

Engenharia de Software

Análise e Projeto Orientados à Objetos

Profª Raquel Mini

raquelmini@ufmg.br

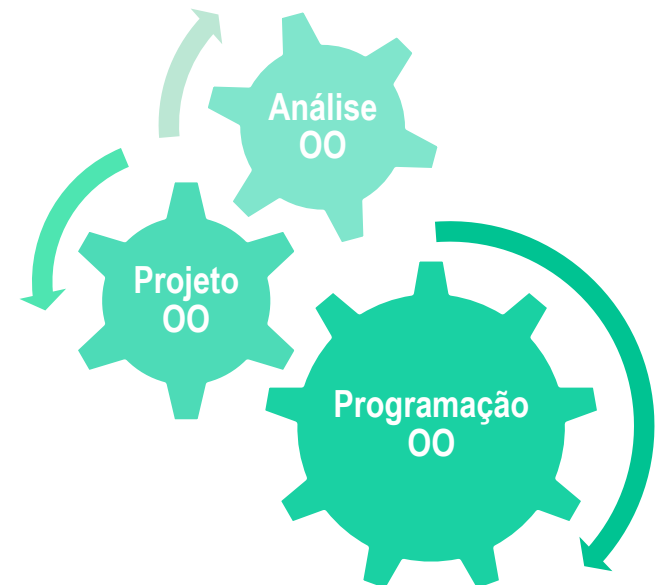
DEE / UFMG

Unidade IV

- ❑ Análise e Projeto Orientados à Objetos
 - Análise orientada a objetos
 - Modelagem de software orientada à objetos
 - Projeto orientado à objetos

Análise e projeto orientados à objetos

- ❑ **Análise:** detalha “o que deve ser feito”
 - Detalha requisitos do sistema
- ❑ **Projeto:** detalha “como será feito”
 - Cria modelos de como o sistema será construído
- ❑ **Programação:** “faz”
 - Constrói o sistema



Análise e projeto orientados à objetos

- ❑ A análise e o projeto orientados à objetos
 - É uma metodologia que nos leva a uma decomposição orientada a objetos de um sistema
 - Os modelos resultantes identificam os **objetos** que compõem o sistema
 - É totalmente distinta da análise estruturada, que busca identificar **procedimentos** (funções)

Paradigma: conjunto de teorias, padrões e métodos que, juntos, representam um modo de organizar o conhecimento



Análise orientada à objetos

- ❑ Método de análise que examina os requisitos a partir da perspectiva das classes e objetos encontrados no vocabulário do domínio do problema
- ❑ Os produtos da **análise** são modelos que servem como entrada para o **projeto**, que por sua vez produz os modelos que são utilizados para a **programação**

Projeto orientado à objetos

- ❑ Método de projeto de software que abrange as tarefas de decompor o sistema de maneira orientada à objetos
 - Gera classes e objetos agrupados
- ❑ Uma notação própria deve ser utilizada para expressar as idéias associadas
 - No nosso caso, utilizaremos a UML
- ❑ O fundamental no projeto é responder as questões de decomposição lógica do sistema
 - Quais são as classes e objetos?
 - Quais são as interfaces destes objetos?
 - Como os objetos interagem entre si?

Análise e projeto orientados à objetos

❑ Programação orientada à objetos

- Método de implementação de software no qual:
 - Os programas são organizados na forma de coleções cooperativas de **objetos**
 - Cada um dos objetos representa uma instância de uma **classe**
 - As classes podem ser membros de **hierarquias** criadas por meio de mecanismos como **herança** e **composição**

❑ Linguagem orientada à objetos

- Linguagem que suporta os mecanismos utilizados na programação orientada a objetos
 - Composição, herança, polimorfismo, encapsulamento, ...

Modelagem de Software Orientada à Objetos

Modelagem de sistemas

- ❑ Como você construiria uma casinha de cachorro?



Modelagem de sistemas

- ❑ Desde que a nova casa seja razoavelmente grande e sem muitas goteiras, seu cachorrinho ficará feliz



Modelagem de sistemas

- ❑ E para construir uma casa para a sua família?



- ❑ Você precisará de um tempo maior e com certeza sua família será mais exigente

Modelagem de sistemas

- ❑ Desde que você se mantenha fiel aos planos e permaneça dentro dos limites e custos, provavelmente sua família ficará satisfeita
- ❑ Caso contrário...



Modelagem de sistemas

- ❑ Para construir um prédio comercial, não será uma boa ideia começar com uma pilha de tábuas e alguns pregos



- ❑ Como você estará tratando com dinheiro de outras pessoas, eles exigirão saber o tamanho, a forma e o estilo do novo prédio

Modelagem de sistemas

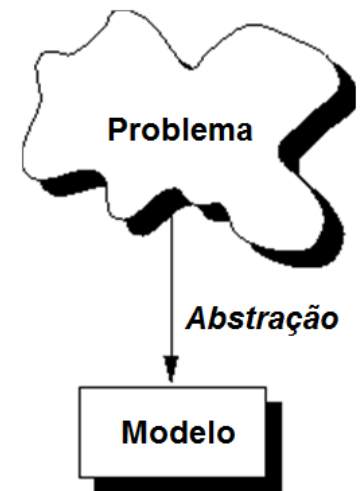
- ❑ A partir do plano será possível fazer estimativa razoável:
 - Tempo
 - Material
 - Pessoas envolvidas
 - Custo
- ❑ É importante definir as expectativas desde o início e gerenciar qualquer modificação com muita cautela

Modelagem de sistemas

- ❑ Muitas empresas começam querendo construir sistemas como se fossem uma casinha de cachorro
- ❑ Se você realmente quiser construir softwares de qualidade, o problema não se restringirá a escrever grande quantidade de código
 - O segredo estará em criar o código correto e pensar em como elaborar menos código
- ❑ O desenvolvimento do software se torna uma questão de métodos, processo e ferramentas

Modelagem de sistemas

- ❑ O que é um modelo?
 - É uma simplificação da realidade
- ❑ Um bom modelo inclui os componentes que possuem ampla repercussão e omite componentes que não são relevantes

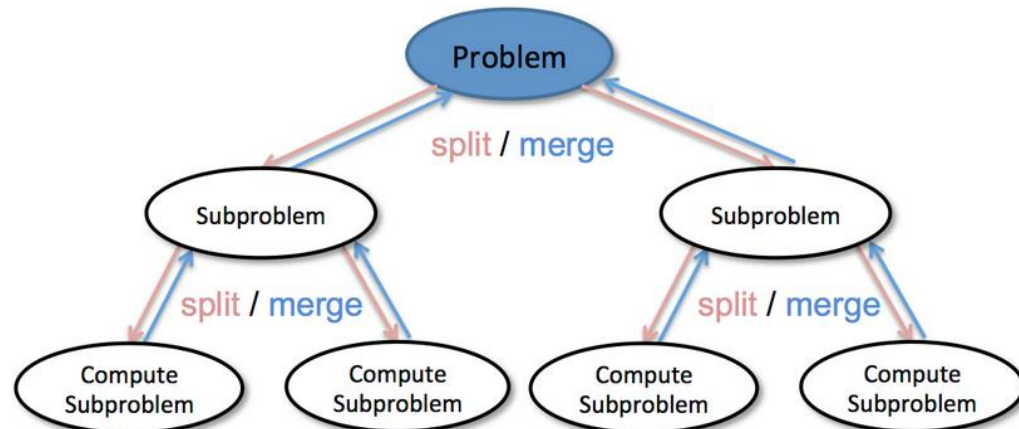


Modelagem de sistemas

- ❑ Por que fazer a modelagem?
 - Para compreender melhor o sistema que estamos desenvolvendo
 - Objetivos:
 - Visualizar o sistema como desejamos que seja
 - Especificar a estrutura e o comportamento do sistema
 - Orientar a construção do sistema
 - Documentar as decisões tomadas

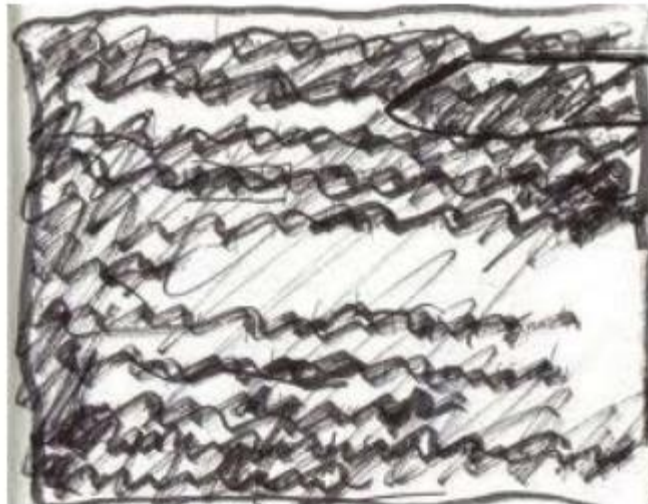
Modelagem de sistemas

- ❑ A capacidade humana é limitada para compreender complexidade
 - Com a modelagem, delimitamos o problema que estamos estudando, restringindo nosso foco a um único aspecto por vez
- ❑ Dividir para conquistar
 - Ataque um problema difícil, dividindo-o em vários problemas menores que você pode solucionar



Modelagem de sistemas

- ❑ O desenvolvedor pode rabiscar uma ideia em um papel para visualizar parte do sistema
 - Esses modelos costumam ser *ad hoc* e não oferecem uma linguagem básica que possa ser compartilhada com outras pessoas facilmente



Modelagem de sistemas

- ❑ Assim como existem linguagem básicas para esboços de projetos na indústria e construção, na engenharia elétrica e na modelagem matemática
 - A indústria de software também pode se beneficiar com o uso de linguagens básicas

Princípios da modelagem

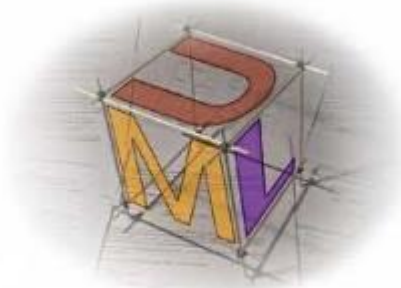
- ❑ Os modelos têm profunda influência sobre a maneira como um determinado problema é atacado e como uma solução é definida
 - Modelos incorretos conduzirão a confusões, desviando a atenção para questões irrelevantes
- ❑ Cada modelo poderá ser expresso em diferentes níveis de precisão
 - Analista/usuário
 - Foco nas questões referentes ao que será visualizado
 - Analista/desenvolvedor
 - Foco na maneira como o sistema funcionará internamente

Princípios da modelagem

- ❑ Nenhum modelo único é suficiente
 - Qualquer sistema será investigado por meio de um conjunto de modelos quase independentes com vários pontos de vista
 - Modelos que possam ser criados e estudados independentemente, mas que continuem inter-relacionados

Visão geral da UML

- ❑ UML (Linguagem de Modelagem Unificada)
 - Linguagem visual utilizada para modelar softwares baseados no paradigma de análise orientada a objetos
 - Não é uma linguagem de programação
 - É uma linguagem de **modelagem** que através de notações permite definir características do sistema
 - Comportamento, estrutura lógica, dinâmica dos processo e até necessidade de equipamentos



Visão geral da UML

- ❑ A UML é uma linguagem destinada a:
 - Visualizar
 - Especificar
 - Construir
 - Documentar
- artefato de um sistema

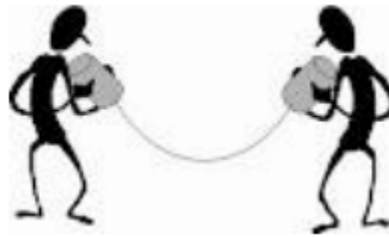
Visão geral da UML

- ❑ É uma linguagem, pois fornece um vocabulário e as regras de combinação das palavras desse vocabulário com a finalidade de comunicar algo
- ❑ É uma linguagem de modelagem pois seu vocabulário e regras possuem seu foco voltado para a representação conceitual e física de um sistema

Visão geral da UML

❑ Vantagens

- Padronização de comunicação entre os membros da equipe
 - Evita ambiguidades e diferenças de interpretação



- A modelagem visual permite que os detalhes do processo sejam expostos ou escondidos conforme a necessidade
 - Auxilia o desenvolvimento de projetos complexos e extensos

Visão geral da UML

❑ Vantagens

- Ajuda a manter a consistência entre a especificação e a implementação através do desenvolvimento iterativo e do planejamento de testes em cada iteração
- Permite avaliar a aderência e a qualidade da arquitetura através de iterações precoces com o usuário
 - Permite que os defeitos possam ser corrigidos antes de comprometer o sucesso do projeto

Visão geral da UML

- ❑ Bons modelos são essenciais para a comunicação entre os *stakeholders*
- ❑ Facilita a programação
 - Ferramentas para modelagem e geração de código

Modelagem orientada a objetos

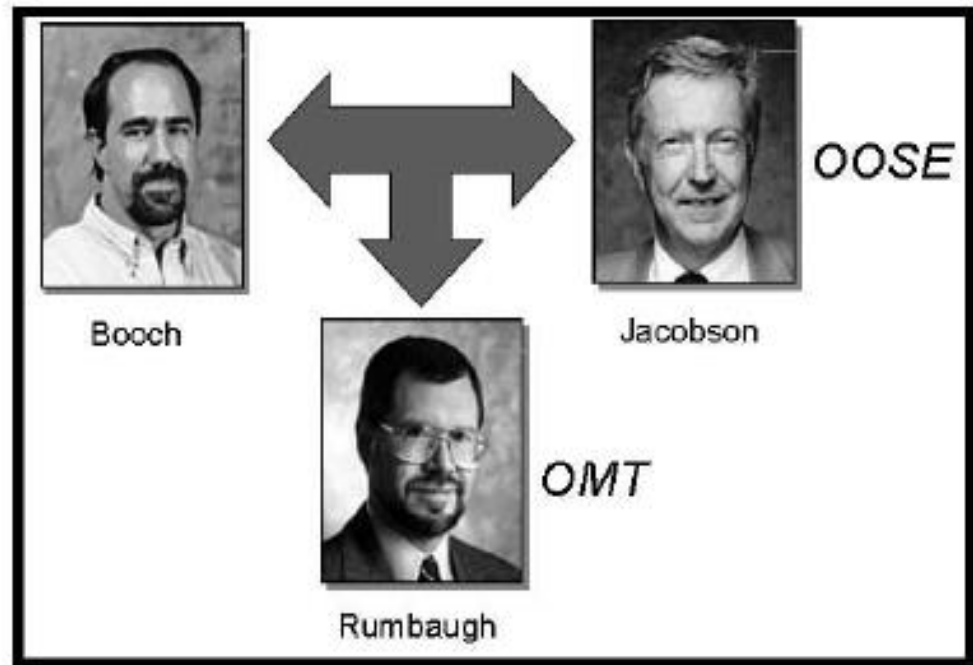
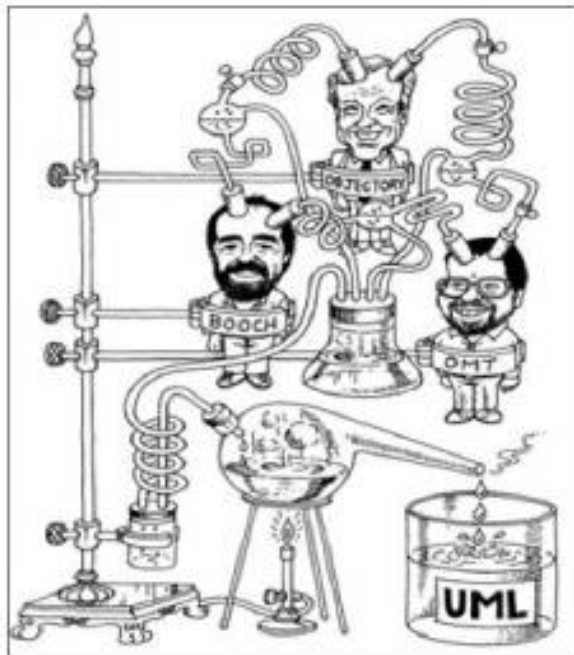
- ❑ Maneira natural de visualizar o software
- ❑ Modela o software semelhante ao mundo real usando objetos
 - Pessoas, animais, plantas, carros, etc.
- ❑ Humanos pensam em termos de objetos
 - Mais alto nível

De onde surgiu a UML?

