Introdução às Linguagens de Modelagem de Software

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

ROTA2030
FUNDEP

Agenda

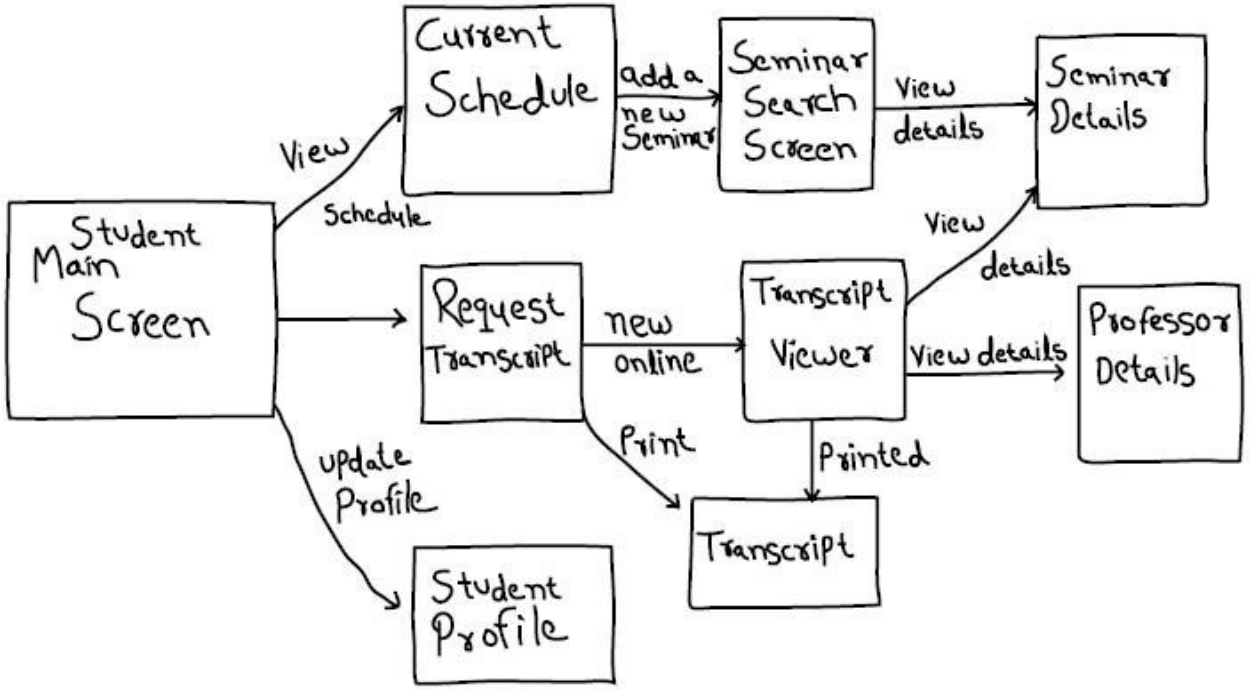
- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Introdução a Modelagem

1. Modelos x documentos: são ortogonais. Modelos podem não ser documentos e existem documentos que não são modelos. Um sketch no verso de um papel toalha é um modelo, embora seja um documento questionável



Introdução a Modelagem

2. Muitos desenvolvedores não gostam de modelagem

- Problema é devido que muitos desenvolvedores quando escutam o termo modelagem pensam em documento
- Devido a má experiência, pensam que modelagem não é funcional
- Frequentemente quando um desenvolvedor é exposto a modelagem é na fase de documentação, depois do desenvolvimento
- O oposto também é verdade: grandes modelos no início, antes de qualquer código

Introdução a Modelagem

3. Modelagem é atrativo para pessoas “visual thinkers”

- Pessoas que gostam de pensar visualmente tem facilidade para trabalhar com modelos
- Existem os extremos, mas geralmente as pessoas se encaixam entre eles
- O ponto é que modelagem vai ser mais efetiva para algumas pessoas do que para outras
- Você precisa entender suas preferências cognitivas e dos membros da sua equipe

Introdução a Modelagem

4. Você precisa pensar antes de agir

- Modelar é pensar em alguma coisa antes de codificar
- A produtividade aumenta quando se pensa no problema antes, desenhando diagramas ou blocos do sistema e entendendo o problema com maior profundidade
- Desenvolvedores produtivos modelam o problema antes de codificá-lo
- Modelagem é uma boa ferramenta para comunicação entre os membros da equipe, tornando o problema mais bem entendido pela equipe

Introdução a Modelagem

5. Conhecimento de domínio é importante

- Se você não entende o domínio do problema, existem poucas chances de ser efetivo
- Todo desenvolvedor deve ser um pouco generalista

