarda para indicar possíveis transições diferentes que dependem de condições booleanas do objeto possuído
- Uma decisão pode ser mostrada etiquetando transições de saída múltiplas de uma ação com condições de guarda diferentes
- Para conveniência, um estereótipo é provido para uma decisão: a forma de diamante tradicional, com uma ou mais setas entrantes e com duas ou mais setas de partida, cada uma etiquetada com uma condição de guarda distinta sem gatilho de evento. Todos os possíveis resultados deveriam aparecer em uma das transições de partida

160

242



161

243

| Faixa 1     | Faixa i     | Faixa n     |
|-------------|-------------|-------------|
| Atividade 3 | Atividade 1 | Atividade 2 |
|             | Atividade i |             |
|             | Atividade n |             |

162

244



## Diagramas de implementação

Diagrama de componente

Diagrama de implantação

163

245

## Diagramas de interação

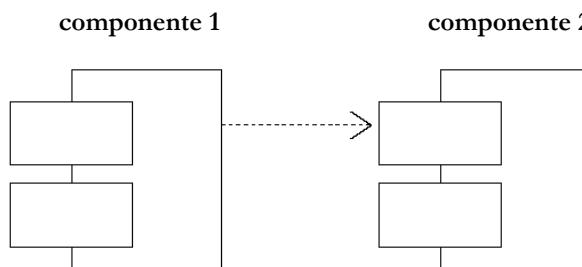
- Mostram aspectos de implementação, inclusive da estrutura de código de fonte e de implementação de run-time.
- Eles se apresentam de duas formas na UML:
  - **Diagrama de componente**
    - mostra a estrutura do próprio código fonte
  - **Diagrama de implantação**
    - mostra a estrutura do sistema de run-time

164

246

## Diagrama de componente

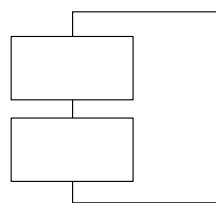
- Um diagrama de componente mostra as organizações e dependências entre componentes com o propósito de modelar a visão de implementação dos módulos de software executáveis com identidade e interface bem-definida de um sistema e seus relacionamentos
- Mostra as dependências entre componentes de software, inclusive componentes de código fonte, componentes de código binário e componentes executáveis



165

247

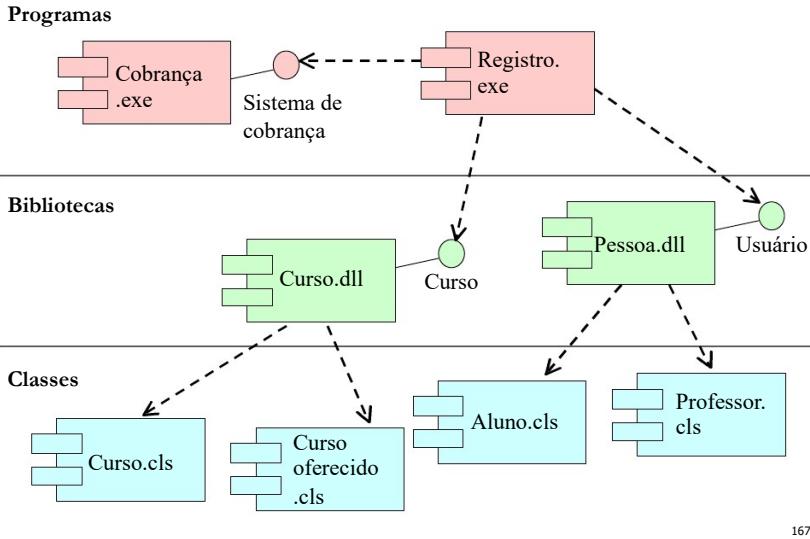
- Um tipo de componente representa um pedaço de código de software (fonte, binário ou executável) e pode ser usado para mostrar o compilador e dependências de run-time
- Uma instância de componente representa uma unidade de código de run-time e pode ser usada para mostrar unidades de código que têm identidade em momento de execução incluindo sua localização nos nós
- Um componente é mostrado como um retângulo maior com dois retângulos pequenos que protraem de seu lado



166

248

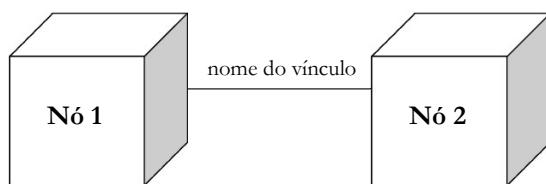
## Exemplo de diagrama de componente



249

## Diagrama de implantação

- A UML fornece diagrama de implantação para mostrar a organização do hardware e a ligação do software aos dispositivos físicos
- Diagrama de implantação mostra vários dispositivos de hardware e suas interfaces físicas
- O tipo do dispositivo de hardware é determinado por seu estereótipo, como processador, dispositivo, exibição, memória, disco e assim por diante