- ❑ Da união de três metodologias de modelagem
 - Método de Booch - Grady Booch
 - Método OOSE - Ivar Jacobson
 - Método OMT - James Rumbaugh

- ❑ Os três amigos começaram a unificá-las em meados da década de 90

Fundadores da UML



História da UML

- ❑ ←1990's: várias propostas de técnicas de modelagem de sistemas
- ❑ 1990's→: proliferação de propostas de modelagem de sistemas OO
 - Percebeu-se a necessidade de um padrão para a modelagem de sistemas, que fosse aceito e utilizado amplamente
- ❑ 1994: Booch, Jacobson e Rumbaugh começaram a unificar suas notações
- ❑ 1996: Primeira versão (beta) da UML foi liberada

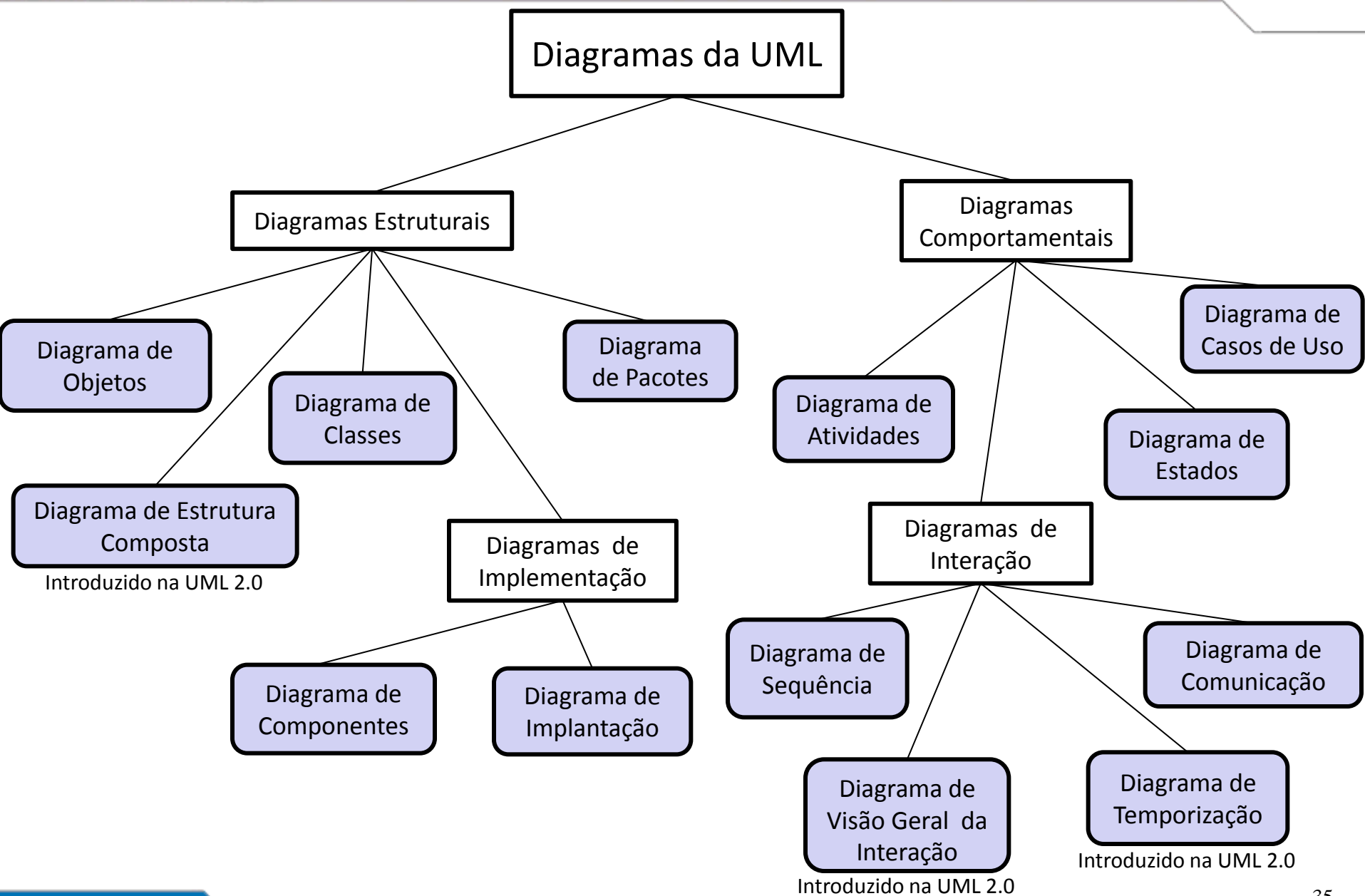
História da UML

- ❑ 1996/97: Grandes empresas formaram a “UML Partners”
 - HP, IBM, Microsoft, Oracle, etc.
- ❑ 1997: UML foi adotada pela a OMG (*Object Management Group*) como linguagem padrão de modelagem
- ❑ Desde então sofreu várias evoluções
 - Atualmente: padrão mundial pela OMG - está na versão 2.4

UML define 13 diagramas

- ❑ Tipos principais de diagramas
 - Estrutural
 - Comportamental

- ❑ Objetivos
 - Visualizar o sistema
 - Especificar estrutura e/ou comportamento
 - Guiar e documentar as decisões



Projeto Orientado à Objetos

Projeto orientado à objetos

- ❑ Na fase de projeto se especifica **como** o sistema será implementado
 - Definição das classes e de como elas irão interagir entre si
 - Analisar cada caso de uso e pensar quais classes deverão existir para realizá-lo
 - Gerar diagramas que possibilitem um entendimento mais claro da solução proposta
 - O ideal é pensar em mais de uma solução técnica possível, avaliá-las e selecionar a melhor

Pensar orientado à objetos

- ❑ Onde quer que você olhe no mundo real, você vê objetos
 - Pessoas, animais, plantas, carros, etc.

- ❑ Humanos pensam em termos de objetos
 - Orientação a objetos é alto nível
 - Ou seja, mais próximo dos humanos que dos computadores

Características de objetos

❑ Classificação

- Animados: possuem vida, se movem...
- Inanimados: não possuem vida

❑ Objetos possuem **atributos**

- Tamanho, forma, cor, peso, etc.

❑ Objetos exibem **comportamentos**

- Uma bola rola, um avião voa
- Uma pessoa anda, fala, pensa, etc.

Classes e objetos

- ❑ Objeto é uma entidade que possui um estado e operações definidas sobre este estado
- ❑ Classe é um “esqueleto” para criação (instanciação) de objetos
 - Como a planta baixa é um “esqueleto” para criação de casas

Definições

❑ Objeto

- Entidade que descreve uma realidade

❑ Classe

- Abstração que define o esqueleto dos objetos

❑ Instância

- Objeto criado a partir de uma classe

Projeto orientado à objetos

- ❑ Maneira natural de visualizar o software
 - Documentação de alto nível
 - Comunicação entre membros da equipe
- ❑ Modela o software semelhante ao mundo real, usando objetos
- ❑ **Objetos** são modelados em termos de seus **atributos** e comportamento (**métodos**)

Vantagens da orientação à objetos

- ❑ Facilidade de entendimento
 - Mapeamento de entidades do mundo real para objetos de sistema
- ❑ Facilidade de manutenção
 - Mais fácil de alterar pois os objetos são independentes
- ❑ Facilidade de reuso
 - Objetos são potencialmente componentes reusáveis

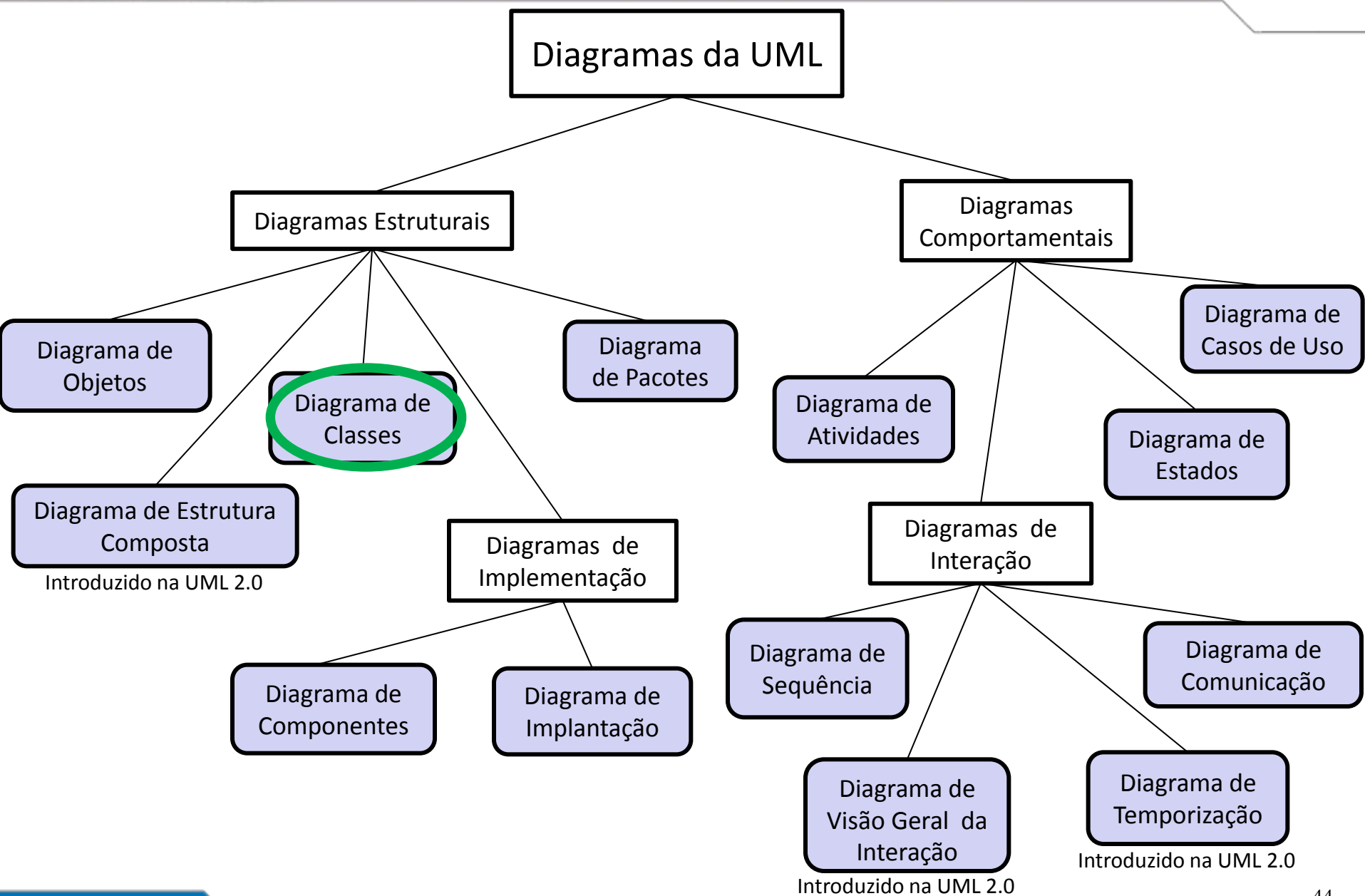
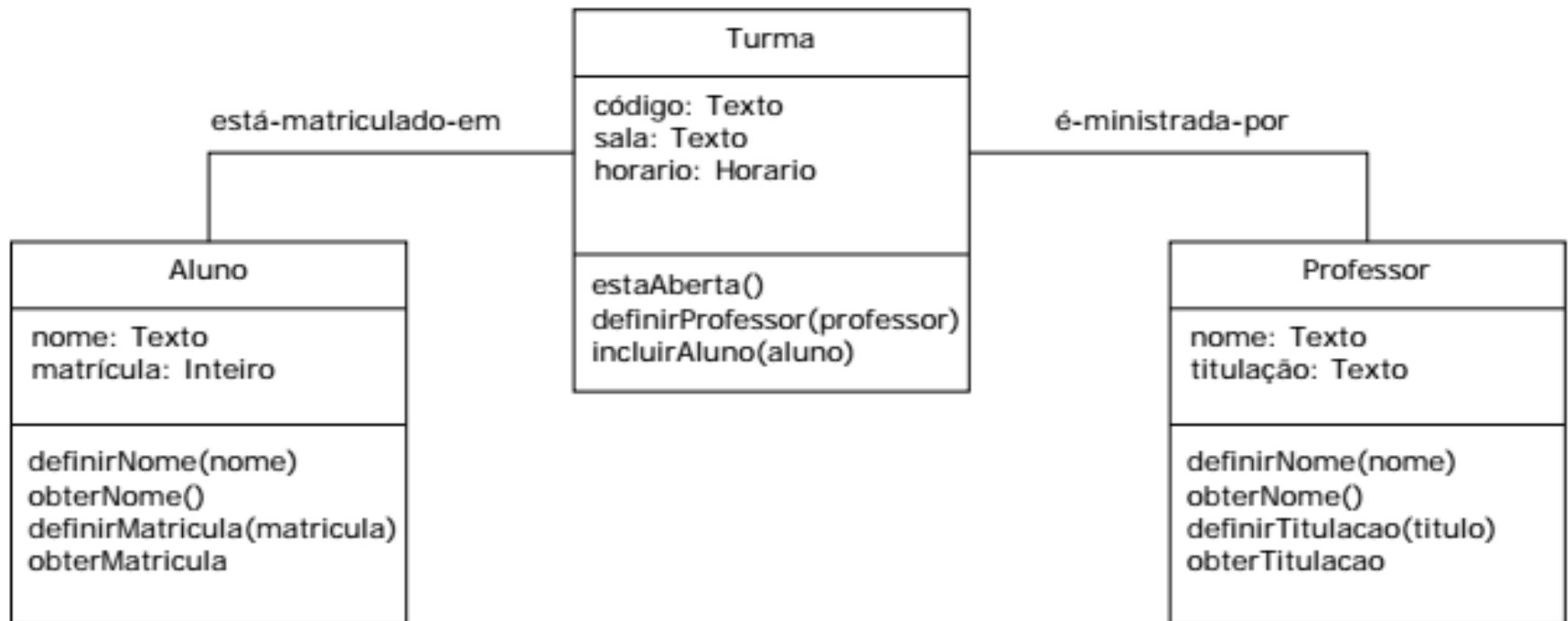


Diagrama de classes

- ❑ Diagrama mais utilizado da UML
- ❑ Mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos
- ❑ Serve de apoio para a maioria dos outros diagramas
- ❑ É o diagrama central da modelagem orientada a objetos

Primeiro diagrama de classes



Elementos de um diagrama de classes

❑ **Classes**

❑ **Relacionamentos**

- Associação
 - Agregação
 - Composição
- Generalização/Especialização
- Dependência
- Realização

Classes

- ❑ Graficamente, as classes são representadas por retângulos incluindo **nome**, **atributos** e **métodos**



- ❑ Devem receber nomes de acordo com o vocabulário do domínio do problema
- ❑ É comum adotar um padrão para nomeá-las
 - Exemplo: todos os nomes de classes serão substantivos singulares com a primeira letra maiúscula

Classes

❑ Atributos

- Representam o conjunto de características (estado) dos objetos daquela classe
- Visibilidade:
 - + público: visível para todas as entidades do sistema
 - - privado: visível somente para a própria classe
 - # protegido: visível para a própria classe e às classes descendentes
 - ~ pacote: visível por todas as entidades dentro do pacote
- Exemplo:
 - + nome : String

Classes

❑ Métodos

- Representam o conjunto de operações (comportamento) que a classe fornece
- Visibilidade:
 - + público: visível para todas as entidades do sistema
 - - privado: visível somente para a própria classe
 - # protegido: visível para a própria classe e às classes descendentes
 - ~ pacote: visível por todas as entidades dentro do pacote
- Exemplo:
 - - `getNome() : String`

Exercícios

4. Defina com suas palavras o termo modelagem de sistemas.
5. Como você poderia usar um modelo de um sistema que já existe? Explique por que nem sempre é necessário que um modelo de sistema seja completo e correto. O mesmo seria verdadeiro caso você estivesse desenvolvendo um modelo de um novo sistema?

Elementos de um diagrama de classes

❑ Classes

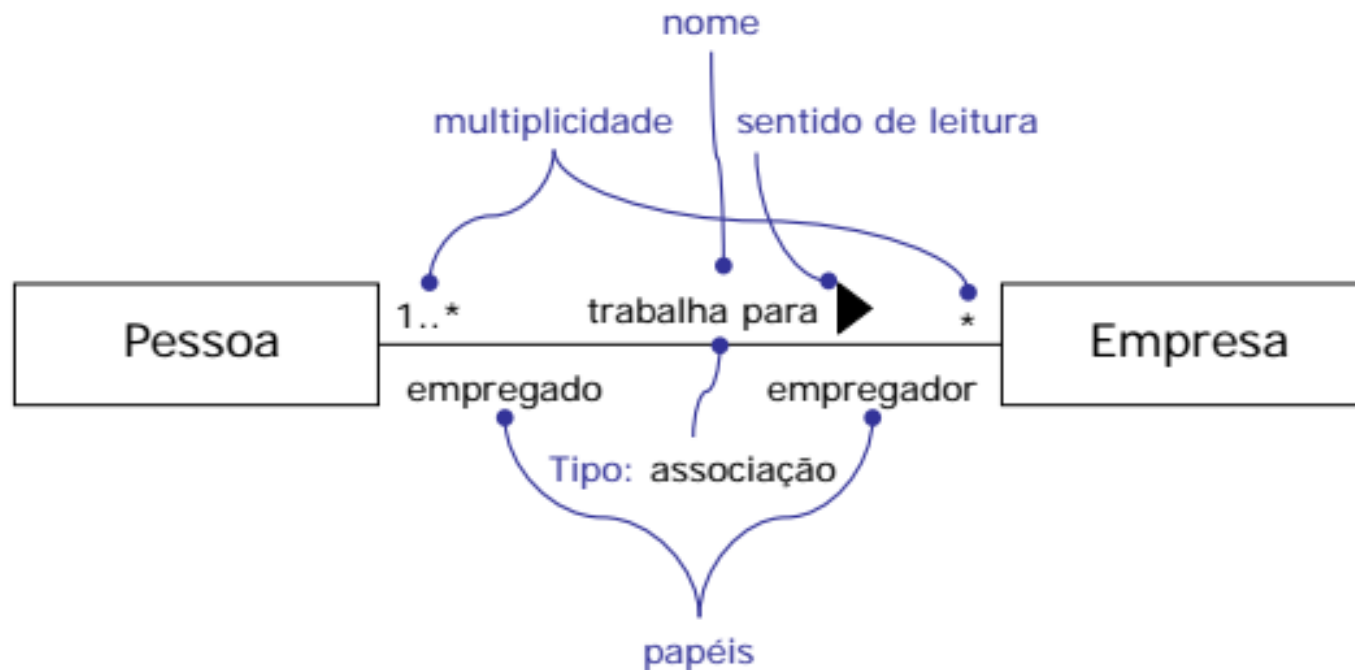
❑ Relacionamentos

- Associação
 - Agregação
 - Composição
- Generalização/Especialização
- Dependência
- Realização

Relacionamentos

- ❑ Os relacionamentos possuem:
 - **Nome:** descrição dada ao relacionamento (faz, tem, possui,...)
 - **Sentido de leitura**
 - **Navegabilidade:** indicada por uma seta no fim do relacionamento
 - **Multiplicidade:** 0..1, 0..*, 1, 1..*, 2, 3..7
 - **Tipo:** associação (agregação, composição), generalização e dependência, realização
 - **Papéis:** desempenhados por classes em um relacionamento

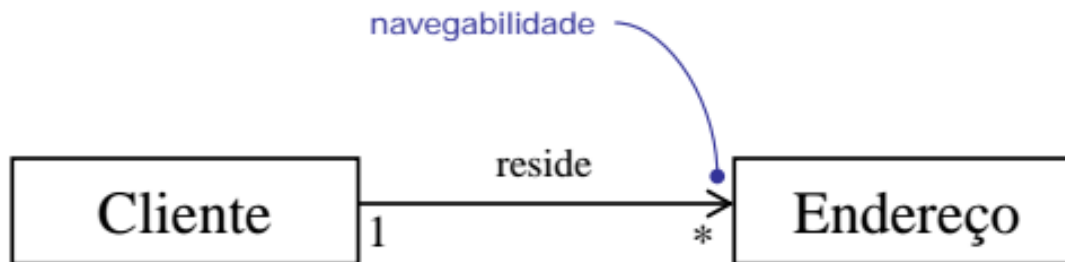
Relacionamentos



E a navegabilidade?