6. Cada tipo de modelo tem suas qualidades e defeitos

- Não existe um modelo perfeito para todas as situações
- Você precisa de diversos modelos

Introdução a Modelagem

7. Modelagem é difícil

- Aprender a modelar só vem com a experiência e quando o desenvolvedor quer aprender
- Em um ambiente de desenvolvimento ágil, cada membro da equipe aprende com os outros e a combinação deles torna o time mais forte

- Unified Modeling Language (UML)
  - Linguagem de modelagem de propósito geral
  - Padroniza a especificação, visualização, construção e documentação de sistemas de software
  - Foi criada pelo Object Management Group (OMG)
  - Primeira especificação da UML 1.0 foi proposta em 1997

UML

- UML não é uma linguagem de programação
- UML é uma linguagem de visualização baseada em diagramas padronizados
- Diagramas são usados para retratar a estrutura e o comportamento do sistema
- Versão atual é a 2.5.1
  - Dezembro de 2017
    - 2.4.1 Julho 2011, 2.3 Maio 2010, 2.2 Janeiro 2009, 2.0 Julho 2005
  - Veremos mais sobre as diferenças entre a 1.0 e 2.x na sequência do curso

SysML

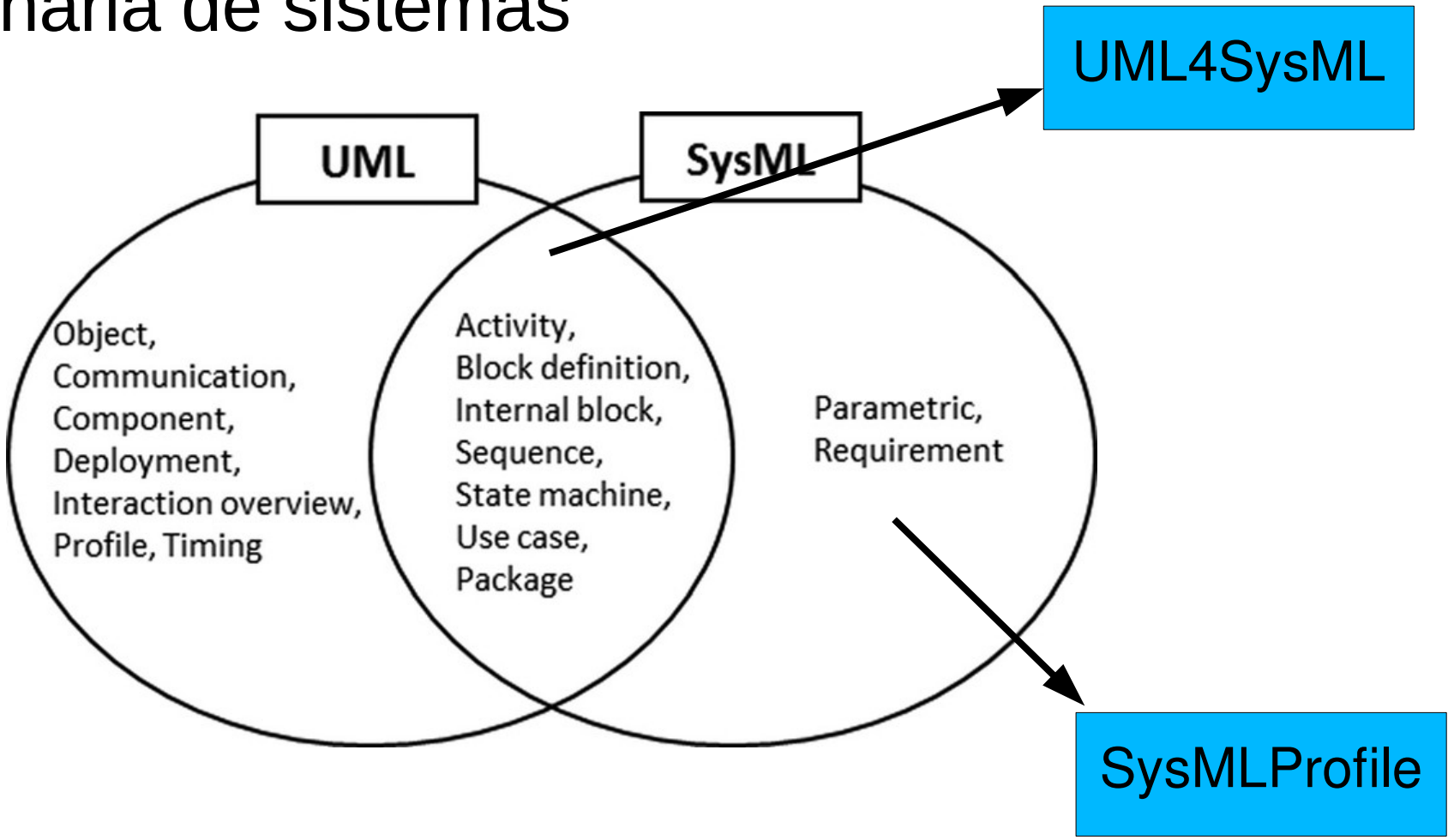
- A UML 2 provê várias construções interessantes para engenharia de sistemas
  - Decomposição estrutural e interconexão via *Parts*, *Ports*, *Connectors*
  - Decomposição comportamental através de diagramas de sequência, atividade e estados
  - Melhoramentos do diagrama de atividades
- Entretanto, o vocabulário usado é relacionado a construção de software
  - Objetos, classes, etc