168

250

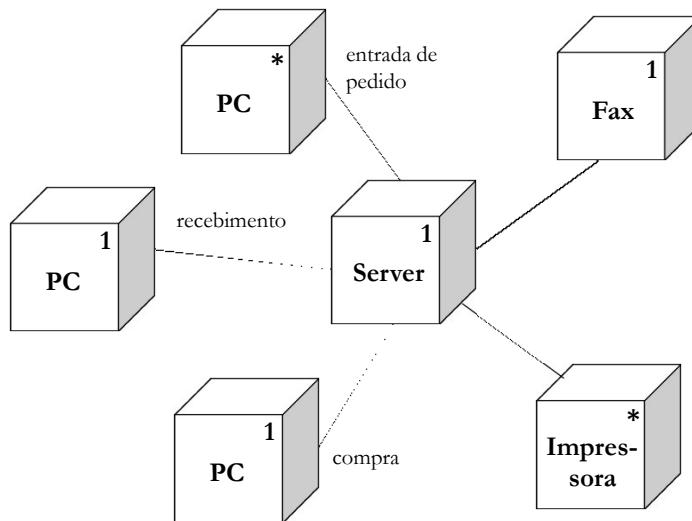
## Diagrama de implantação

- Um diagrama de implantação é um gráfico de nós conectado por associações de comunicação
- Na UML, um nó é um objeto físico de run-time que representa um recurso computacional com pelo menos memória ou capacidade de processamento
- Representa um recurso no real mundo que é passível de distribuição e pode executar elementos dos modelos lógicos
- Objetos e componentes de instâncias computacionais de run-time podem residir em instâncias de nó
- Um nó é mostrado como uma figura que apresenta visão tridimensional como um cubo

169

251

## Diagrama de implantação

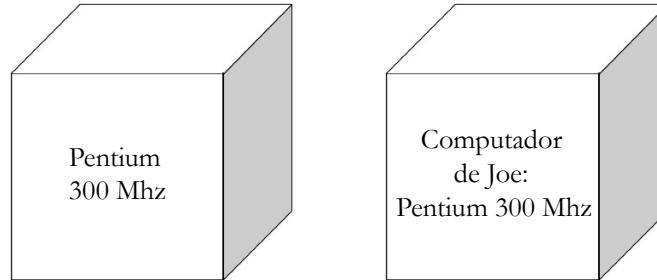


170

252

## Nome do nó

---



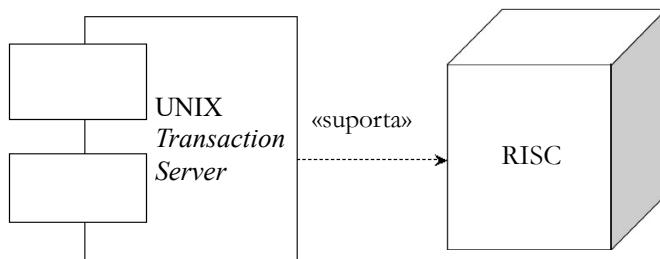
**Pentium 300 Mhz é um tipo de nó enquanto computador de Joe é uma instância de tipo**

171

253

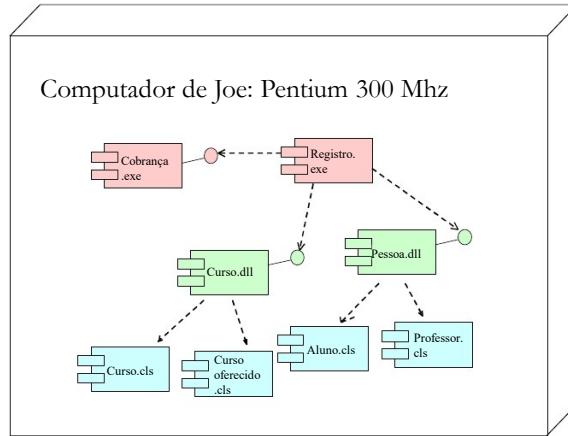
## Tipo de nó suportando tipo de componente de run-time

---



172

254



173

255

## Referências

- PRESSMAN, Roger. **Engenharia de software.** São Paulo: McGraw-Hill., 2006
- BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro: Campus, 2007.
- TONSING, Sérgio Luiz. Engenharia de Software – Análise e Projeto de Sistemas. 2<sup>a</sup> Edição. Rio de Janeiro: Ed. Ciência Moderna, 2008.
- LARMAN, Graig. Utilizando UML e Padrões. 3<sup>a</sup>. Ed. Bookman. 2008.
- SOMMERRVILLE, Ian. Engenharia de Software. 6<sup>a</sup> ed. Ed. Addison Wesley. 2003.

256