Relacionamentos

- ❑ O cliente sabe quais são seus endereços, mas o endereço não sabe a quais clientes pertence



Elementos de um diagrama de classes

- ❑ Classes

- ❑ **Relacionamentos**

 - **Associação**

 - Agregação
 - Composição

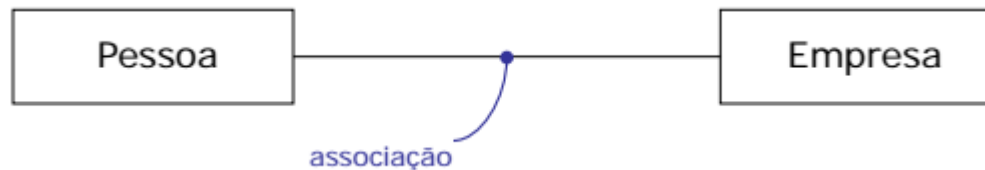
 - Generalização/Especialização

 - Dependência

 - Realização

Relacionamento: associação

- ❑ Uma associação é um relacionamento estrutural que indica que os objetos de uma classe estão vinculados a objetos de outra classe
- ❑ Uma associação é representada por uma linha sólida conectando duas classes



Relacionamento: associação

❑ Indicadores de multiplicidade:

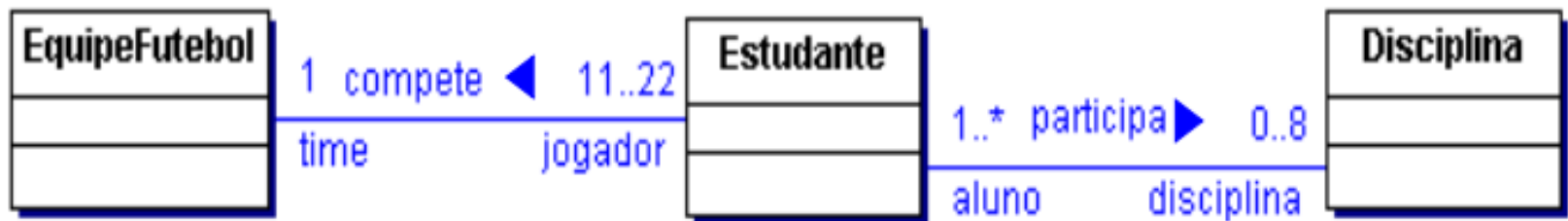
- 1 Exatamente um
- 1..* Um ou mais
- 0..* Zero ou mais (muitos)
- * Zero ou mais (muitos)
- 0..1 Zero ou um
- m..n Faixa de valores (por exemplo: 4..7)



Relacionamento: associação

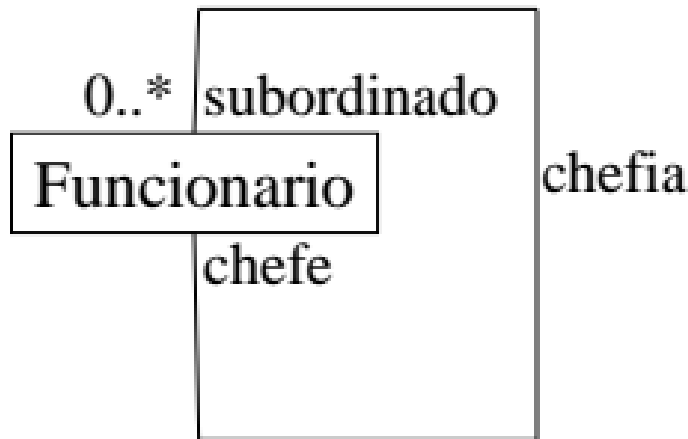
❑ Exemplo

- Um Estudante pode ser
 - Um aluno de uma Disciplina e
 - Um jogador da Equipe de Futebol
- Cada Disciplina deve ser cursada por no mínimo 1 aluno
- Um aluno pode cursar de 0 até 8 disciplinas



Relacionamento: associação

- ❑ Associações Unárias ou Reflexivas ocorrem quando há relacionamento entre objetos de uma mesma classe



Elementos de um diagrama de classes

- ❑ Classes

- ❑ **Relacionamentos**

- **Associação**

- **Agregação**

- Composição

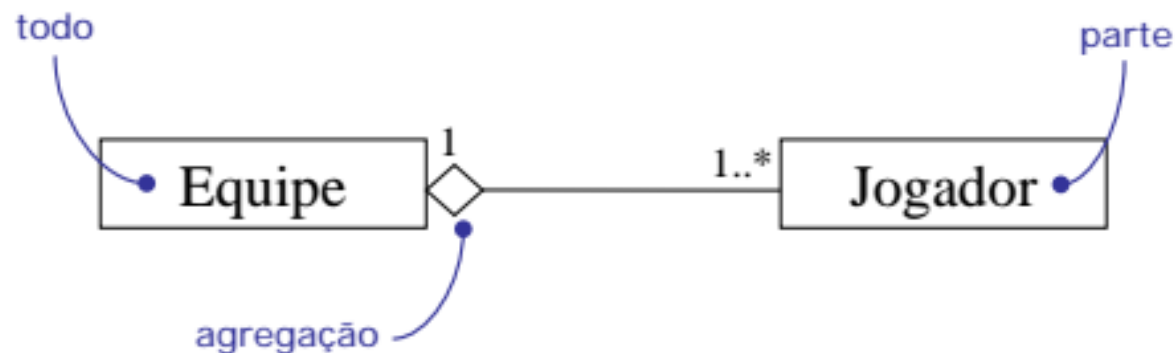
- Generalização/Especialização

- Dependência

- Realização

Relacionamentos: agregação

- ❑ É um tipo especial de associação
- ❑ Utilizada para indicar “todo-parte”
- ❑ Tempo de vida da parte não é dependente da agregação



Elementos de um diagrama de classes

- ❑ Classes

- ❑ **Relacionamentos**

 - **Associação**

 - Agregação

 - **Composição**

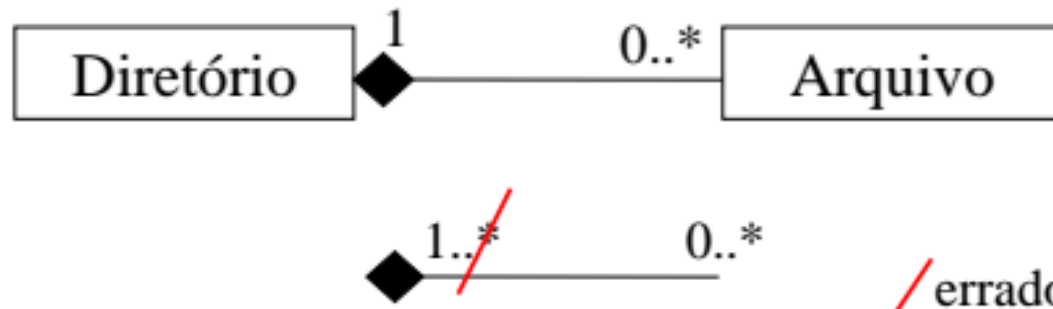
 - Generalização/Especialização

 - Dependência

 - Realização

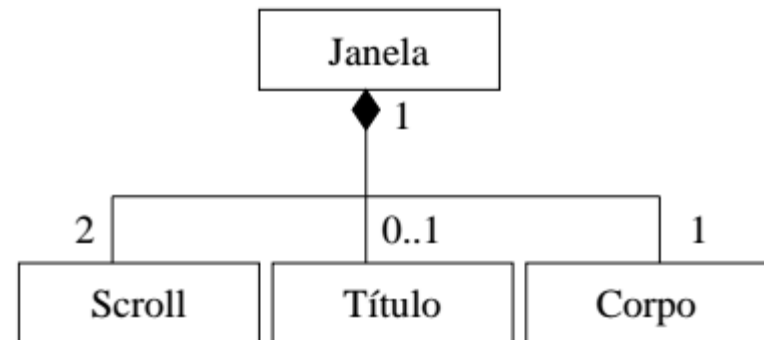
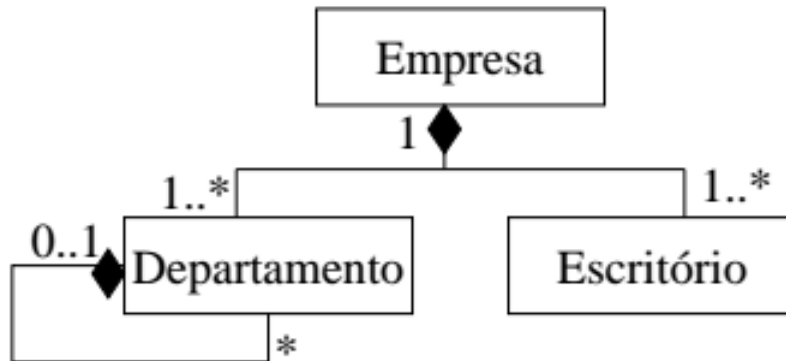
Relacionamento: composição

- ❑ É uma variante semanticamente mais “forte” da agregação
- ❑ Os objetos “parte” só podem pertencer a um único objeto “todo” e têm o seu tempo de vida coincidente com o dele
 - Quando o “todo” morre todas as suas “partes” também morrem



Relacionamento: composição

□ Exemplos:



Elementos de um diagrama de classes

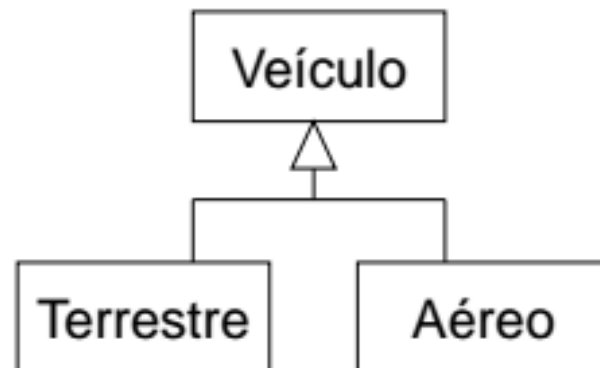
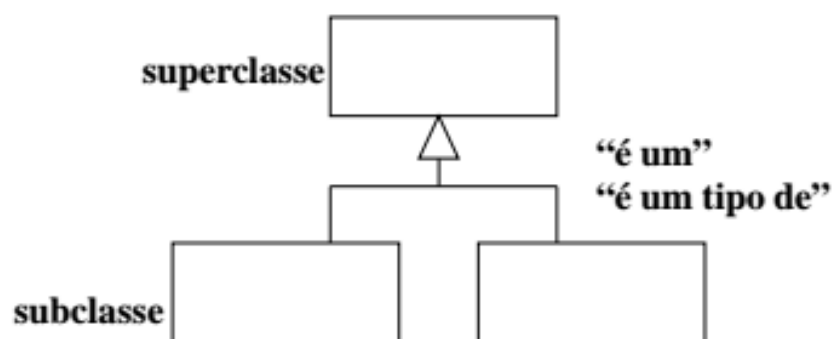
❑ Classes

❑ Relacionamentos

- Associação
 - Agregação
 - Composição
- **Generalização/Especialização**
- Dependência
- Realização

Relacionamento: generalização/especialização

- ❑ É um relacionamento entre itens gerais (superclasses) e itens mais específicos (subclasses)



Elementos de um diagrama de classes

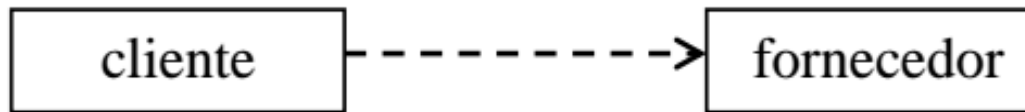
❑ Classes

❑ Relacionamentos

- Associação
 - Agregação
 - Composição
- Generalização/Especialização
- **Dependência**
- Realização

Relacionamento: dependência

- ❑ Deixa explícito que mudanças em uma classe podem gerar consequências em outra classe



- ❑ Obs:
 - A classe cliente depende de algum serviço da classe fornecedor
 - A mudança de estado do fornecedor afeta o objeto cliente
 - Fornecedor é recebido por parâmetro de método

Elementos de um diagrama de classes

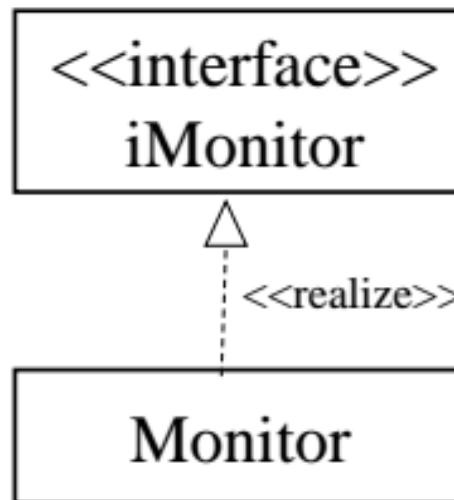
❑ Classes

❑ Relacionamentos

- Associação
 - Agregação
 - Composição
- Generalização/Especialização
- Dependência
- **Realização**

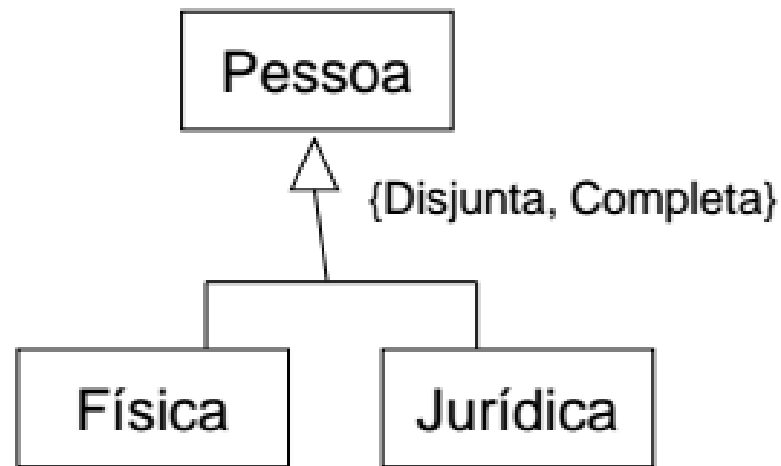
Relacionamento: realização

- ❑ Utilizado para indicar que uma classe implementa uma interface



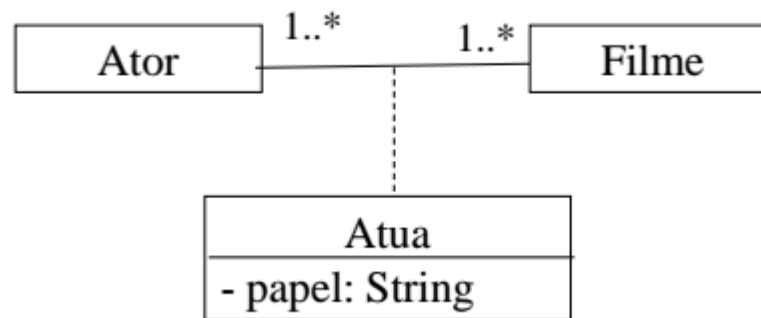
Relacionamentos

- ❑ Restrições definem condições a serem validadas durante a implementação



Classe de associação

- ❑ Classe de associação (classe associativa) ocorre quando há propriedades relacionadas à associação que precisamos guardar
 - Produzida pela associação muitos (*) para muitos (*) entre duas classes

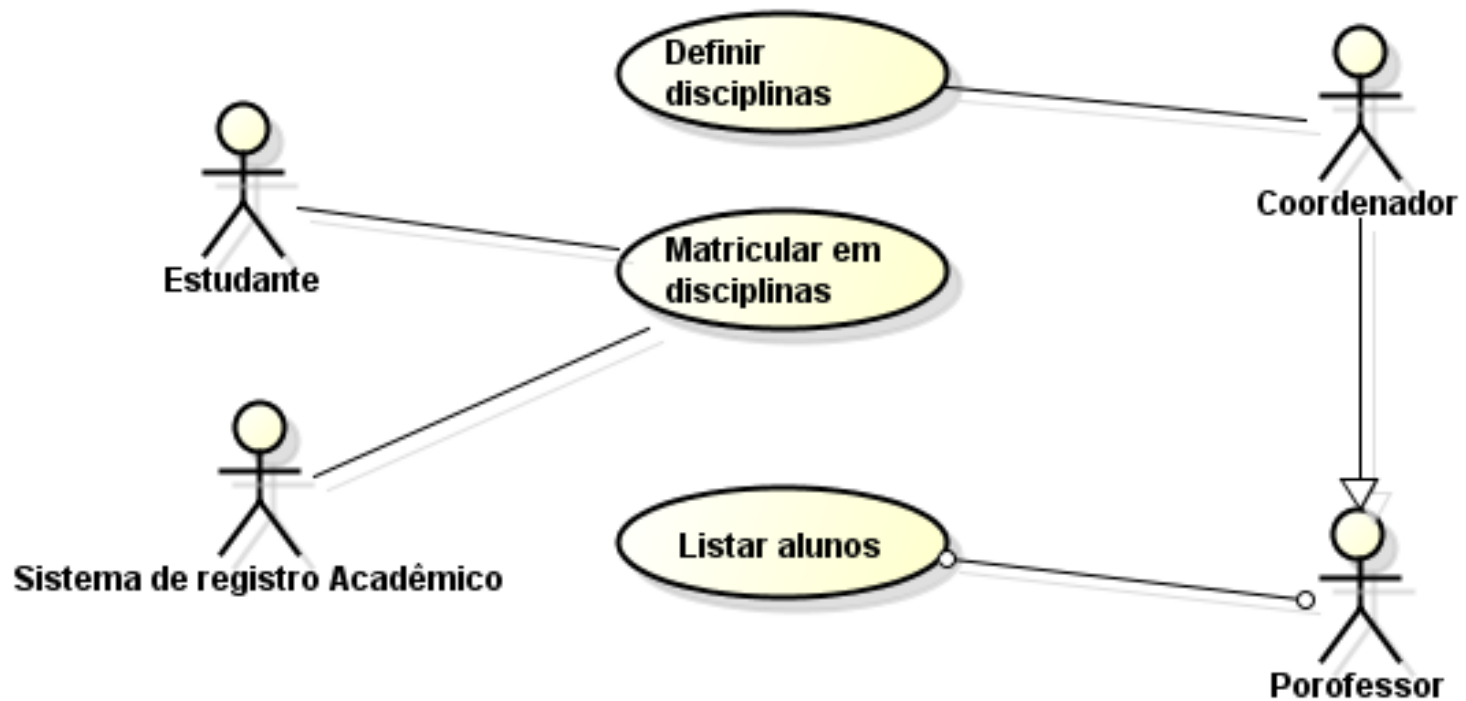


Exemplo – Sistema de Matrícula

- ❑ A Universidade XYZ deseja informatizar seu sistema de matrículas:
 - A universidade oferece vários cursos.
 - O Coordenador de um curso define as disciplinas que serão oferecidas pelo seu curso num dado semestre.
 - Várias disciplinas são oferecidas em um curso.
 - Várias turmas podem ser abertas para uma mesma disciplina, porém o número de estudantes inscritos deve ser entre 3 e 10.
 - Estudantes selecionam 4 disciplinas.
 - Quando um estudante matricula-se para um semestre, o Sistema de Registro Acadêmico (SRA) é notificado.
 - Após a matrícula, os estudantes podem, por um certo prazo, utilizar o sistema para adicionar ou remover disciplinas.
 - Professores usam o sistema para obter a lista de alunos matriculados em suas disciplinas.
 - Todos os usuários do sistema devem ser validados.