SysML

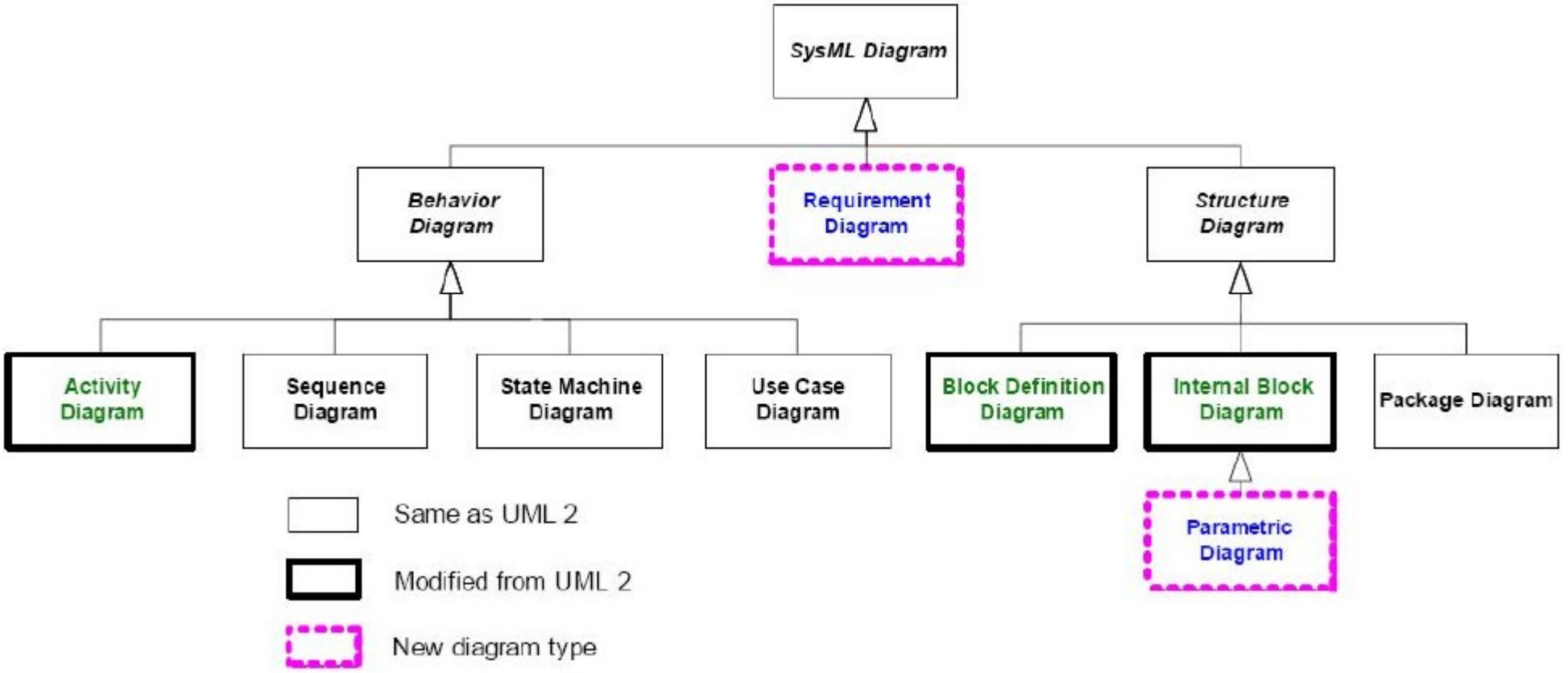
- História breve da SysML
  - UML para Engenharia de Sistemas lançada em 2003 pela OMG
  - Especificação da SysML 1.0 criada em 2006 pela OMG
  - Especificação da SysML 1.1 criada em 2008 pela OMG
  - Especificação atual 1.6 lançada em Dezembro de 2019
    - https://sysml.org/sysml-specs/