Exemplo – Sistema de Matrícula

❑ Diagrama de caso de uso



Exemplo – Sistema de Matrícula

❑ Descrição do caso de uso “Matricular em Disciplina”

- Esse caso de uso se inicia quando o Estudante de Curso inicia uma sessão no sistema e apresenta suas credenciais.
- O sistema verifica se a credencial é válida.
- O sistema solicita que o estudante realize sua matrícula, selecionando 4 disciplinas.
- O estudante preenche um formulário eletrônico de matrícula e o submete para uma análise de consistência.
- O sistema analisa as informações contidas no formulário.
- Se as informações são consistentes, o estudante é incluído em turmas abertas de 4 disciplinas, iniciando pelas preferenciais.
- Se as informações não são consistentes, o sistema informa o motivo da inconsistência e solicita que o formulário seja alterado.

Exemplo – Sistema de Matrícula

- ❑ Diagrama de classes: identificando as classes

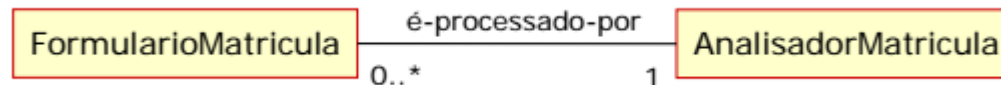


Exemplo – Sistema de Matrícula

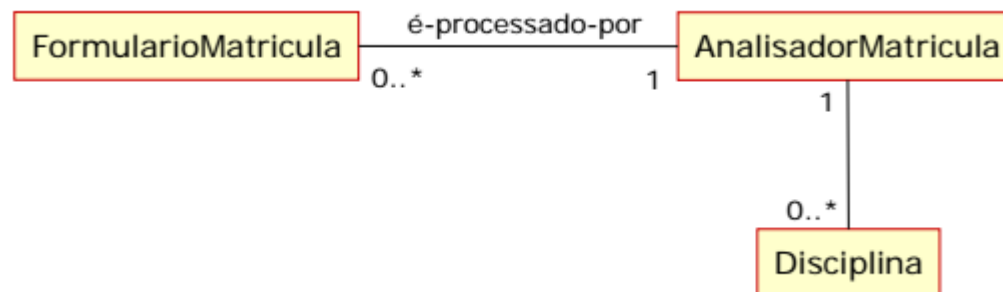
- ❑ Diagrama de classes: identificando os relacionamentos
 - Exemplos de candidatos a relacionamentos:
 - **A** é parte física ou lógica de **B**
 - **A** está contido fisicamente ou logicamente em **B**
 - **A** é uma descrição de **B**
 - **A** é membro de **B**
 - **A** é subunidade organizacional de **B**
 - **A** usa ou gerencia **B**
 - **A** se comunica/interage com **B**
 - **A** está relacionado com uma transação **B**
 - **A** é possuído por **B**
 - **A** é um tipo de **B**

Exemplo – Sistema de Matrícula

- ❑ Diagrama de classes: identificando os relacionamentos
 - O formulário de matrícula é processado por um analisador de matrícula

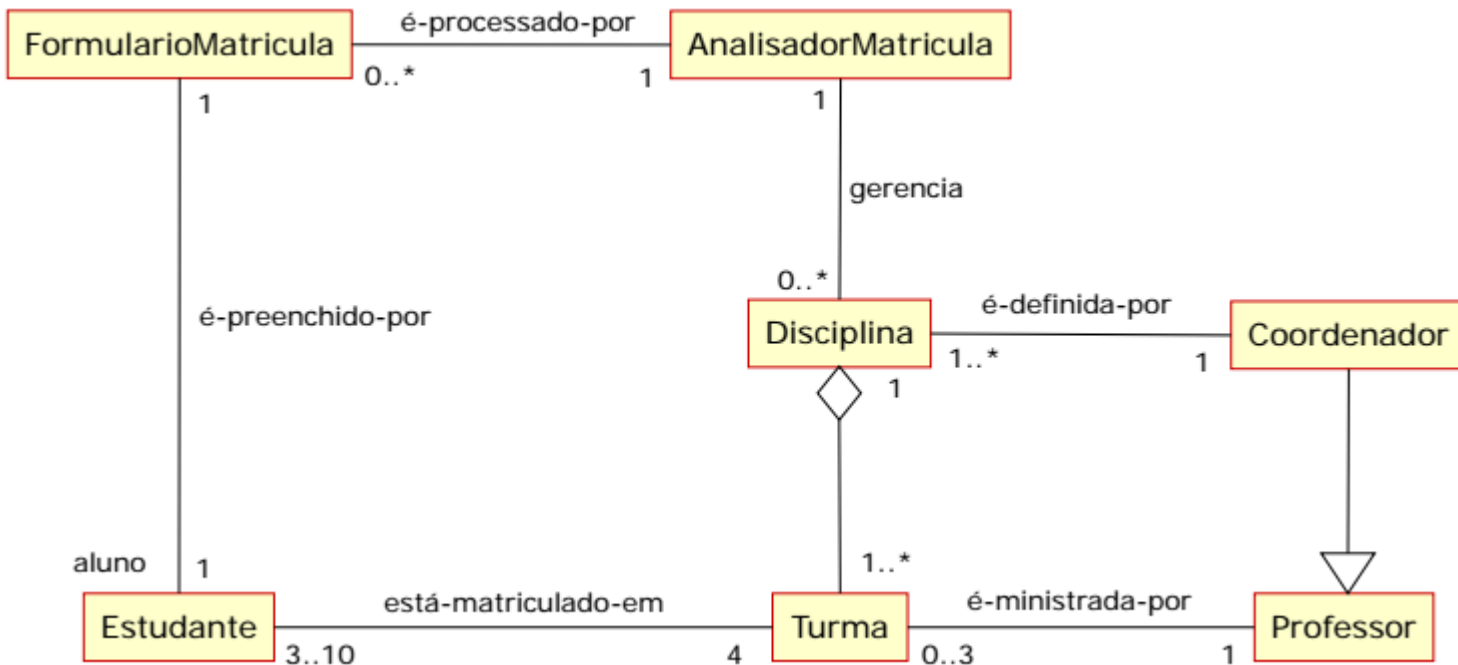


- O analisador de matrícula gerencia a disciplina



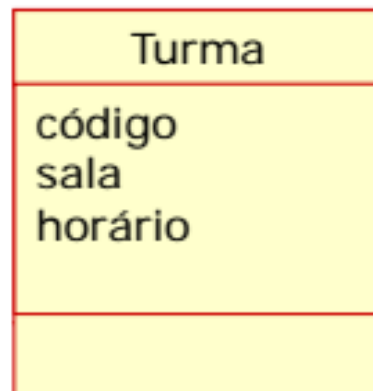
Exemplo – Sistema de Matrícula

□ Diagrama de classes



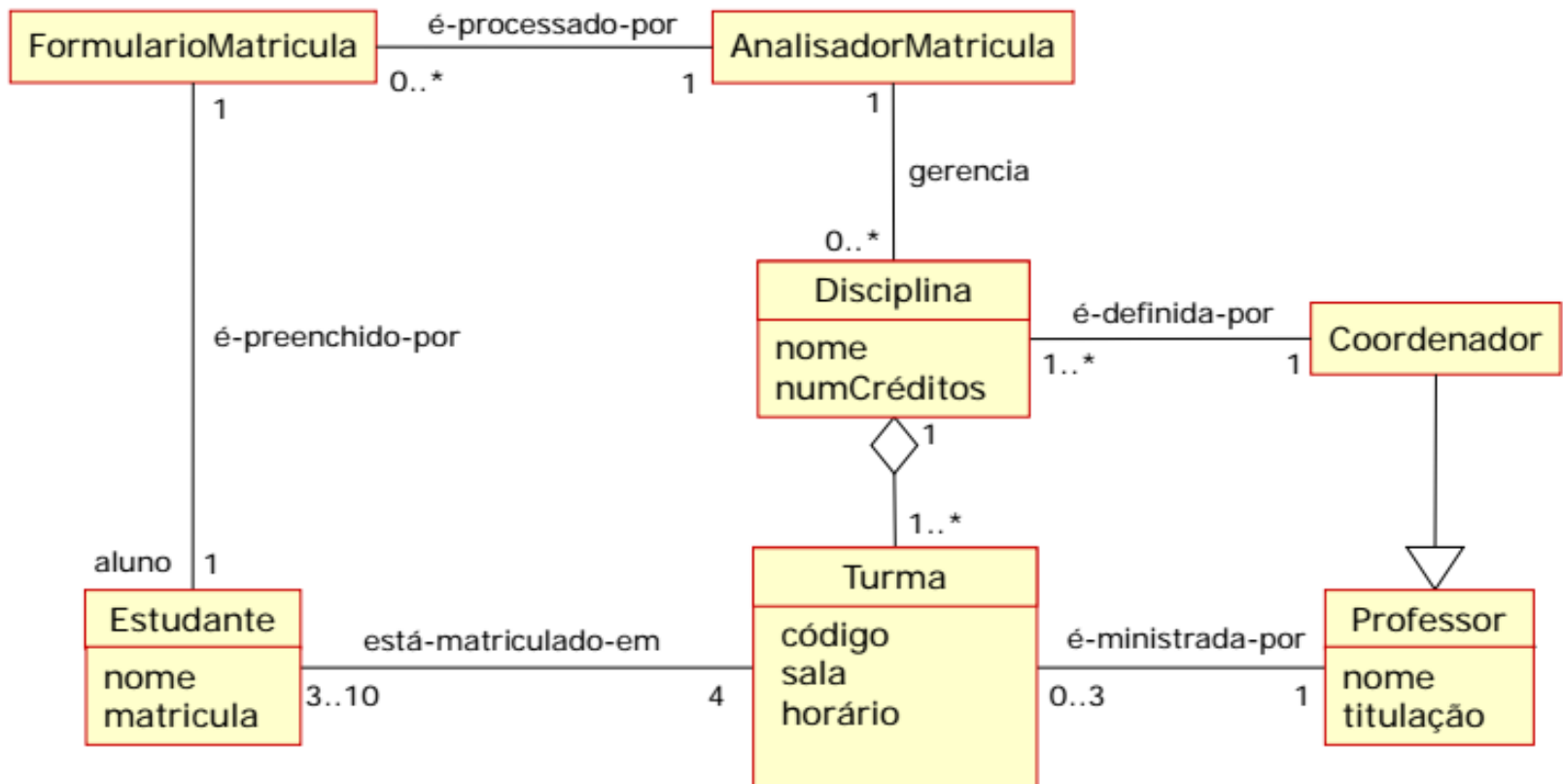
Exemplo – Sistema de Matrícula

- ❑ Diagrama de classes: identificando os atributos
 - Os atributos podem ser encontrados examinando-se as descrições dos casos de uso e também pelo conhecimento do domínio do problema
 - Cada turma oferecida possui um código, uma sala e um horário



Exemplo – Sistema de Matrícula

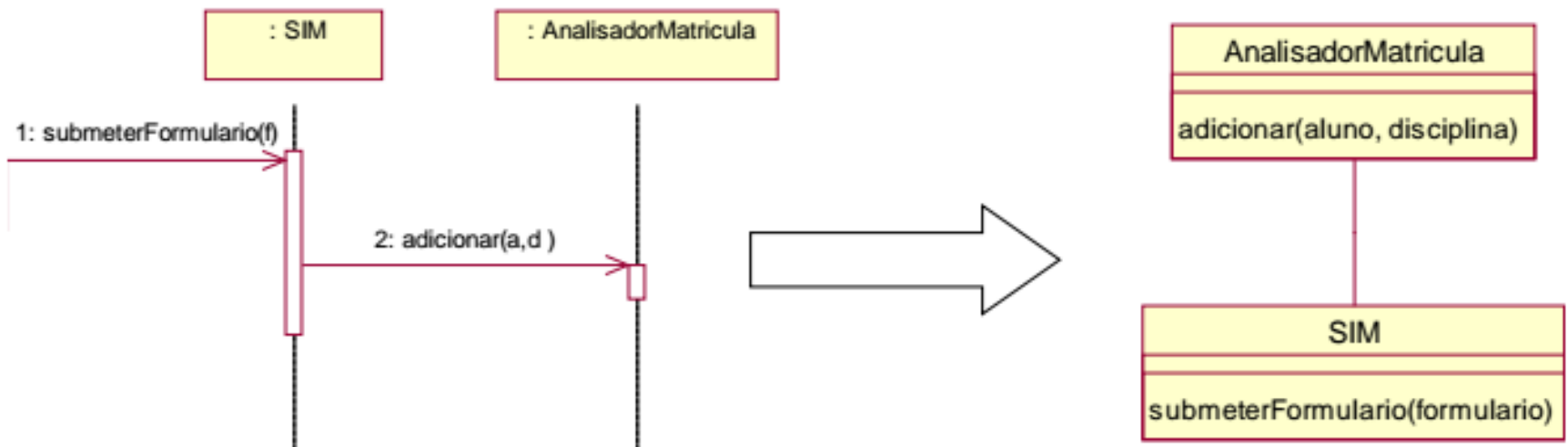
□ Diagrama de classes



Exemplo – Sistema de Matrícula

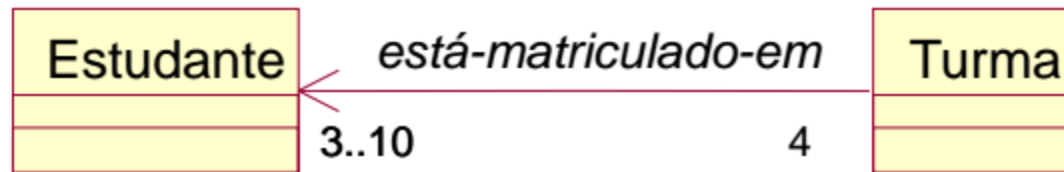
❑ Diagrama de classes

- Identificando os métodos
 - Somente depois de modelar os diagramas de sequência



Exemplo – Sistema de Matrícula

- ❑ Diagrama de classes
 - E a navegabilidade?



```
public class Estudante {
    private String nome;
    private String matricula;
    ...
}
```

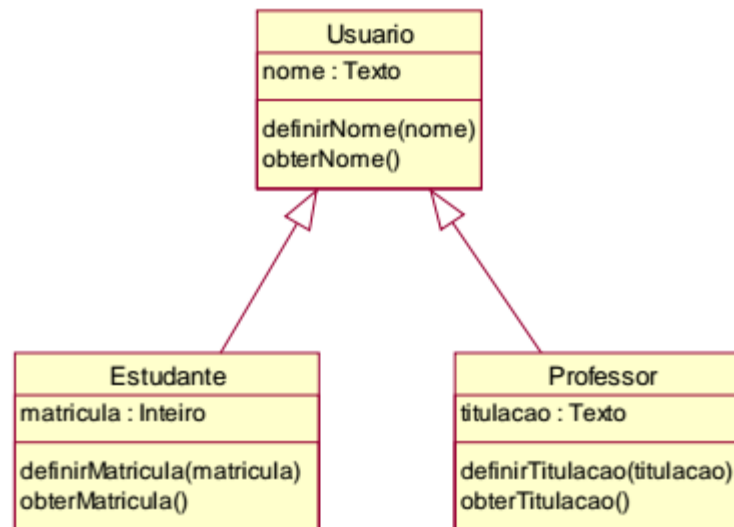
```
public class Turma {
    private String codigo;
    private String sala;
    private Estudante alunos[];
    ...
}
```

Obs: Turma não aparece como atributo de Estudante!

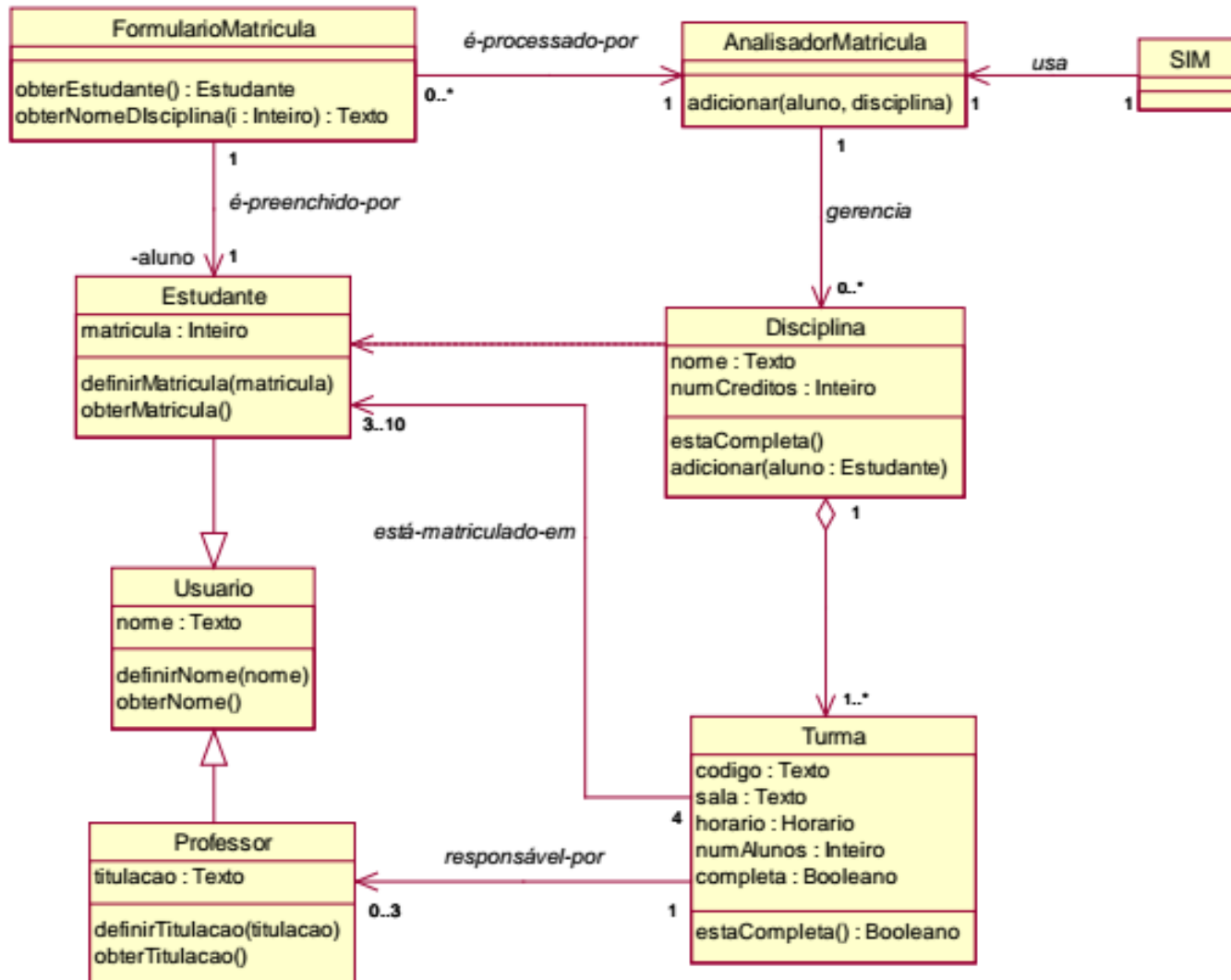
Exemplo – Sistema de Matrícula

❑ Diagrama de classes

- Acrescentando generalizações:
 - Atributos, operações e/ou relacionamentos comuns podem ser movidos para uma classe mais geral



Exemplo – Sistema de Matrícula



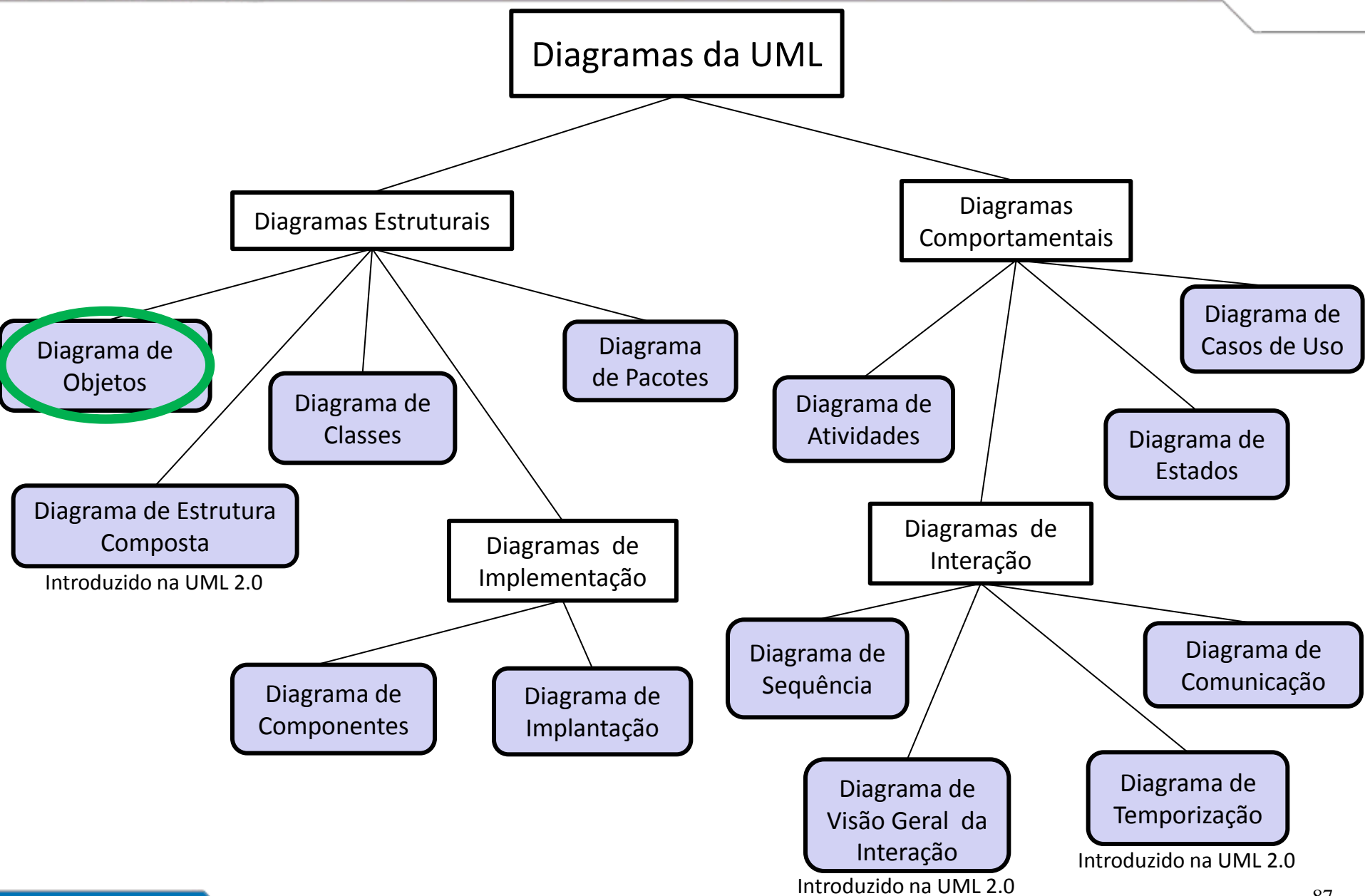
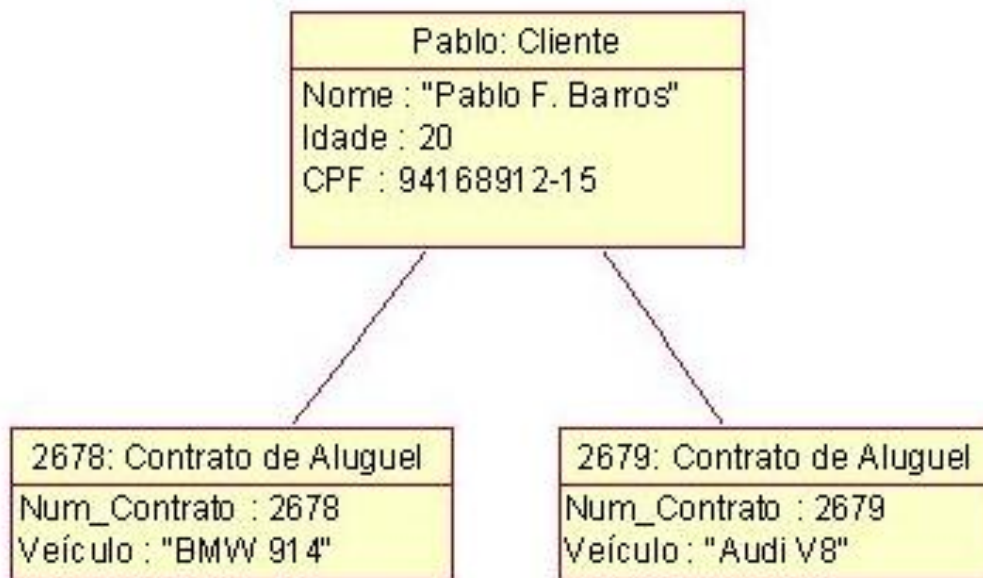


Diagrama de objetos

- ❑ Complemento do diagrama de classes
- ❑ Exibe os valores armazenados pelos objetos de um diagrama de classes

Diagrama de objetos



Exercícios

6. Construa os diagramas de caso de uso e de classes tendo como referência a tela de reserva de carro apresentada a seguir considerando que somente clientes já cadastrados poderão fazer a reserva do veículo *online*.

Exercícios



Old Locadora

Tela Reservar Carro

CPF cliente:

OK

Dados Cliente:

Nome:

XXXXXXXXXX

Endereço:

XXXXXXXXXX

Telefone:

XXXXXXXXXX

Selecione Modelo Veículo:



Preço Diária:

R\$ XXXXXXXX

OK

Período Aluguel:

Início:

Término:

Veículo Disponível:

<Lista veículos>

Placa:

XXXXXXXXXX

Ano:

XXXXXXXXXX

Descrição:

XXXXXXXXXX

OK

Valor Aluguel:

R\$ XXXXXXXX

Reservar

Cancelar


Observações:

- Só veículos com "Situação = Disponível" serão selecionados;
- Os campos em branco são para preenchimento do ator envolvido;
- Os campos com "XXXXXX" são respostas do sistema a entrada do usuário.

Exercícios

7. Construa os diagramas de caso de uso e de classes tendo como referência a tela de cadastramento apresentada a seguir. Observe que esta mesma tela deverá ser preenchida por dois atores diferentes: primeiro o Funcionário, depois o Gerente registrando os preços de compra e venda.

Exercícios

**Brain Livraria**


Tela Cadastrar Livro


Preenchimento pelo Funcionário


ISBN:

Título:

Sinopse:

CD:  +

Autor:  +

Editora:  +

Preenchimento pelo Gerente

Preço de Compra: R\$

Taxa Acréscimo: %

Preço de Venda: R\$

Sistema Operacional:

Autor (es):

Dados Editora: Nome:
Contato:
Telefone:

Observações:

- Os campos em branco são para preenchimento do ator envolvido;
- Os campos com "xxxxxx" são respostas do sistema a entrada do usuário;
- O sinal de + indica que mais um item pode ser cadastrado.

Exercícios

8. Rafaela possui vários temas de festas infantis para aluguel. Ela precisa controlar os aluguéis e para isso quer uma aplicação que permita cadastrar o nome e o telefone do cliente, o endereço completo da festa, o tema escolhido, a data da festa, a hora de início e término da festa. Além disso, para alguns clientes antigos, Rafaela oferece descontos. Sendo assim, é preciso saber o valor realmente cobrado num determinado aluguel. Para cada tema, é preciso controlar a lista de itens que compõem o tema (ex. castelo, boneca da cinderela, bruxa, etc), o valor do aluguel e a cor da toalha da mesa que deve ser usada com o tema. Desenhe os diagramas de casos de uso e de classes.

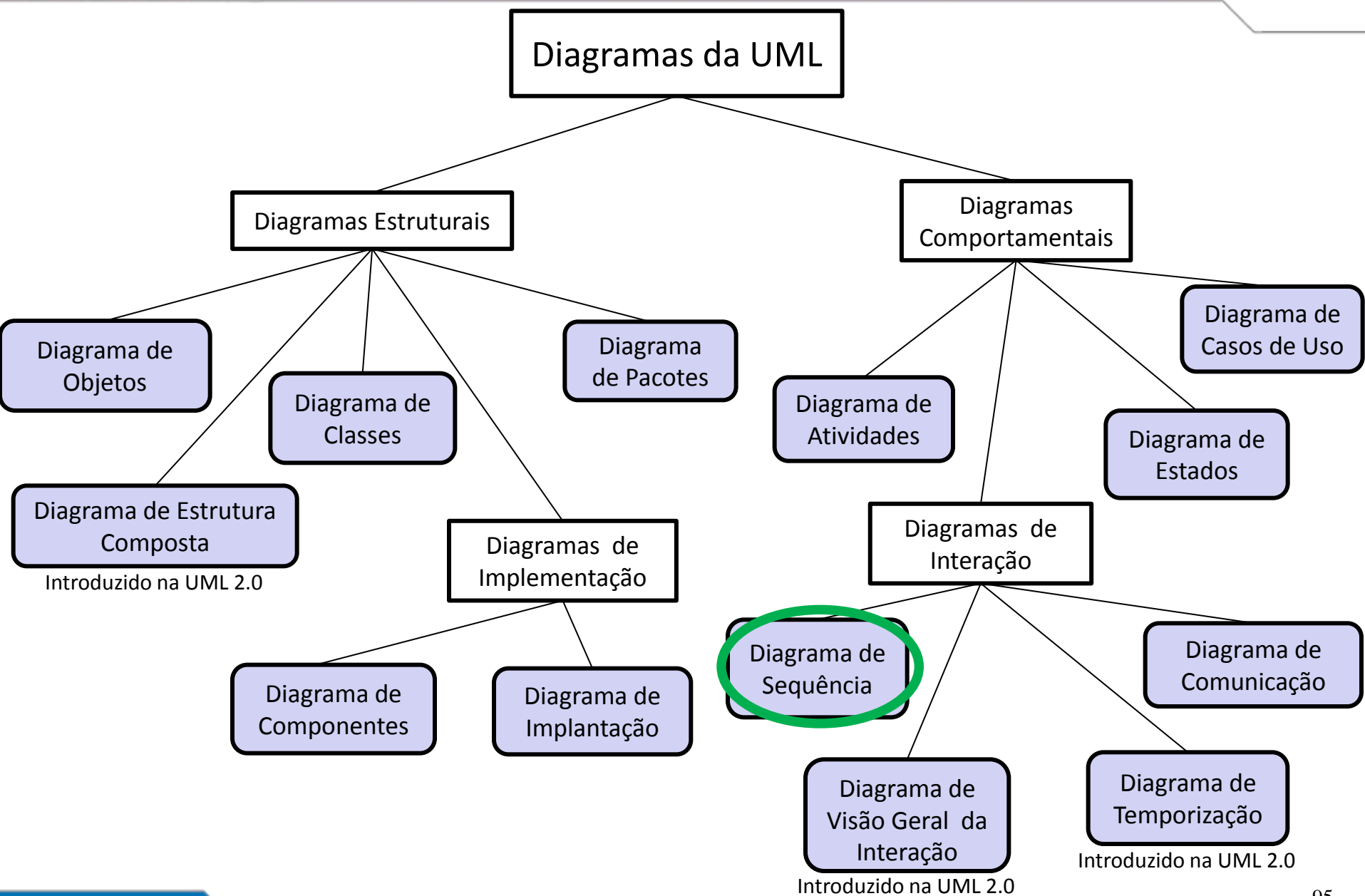
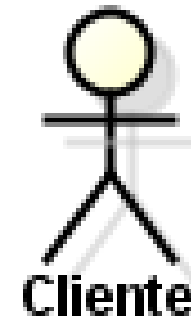


Diagrama de sequência

- ❑ Mostra como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização de uma operação
- ❑ Pode ser usado para detalhar um caso de uso
- ❑ Identifica
 - Ator responsável pelo evento
 - Objetos envolvidos na ação

Atores

- ❑ Exatamente os mesmos descritos no diagrama de casos de uso
- ❑ Entidade externas que
 - Interagem com o sistema
 - Solicitam serviços



Objetos

- ❑ Indicam instâncias de uma classe envolvida no processo
 - As classes são mostradas no diagrama de classes

- ❑ Representados por retângulos
 - Nome do objeto (em minúsculo)
 - Nome da classe (inicial maiúscula)

Diagrama de sequência

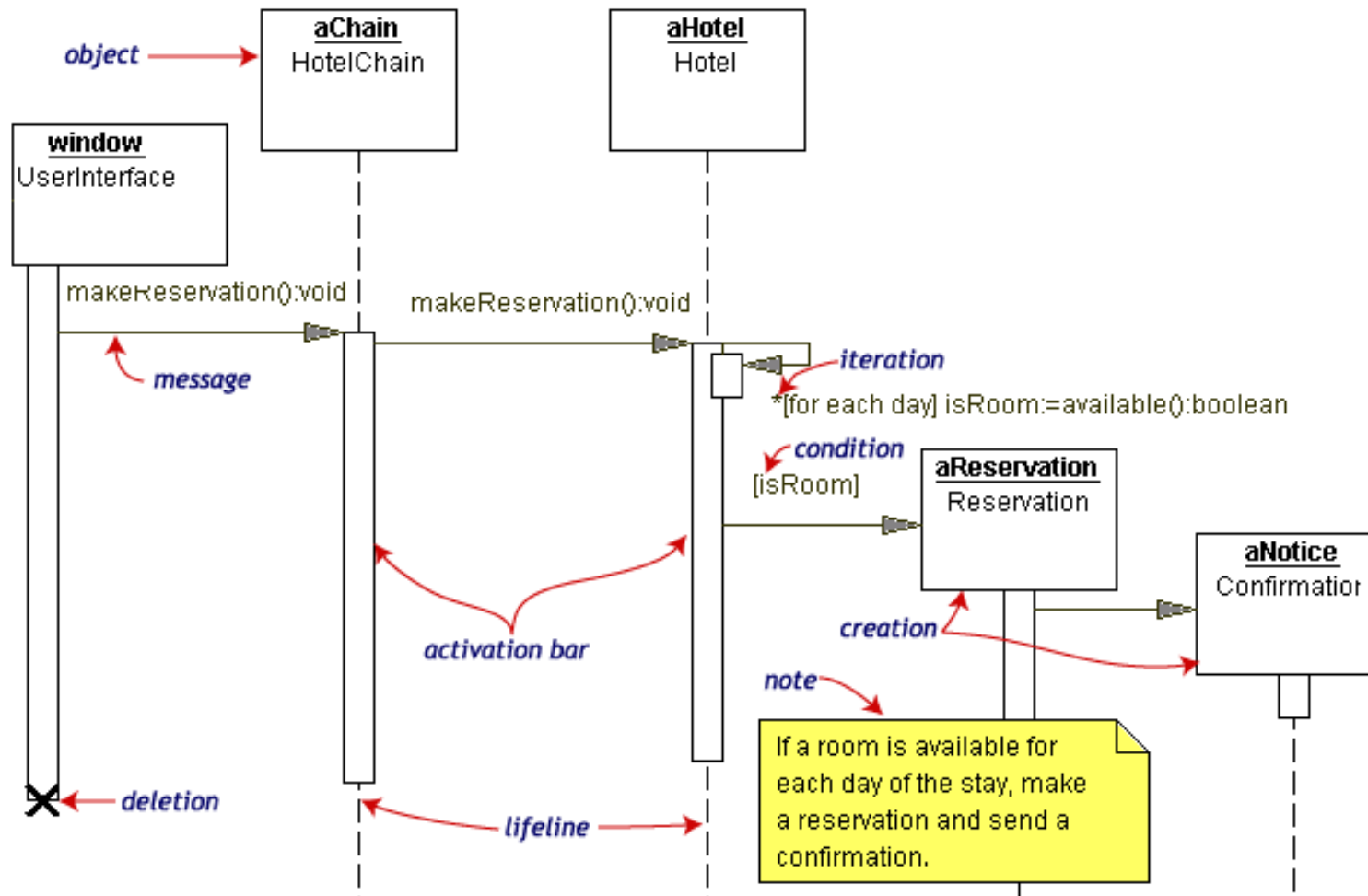


Diagrama de sequência

- ❑ Elementos de um diagrama de sequência:
 - Linhas verticais representando o tempo de vida de um objeto (*lifeline*)
 - As linhas verticais são preenchidas por barras verticais que indicam exatamente quando um objeto passou a existir
 - Quando um objeto desaparece, existe um "X" na parte inferior da barra