SysML

- SysML reusa um subconjunto da UML 2 e adiciona algumas extensões endereçando engenharia de sistemas



SysML - Taxonomia



SysML

- Os dois novos diagramas (requirements e parametric) permitem o gerenciamento de requisitos e análise de desempenho
- Tem menos diagramas que a UML
- Específica para um domínio (sistemas complexos que envolvam hardware)
- Ex: definição de block na SysML reutiliza o diagrama composto (composite) da UML

UML x SysML

- SysML é específica para a modelagem voltada para engenharia de sistemas
- UML é genérica para modelagem de software
  - Entretanto, possui extensões específicas que permitem a sua utilização também para engenharia de sistemas
- Na verdade, a extensão UML-MARTE permite inclusive a modelagem de aspectos temporais

Porquê precisamos de modelagem?

- Aplicações complexas precisam a colaboração e planejamento de equipes multidisciplinares
- Por isso há a necessidade de uma comunicação clara e concisa entre as equipes
- Pessoas da área de negócios/vendas/administração têm dificuldade de entender código
  - UML ajuda os não programadores a entender os requisitos e funcionalidades

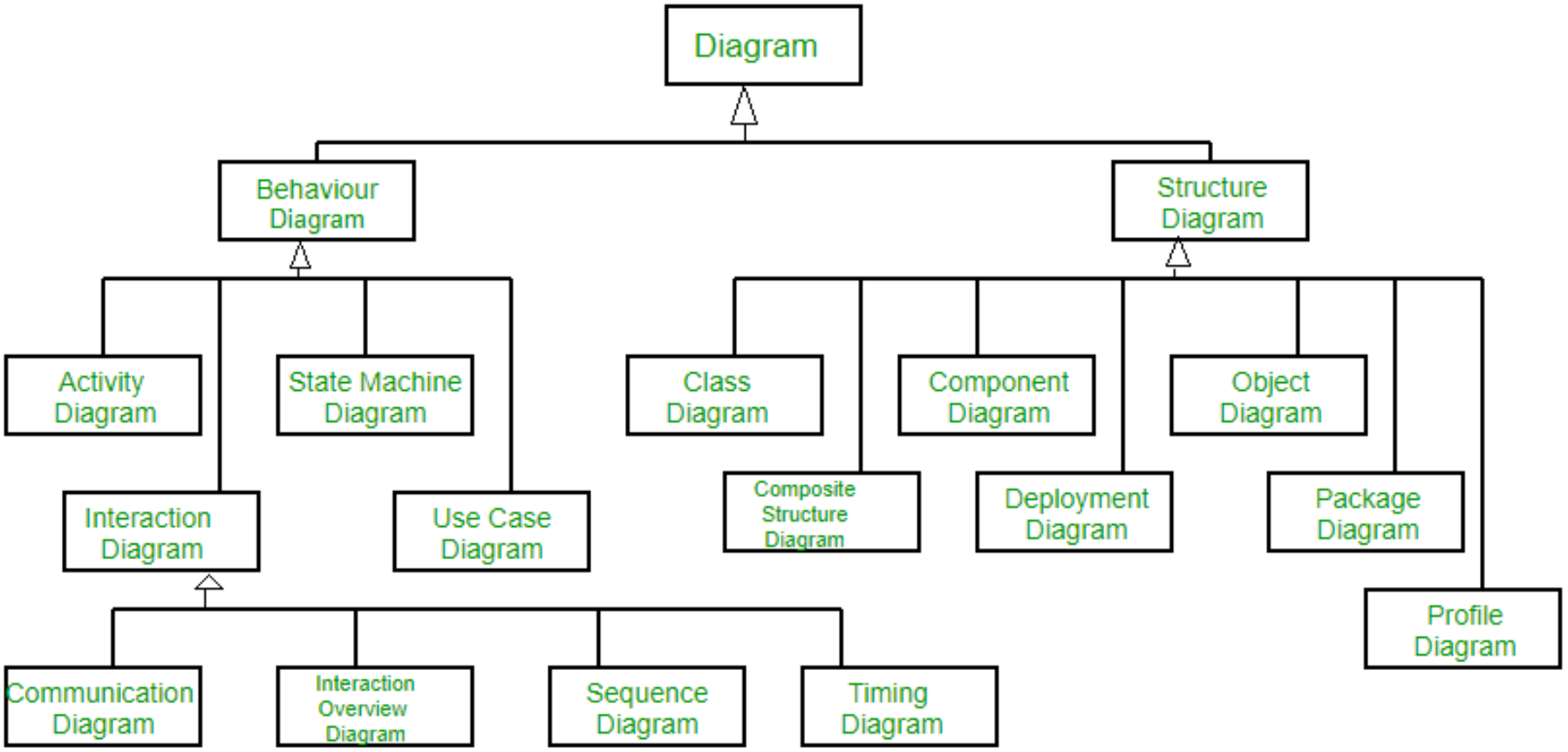
Porquê precisamos de modelagem?