Diagrama de sequência

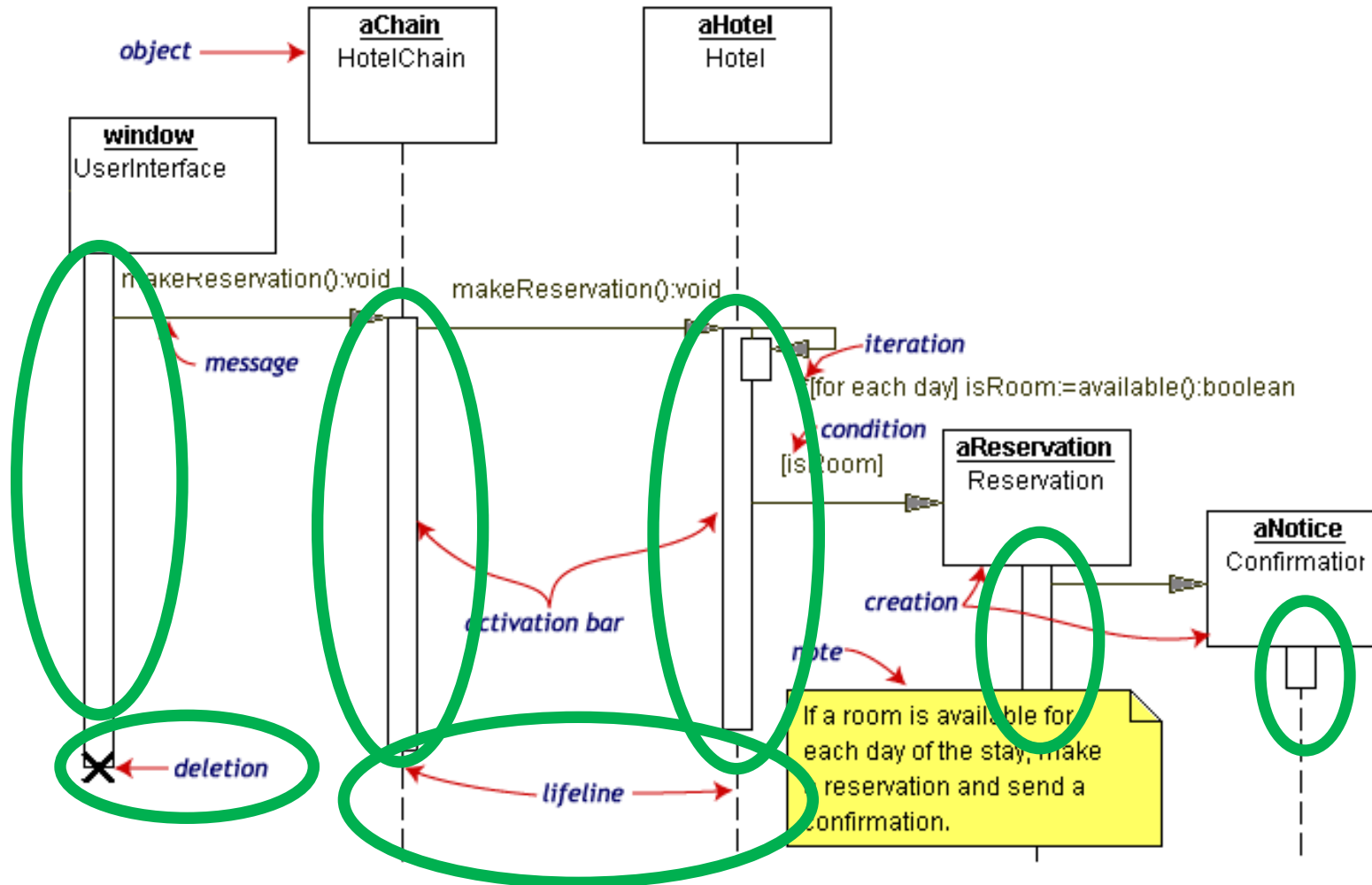


Diagrama de sequência

- ❑ Podem haver vários períodos em que o objeto se encontra ativo

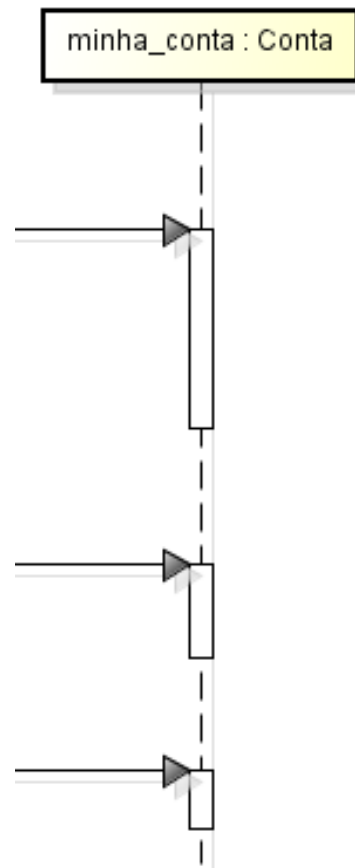
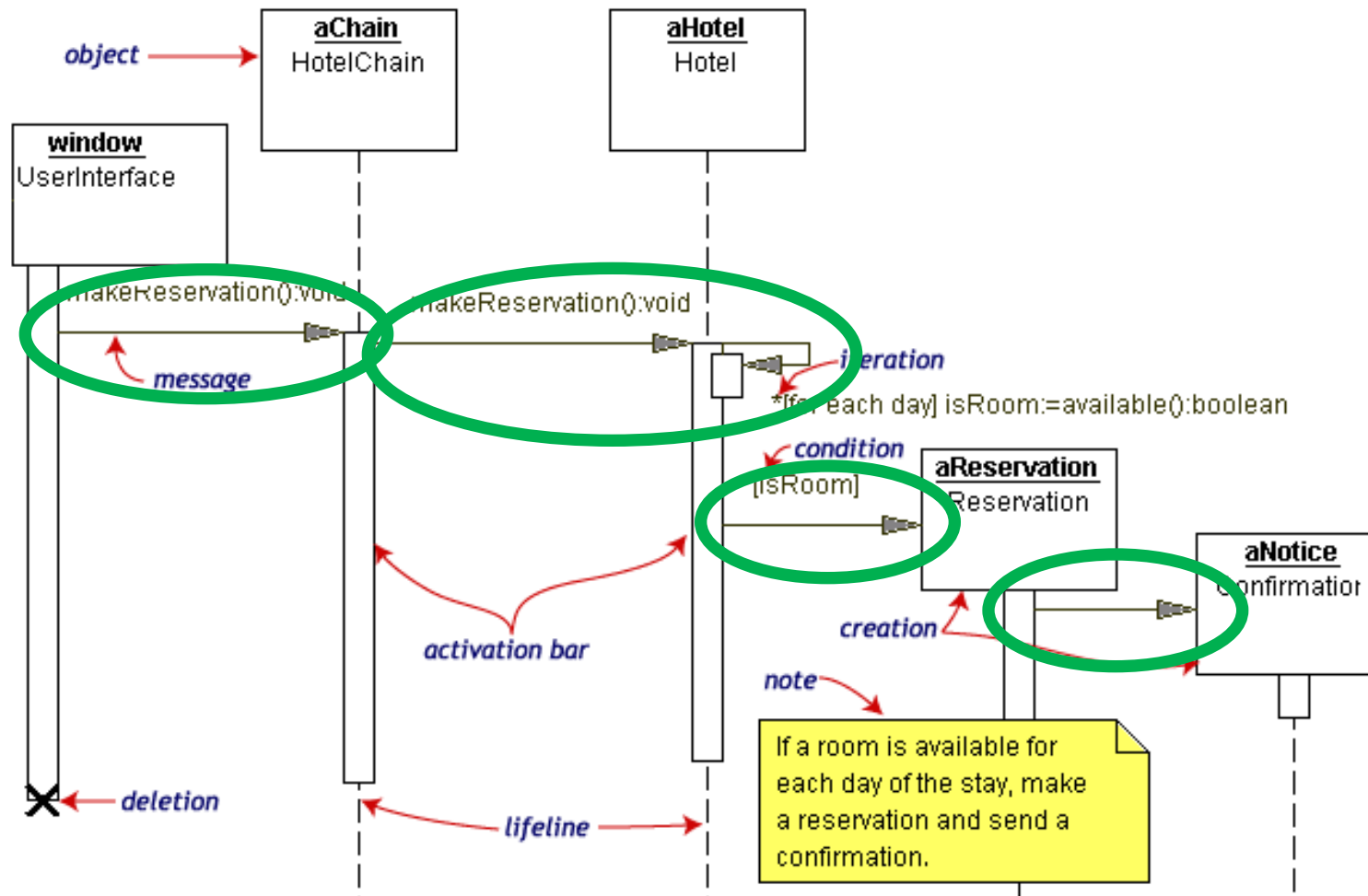


Diagrama de sequência

- ❑ Elementos de um diagrama de sequência :
 - Linhas horizontais ou diagonais representando mensagens trocadas entre objetos
 - Estas linhas são acompanhadas de um rótulo que contém o nome da mensagem e, opcionalmente, os parâmetros da mesma
 - Também podem existir mensagens enviadas para o mesmo objeto, representando uma iteração

Diagrama de sequência



Mensagens no diagrama de sequência

- ❑ Representam a comunicação entre objetos e/ou atores do diagrama de sequência
- ❑ Exemplos de mensagens
 - Chamadas de um método de um objeto por outro objeto
 - Comunicação entre dois atores

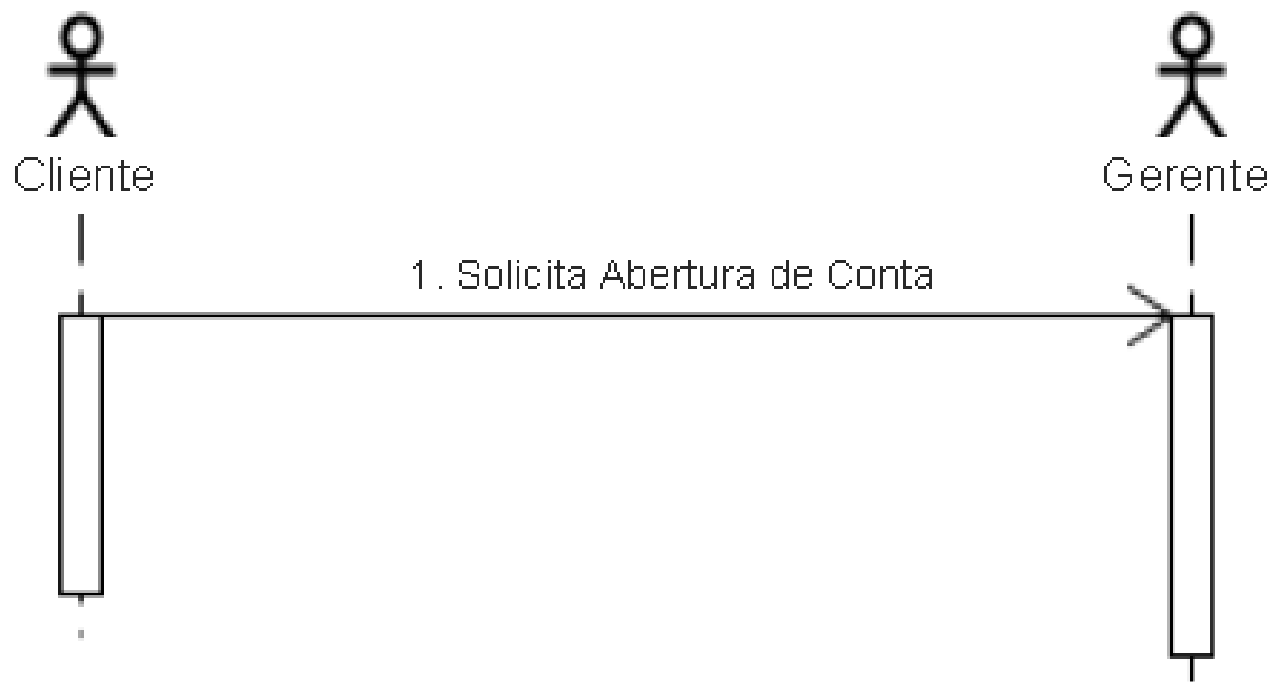
Tipos de mensagens

- ❑ Ator para ator
- ❑ Ator para objeto
- ❑ Objeto para objeto
- ❑ Objeto para ator

Mensagens de ator para ator

- ❑ Indica a conversa entre atores
- ❑ Os atores podem não fazer parte do sistema
 - Mas, facilita a compreensão do processo
- ❑ Não é muito comum de se modelar

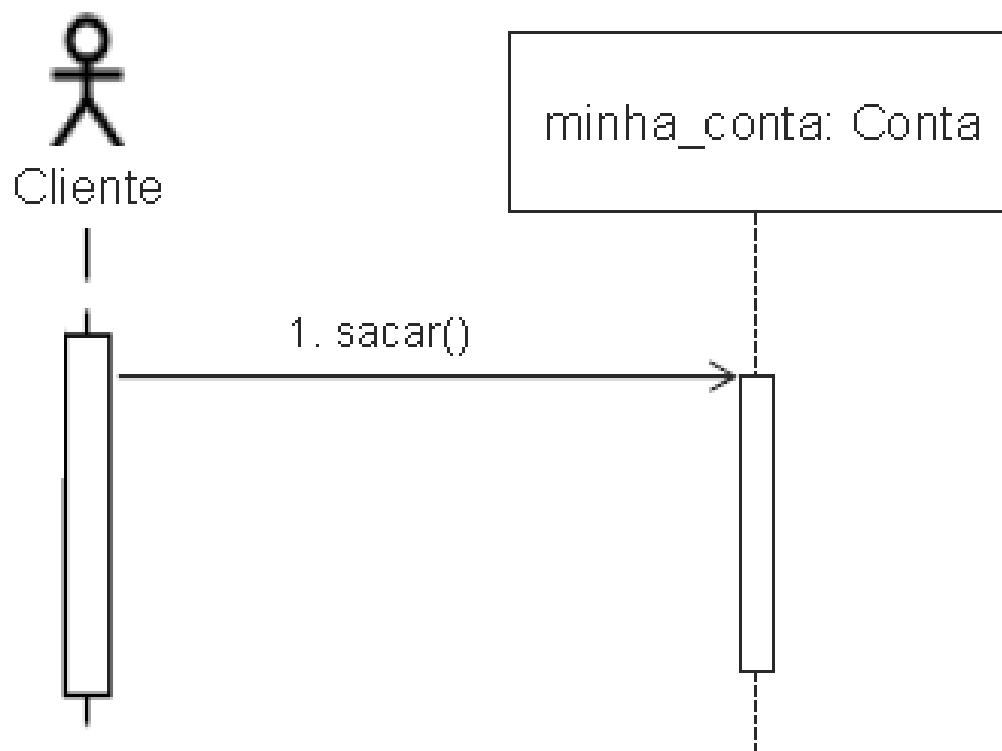
Comunicação entre atores



Mensagem ator para objeto

- ❑ Indica uma solicitação de serviço feita pelo ator ao sistema
- ❑ O ator produz um evento que força o disparo de um método
- ❑ Tipo comum quando se modela casos de uso

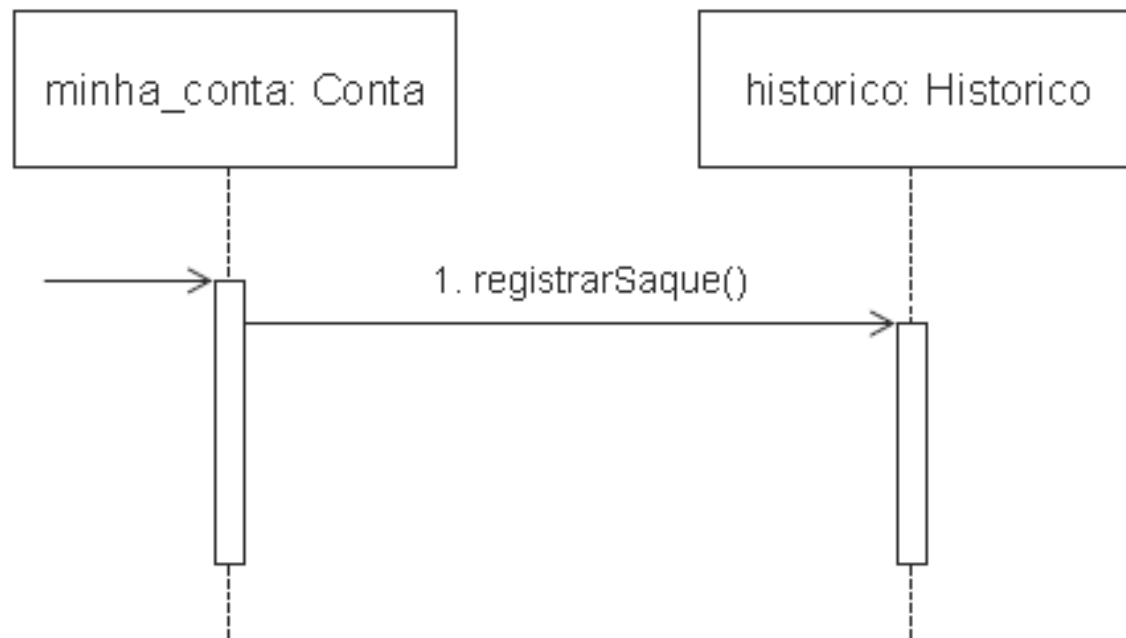
Comunicação ator para objeto



Mensagem objeto para objeto

- ❑ Indica que um objeto transmite uma mensagem para outro objeto
 - Exemplo, solicitando a execução de um método
- ❑ Tipo mais comum de troca de mensagens

Comunicação entre objetos

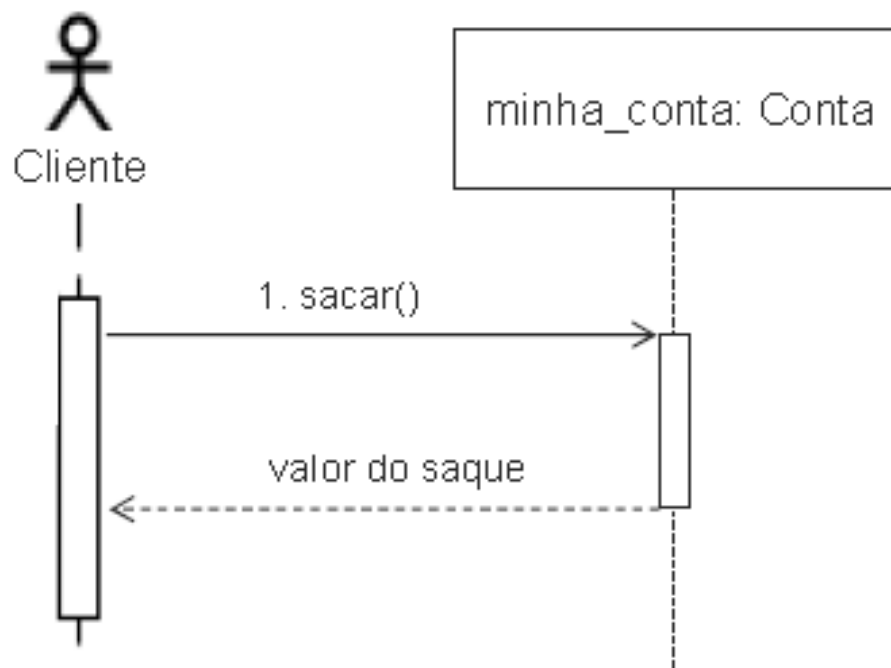


Mensagem objeto para ator

- ❑ Indica a resposta de uma solicitação de serviço feita pelo ator
 - Ocorre normalmente quando o objeto envia uma **mensagem de retorno**
- ❑ Mensagens de retorno são representadas por linhas tracejadas
 - Pode conter legenda indicando o retorno



Comunicação entre objeto e ator



Mensagem de retorno

- ❑ Além de resposta ao ator, mensagens de retorno podem indicar respostas para objetos
 - Pode retornar informações específicas do método chamado
- ❑ Mensagens de retorno são opcionais em diagramas de sequência

Mensagem de retorno

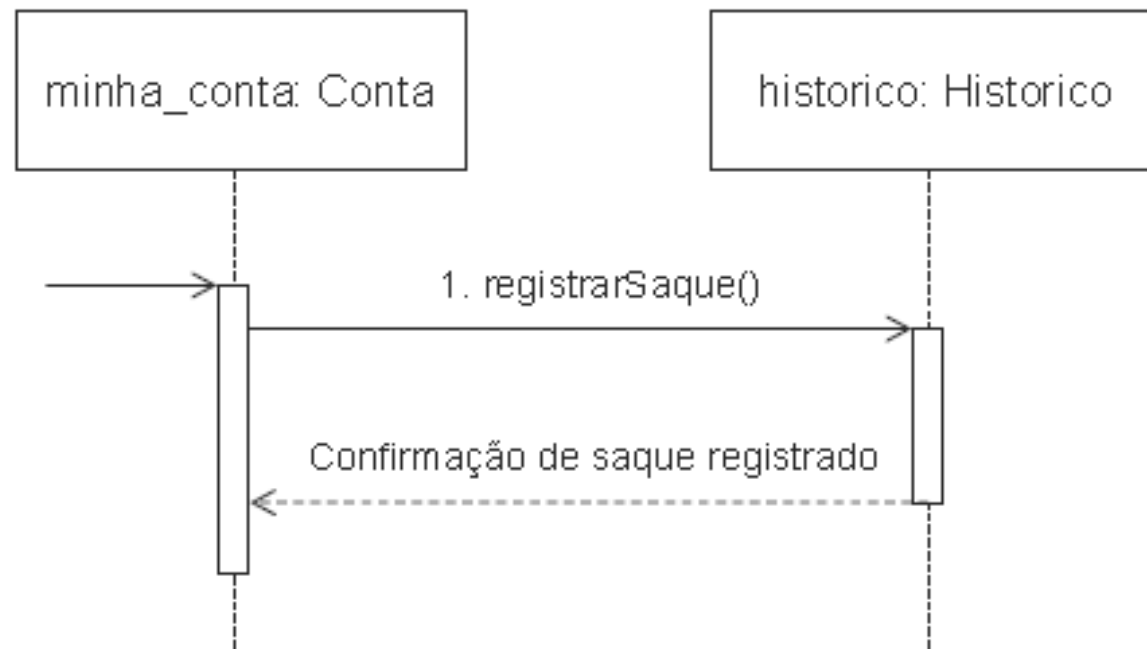
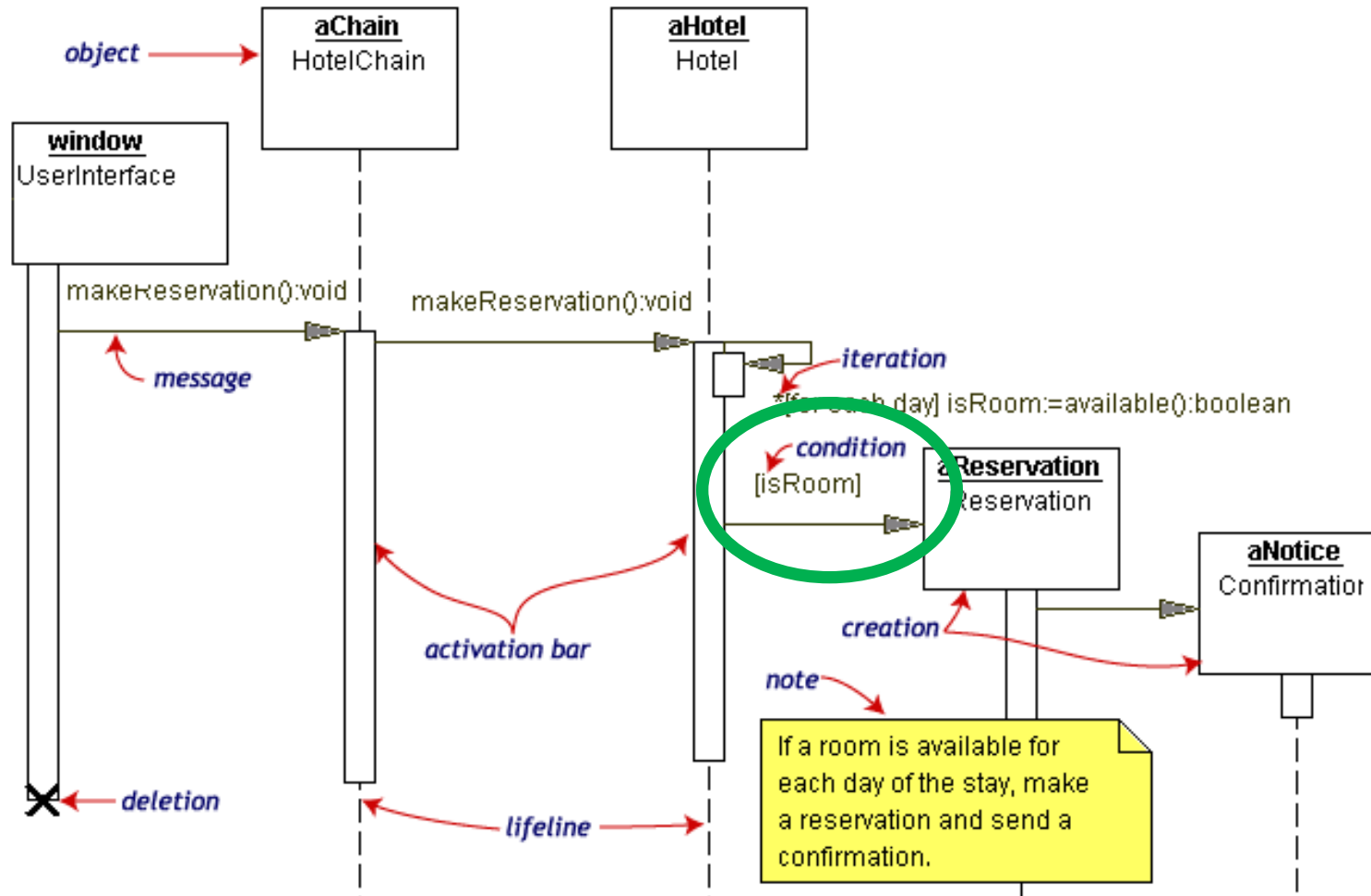


Diagrama de sequência

- ❑ Elementos de um diagrama de sequência :
 - Uma condição é representada por uma mensagem cujo rótulo é envolvido por colchetes

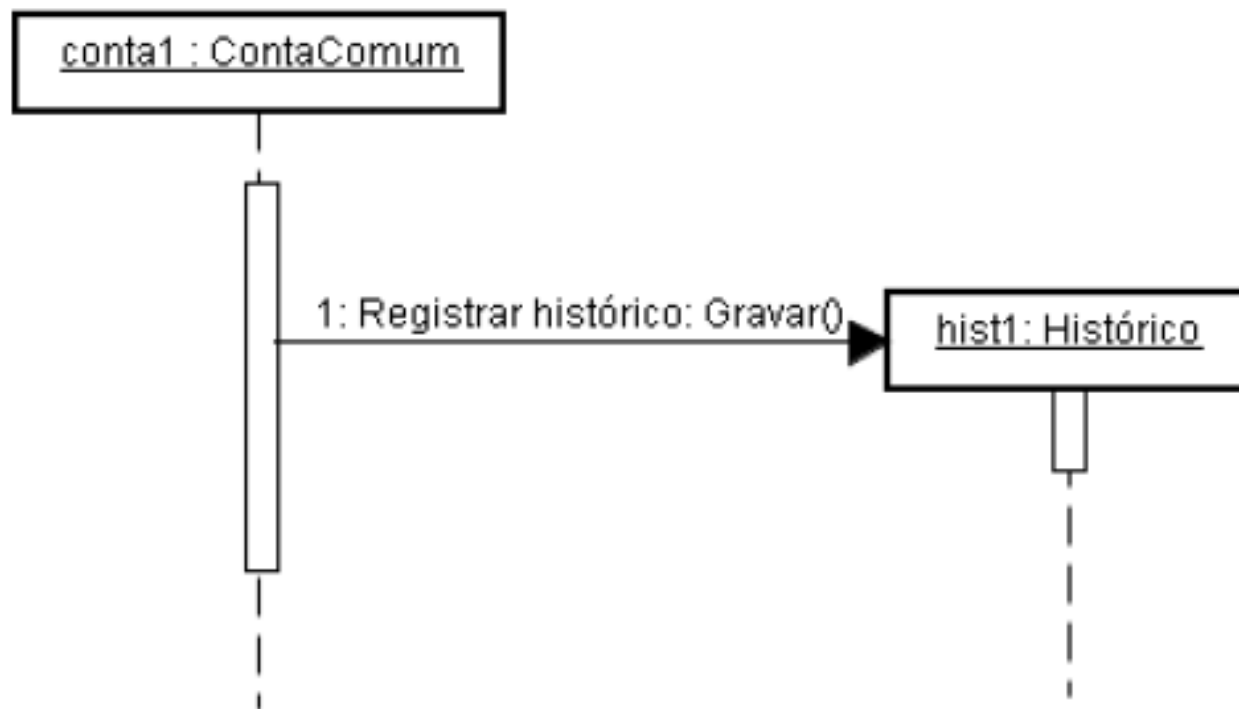
Diagrama de sequência



Instanciação de um objeto

- ❑ A seta atinge o retângulo que representa o objeto
 - O objeto passa a existir a partir daquele momento
- ❑ A mensagem representa a chamada do método construtor

Exemplo de instanciação

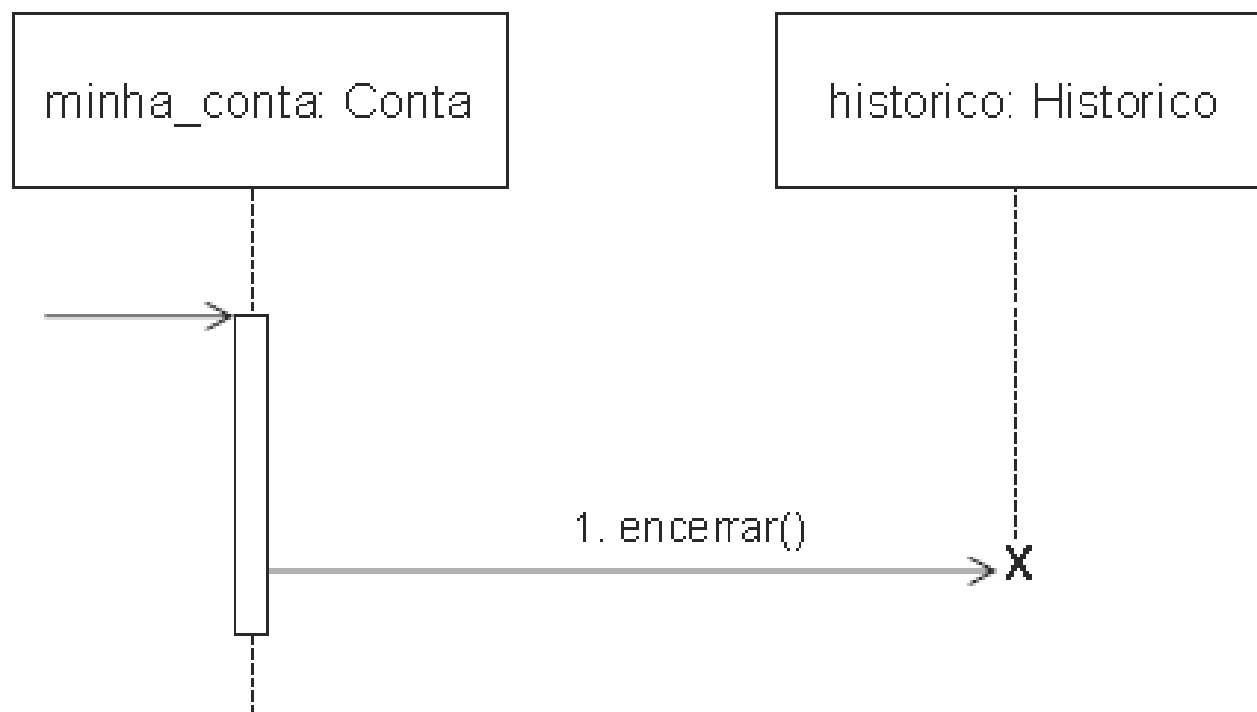


Destrução de um objeto

- ❑ A seta atinge o fim da linha da vida de um objeto
 - Um X marca a destruição do objeto
 - O objeto deixa de existir a partir daquele momento

- ❑ A mensagem representa a chamada do método destrutor

Destruição de um objeto



Auto-chamadas

- ❑ Mensagens que um objeto envia para si mesmo
 - A mensagem parte do objeto e atinge o próprio objeto

- ❑ Utilizado para indicar que o objeto precisa executar algumas operações relacionadas ao serviço solicitado

Exemplo de auto-chamadas

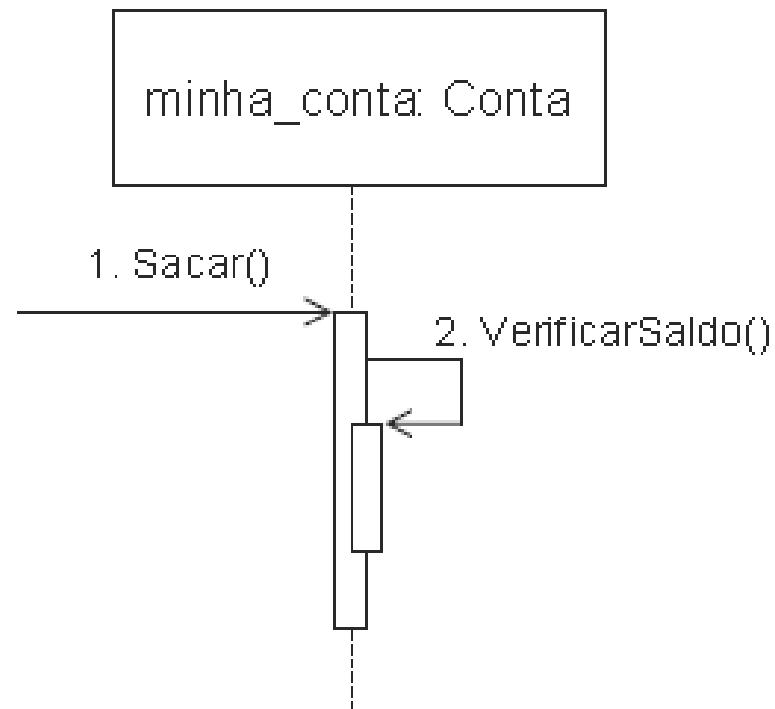
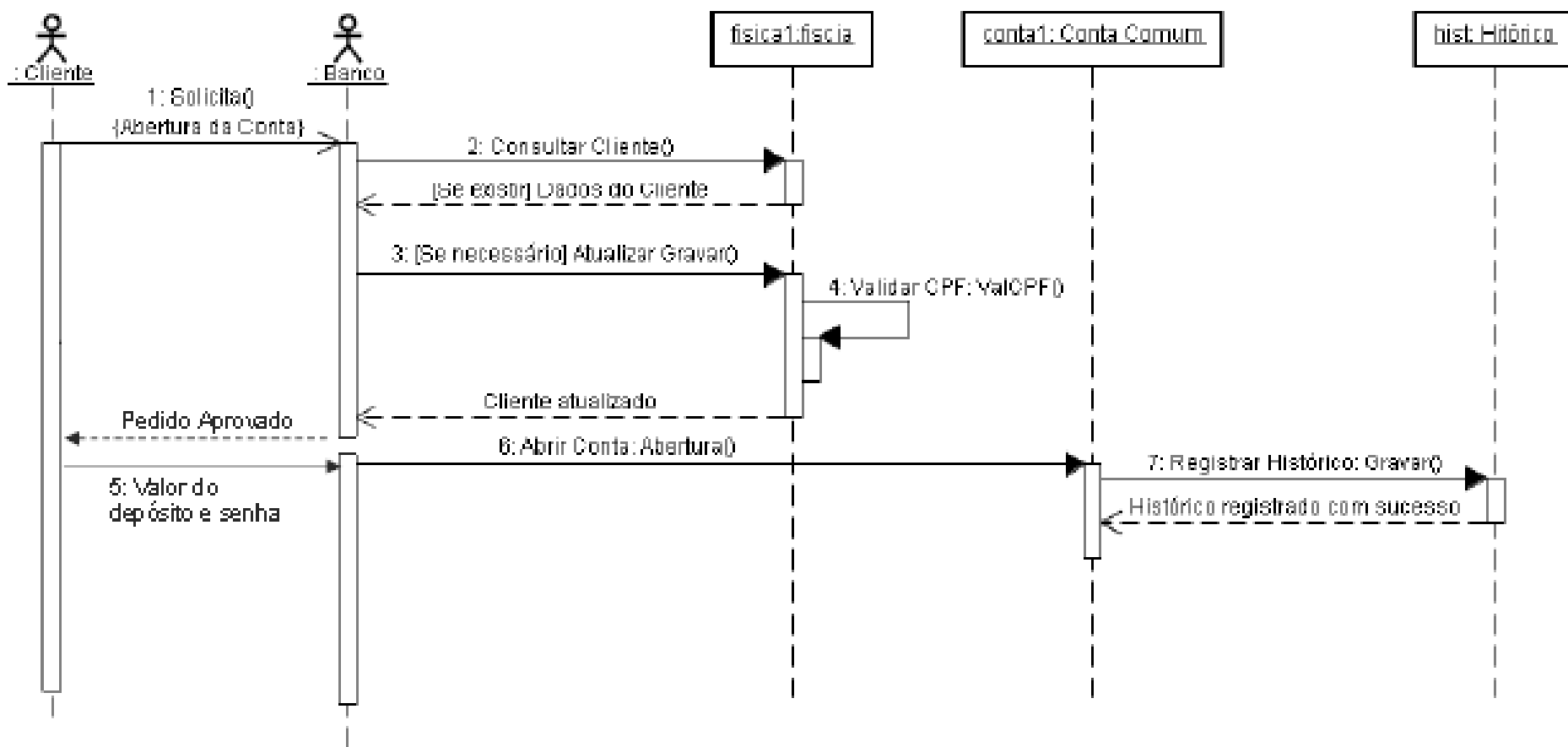