- Menor tempo de desenvolvimento
  - Definição macro do sistema através de notação conhecida e comum a todos
  - Membros da equipe conseguem visualizar o todo, partes e a interação entre as partes
  - Menor tempo gasto em codificação (pois se pensou o problema de antemão)

UML

- UML tem uma forte ligação com análise e projeto orientado a objetos
- Usa elementos e formas associadas entre eles para criar diagramas
- Existem duas classes de diagramas em UML:
  - Diagramas estruturais: capturam aspectos estáticos ou da estrutura do sistema. Diagramas: componente, objeto, classe e deployment
  - Diagramas comportamentais: capturam aspectos dinâmicos ou comportamentais do sistema. Diagramas: caso de uso, estados, atividade e interação

Diagramas da UML 2.2



Conceitos de OO usados pela UML

- Classe: define a estrutura e funções de um objeto
- Objetos: é a unidade fundamental de um sistema que é usado para representar uma entidade. Objetos ajudam a decompor um sistema grande e complexo em módulos. Modularidade ajuda a dividir o sistema em componentes menores de mais fácil entendimento
- Herança: mecanismo pela qual classes filhas herdam as propriedades das classes pais
- Abstração: detalhes de implementação são “escondidos” do usuário

Conceitos de OO usados pela UML

- Encapsulamento: proteger os dados (atributos da classe ou objeto) do mundo exterior
- Polimorfismo: mecanismo que permite que funções ou métodos existam em diferentes formas

UML 2.x

- Foram adicionados à UML 2.x
  - Metodologias de desenvolvimento de software ágil foram incorporadas e a especificação original foi ampliada
  - UML 1.x especificava 9 diagramas. UML 2.x aumentou para 13 diagramas. Os 4 novos diagramas são: timing, communication, interaction overview, e composite structure
  - UML 2.x renomeou o diagrama statechart para diagrama de máquina de estados
  - UML 2.x adicionou a possibilidade de decompor o sistema de software em componentes e sub-componentes

Diagramas Estruturais

- Diagrama de classes
  - Um dos mais usados no desenvolvimento de software orientado a objetos
  - Mostra as classes do sistema e a estrutura estática do mesmo
  - Mostra a relação entre as classes e objetos
- Diagrama de estrutura composta
  - Representa a estrutura interna de uma classe e seus pontos de interação com outras partes do sistema
- Diagrama de objeto
  - Mostra as instâncias das classes do sistema em um determinado ponto
  - Objetos e suas relações (caso especial do diagrama de classes)

Diagramas Estruturais

- Diagrama de Componentes
  - Usados para modelar detalhes de implementação
  - Representam a estrutura relacional entre os elementos de software e ajudam a entender se os requisitos funcionais foram atendidos
- Diagrama de Deployment
  - Usados para representar o hardware e software
  - Mostra quais os componentes de hardware existem e quais os componentes de software que executam sobre o hardware
- Diagrama de Pacotes
  - Mostra a organização dos pacotes e suas dependências
  - Organiza os diagramas em grupos (pacotes)

Diagramas Comportamentais

- Diagrama de Máquina de estados
  - Representa o comportamento do sistema através de estados finitos e suas transições
  - Modela comportamento dinâmico do sistema em resposta ao tempo e eventos
- Diagrama de Atividades
  - Ilustra o fluxo de controle do sistema
  - Mostra os passos para a execução de um caso de uso
- Diagrama de Casos de Uso
  - Ilustra as funcionalidades do sistema ou parte do sistema
  - Demonstra os requisitos funcionais
  - Visão de alto nível do que o sistema faz

Diagramas Comportamentais

- Diagrama de sequência
  - Mostra a interação entre objetos em uma ordem sequencial
  - Representação próxima ao código
- Diagrama de comunicação
  - Usado para demonstrar a troca de mensagens entre objetos
  - Similar ao diagrama de sequência, mas tem uma forma mais livre de representação

Diagramas Comportamentais

- Diagrama de tempo
  - É uma forma especial do diagrama de sequência
  - Usado para demonstrar o comportamento dos objetos em um período de tempo

- Diagrama de visão geral de interação
  - Modela uma sequência de ações
  - Ajuda a simplificar interações complexas
  - É uma mistura entre os diagramas de atividade e sequência

Resumo

- Ao todo a UML 2.x oferece suporte para 13 diagramas
- Alguns deles têm objetivos similares
  - Ex: diagramas de sequência e comunicação
  - Ex: visão geral de interação e atividades
- Utilizar todos eles em um projeto não é necessário
- Iremos focar em um subconjunto deles

Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Diagramas de Casos de Uso

- A UML e os diagramas de casos de uso não interferem no levantamento de requisitos
- Casos de uso auxiliam o desenvolvedor para validar os requisitos extraídos do usuário
- Suponha o seguinte diálogo entre usuário e analista

Desenvolvedor:

O que o sr. espera desse sistema?