Diagrama de sequência



Detalhando caso de uso com diagrama de sequência

- ❑ Caso de uso é um processo disparado pelo o usuário

- ❑ O diagrama de sequência pode detalhar um caso de uso e mostrar
 - a ordem em que os eventos acontecem
 - as mensagens que são enviadas
 - os métodos que são chamados
 - como os objetos interagem entre si

Caso de uso X sequência

- ❑ Um diagrama de casos de uso pode gerar vários diagramas de sequência
- ❑ Nem sempre um caso de uso gera um diagrama de sequência
- ❑ Diagramas de sequência são comuns quando há relacionamentos do tipo `<<include>>` ou `<<extend>>`

Exemplo: casos de uso

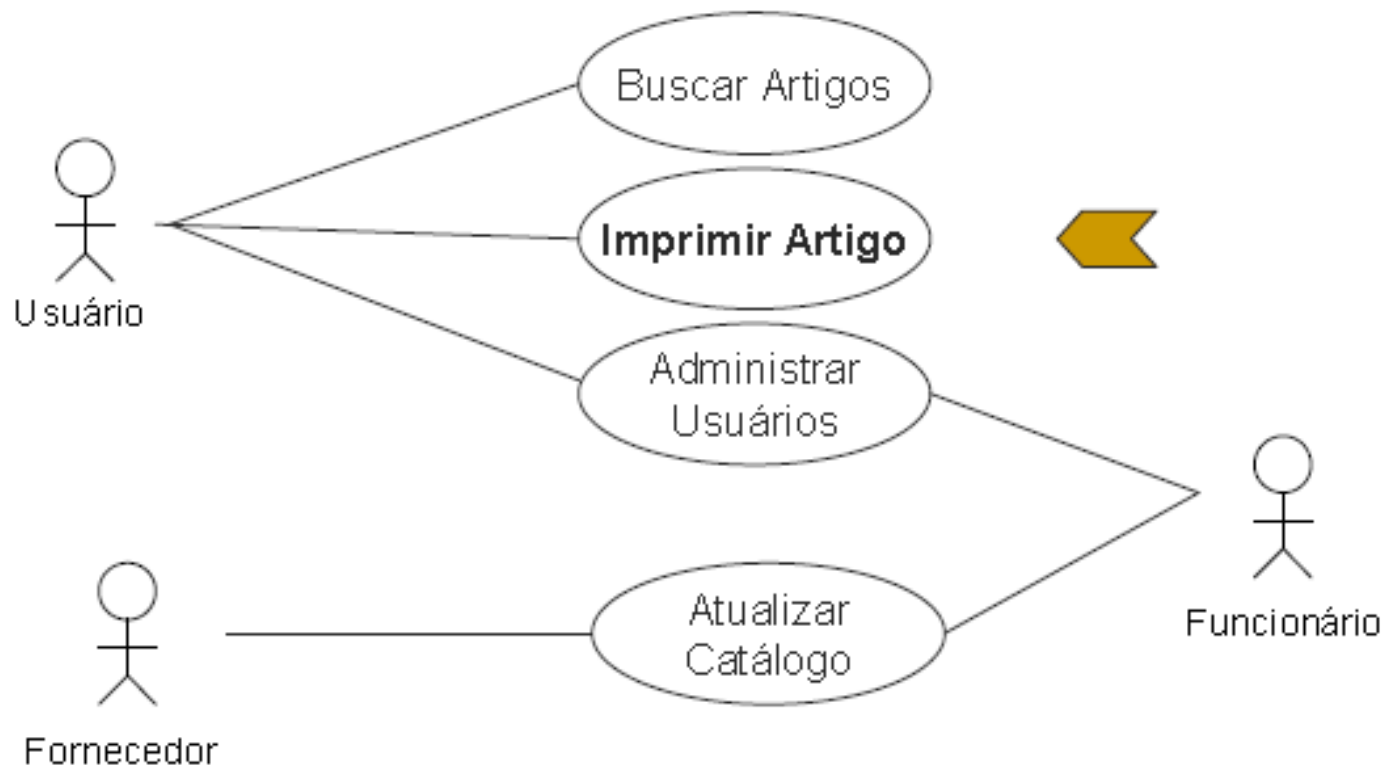
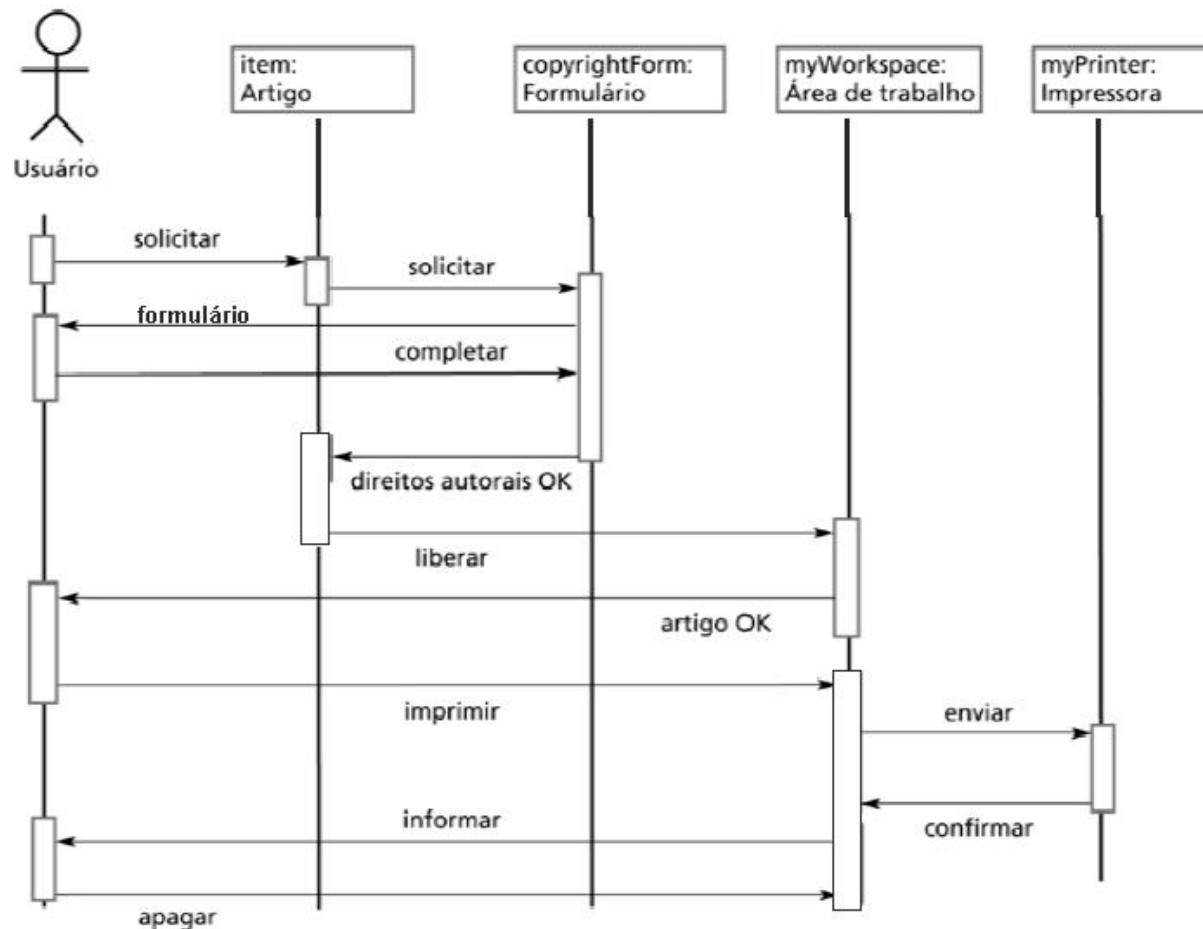
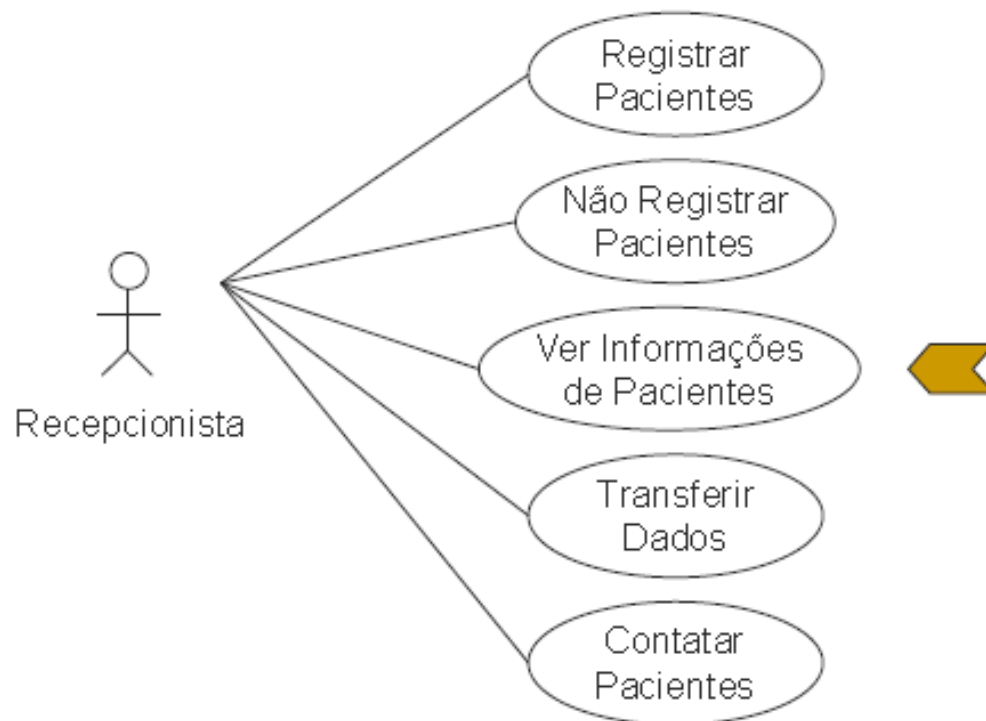


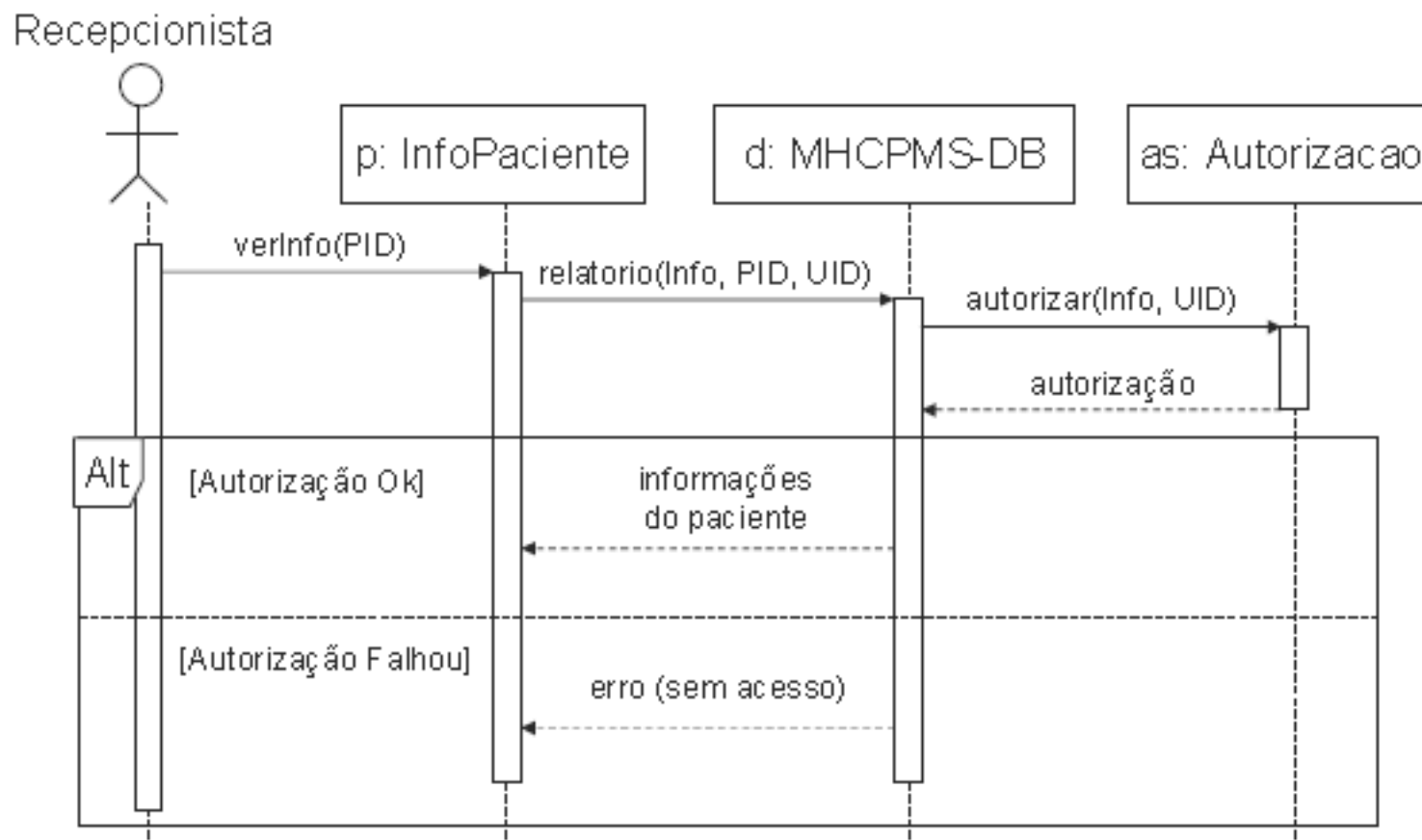
Diagrama de sequência: Imprimir Arquivo



Exemplo 2: casos de uso

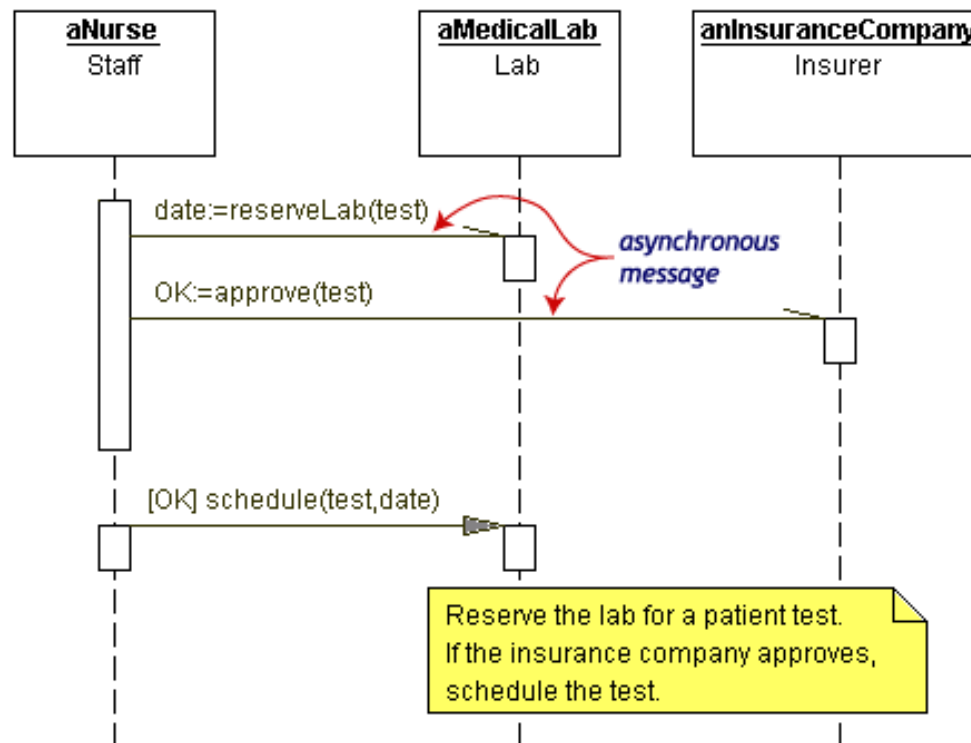


Ver informação de paciente



Processos concorrentes

- ❑ É possível representar mensagens concorrentes assíncronas (mensagens que são processadas em paralelo sem um tempo definido para a sua realização)



Exercícios

9. Construa o diagrama de sequência para o caso de uso “Reservar Carro” descrito no exercício 4.
10. Construa o diagrama de sequência para o caso de uso “Cadastrar Livro” descrito no exercício 5.
11. Desenhe um diagrama de sequência que mostre as interações dos objetos em um sistema de agenda de grupo quando um grupo de pessoas está organizando uma reunião.