Usuário:

Eu tenho uma loja de peças. Gostaria que o meu PV fosse interligado com meu estoque e eu pudesse a qualquer momento alterar valores dos FPs. Posso oferecer descontos a alguns tipos de clientes, mas preciso autorizar essa operação.

No fim do mês quero um relatório dos produtos que mais venderam. Preciso também saber a estatística de vendas por forma de pagamento.

De tempos em tempos deve aparecer na tela do sistema uma promoção relâmpago que dê um brinde ao cliente.

Preciso que o sistema controle os pedidos também.

Diagramas de Casos de Uso

- O que você identificaria como ambíguo ou não compreensível nessa descrição do usuário?
- Cabe ao analista/desenvolvedor buscar o esclarecimento desses itens
  - O que é PV e PF?
  - Que tipos de clientes podem receber descontos?
  - Como seria feita esta autorização e por quem?
  - Que quantidade de produtos deve aparecer no relatório dos que mais venderam?
  - A estatística leva em conta que período?
  - Quanto tempo significa “de tempos em tempos”?
  - Quais pedidos precisam ser controlados (clientes ou fornecedores)?

Diagramas de Casos de Uso

- A modelagem de casos de uso ajuda a unir o usuário e desenvolvedor
- Expressa os requisitos de uma forma que não sejam para o desenvolvedor um texto extenso e pouco estruturado
- E para o usuário não sejam uma sequência de comando ou código sem o menor sentido
- Um caso de uso descreve uma sequência de ações que representam um cenário principal e alternativos, com o objetivo de demonstrar o comportamento de um sistema (ou parte dele)

Diagramas de Casos de Uso

- O primeiro passo é separar as funcionalidades do sistema
- Destas funcionalidades, devemos agrupar um conjunto de ações que tenham um objetivo bem definido
- Ex: sistema de vendas de uma loja de roupas
  - Consultar informações sobre um produto, efetuar reserva, emitir comprovante de reserva, efetuar venda, emitir NF, realizar fechamento do caixa diário, etc
  - Cada uma das tarefas possui um conjunto de ações
  - Venda: informar identificação do(a) vendedor(a), identificação do produto, quantidade, etc
  - Estamos pensando em um cenário principal

Diagramas de Casos de Uso

- O cenário principal descreve um conjunto de ações que serão executadas considerando que nada de errado ocorrerá durante a execução da sequência
- Ex: Emitir saldo em um terminal de caixa eletrônico

Cenário Principal
1. O sistema realiza a leitura do cartão magnético do correntista. 2. O correntista informa a senha. 3. O sistema valida a senha. 4. O correntista seleciona a opção de saldo. 5. O correntista informa o tipo de saldo: conta corrente, poupança, aplicações. 6. O sistema processa e mostra o saldo solicitado pelo cliente. etc.

Diagramas de Casos de Uso

- O que pode ocorrer de errado no exemplo anterior?
- Como representar as exceções?
  - Como cenários alternativos (subitens do cenário principal)

Alternativa: Problemas na leitura do cartão magnético

1a) Se o sistema não conseguir ler os dados do cartão magnético, tentar nova leitura por, no máximo, mais duas vezes. Caso persista o problema, encerrar o caso de uso[17].

Alternativa: Senha Inválida

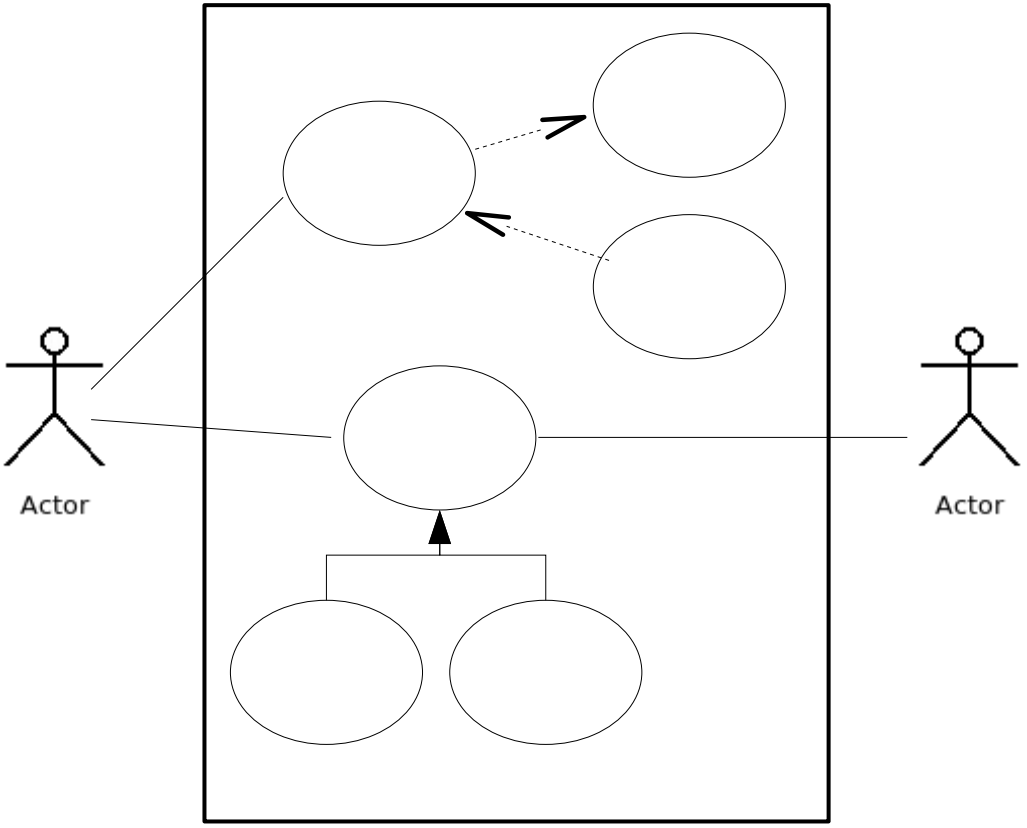
3a) Se a senha digitada não for igual à senha cadastrada no sistema, informar ao correntista e solicitar nova digitação. Esse processo pode ser repetido por no máximo três tentativas (incluindo a primeira). Após a terceira tentativa, a conta do usuário deverá ser bloqueada e o caso de uso encerrado. Extend Bloquear Conta[18].

Alternativa: Conta Inexistente

6a) Se o correntista não possuir o tipo de conta selecionada, informar a ele e encerrar o caso de uso.

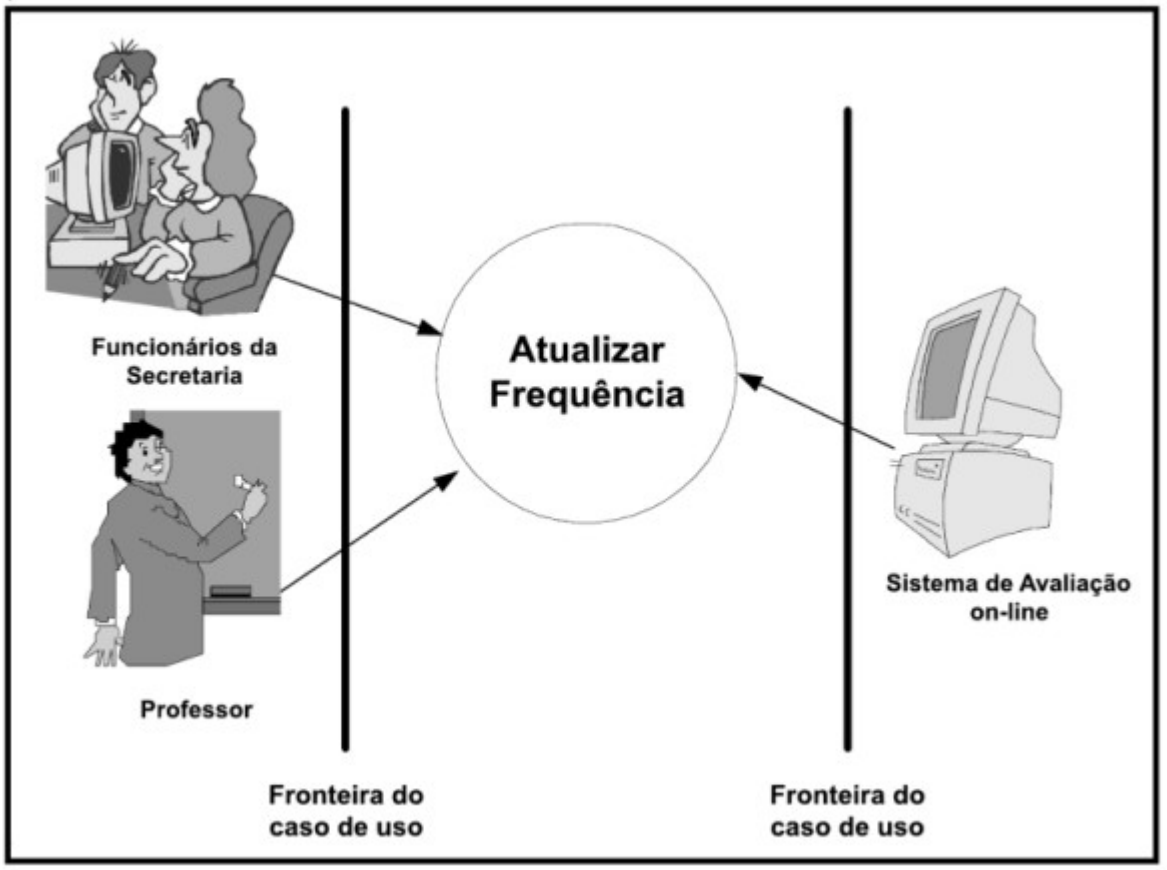
Diagramas de Casos de Uso

- A partir da descrição dos cenários principais e alternativos, podemos ir para o diagrama de caso de uso



Diagramas de Casos de Uso

- Ex: atualizar frequência

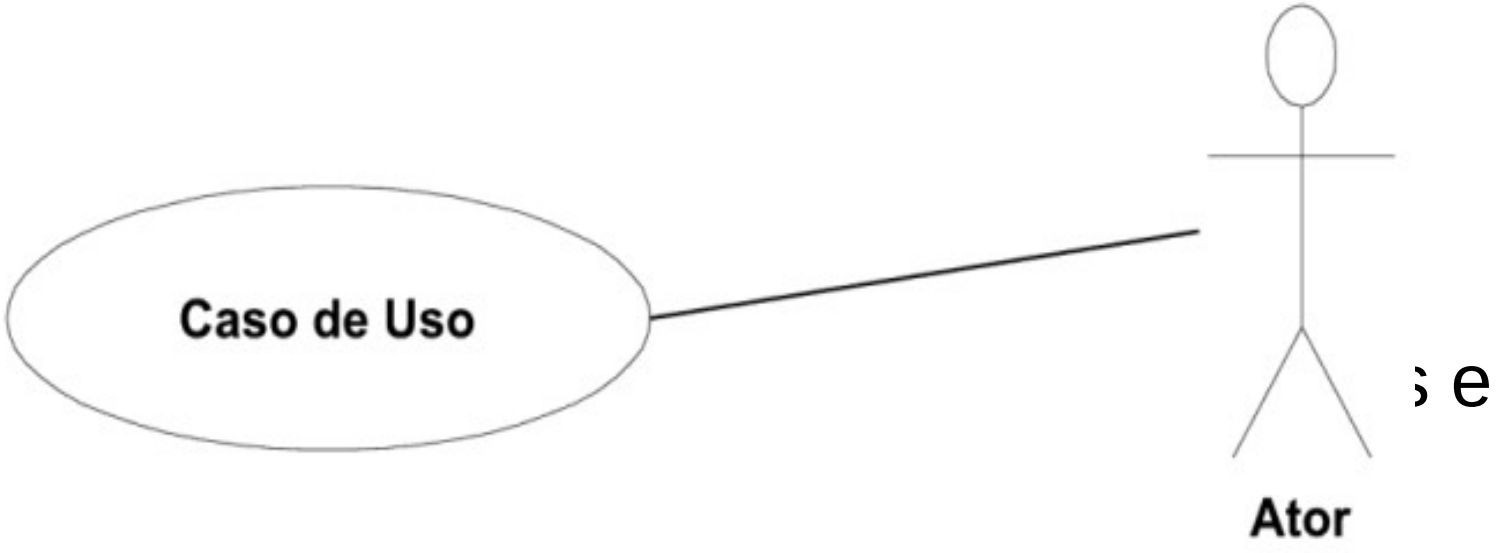


Relacionamento entre Casos de Uso e Atores

- Os casos de uso se relacionam entre si e entre os atores
- Relacionamentos de casos de uso: generalização, extensão e inclusão
- Relacionamentos de atores: generalização
- Relacionamentos entre atores e casos de uso: associação

Relacionamento entre Casos de Uso e Atores

- Associação
  - Representa a interação do ator com o caso de uso por meio do envio e recebimento de mensagens
  - As
  - É o
  - Ex: ator Correntista envia e recebe mensagens do caso de uso Calcular Empréstimo Pessoal

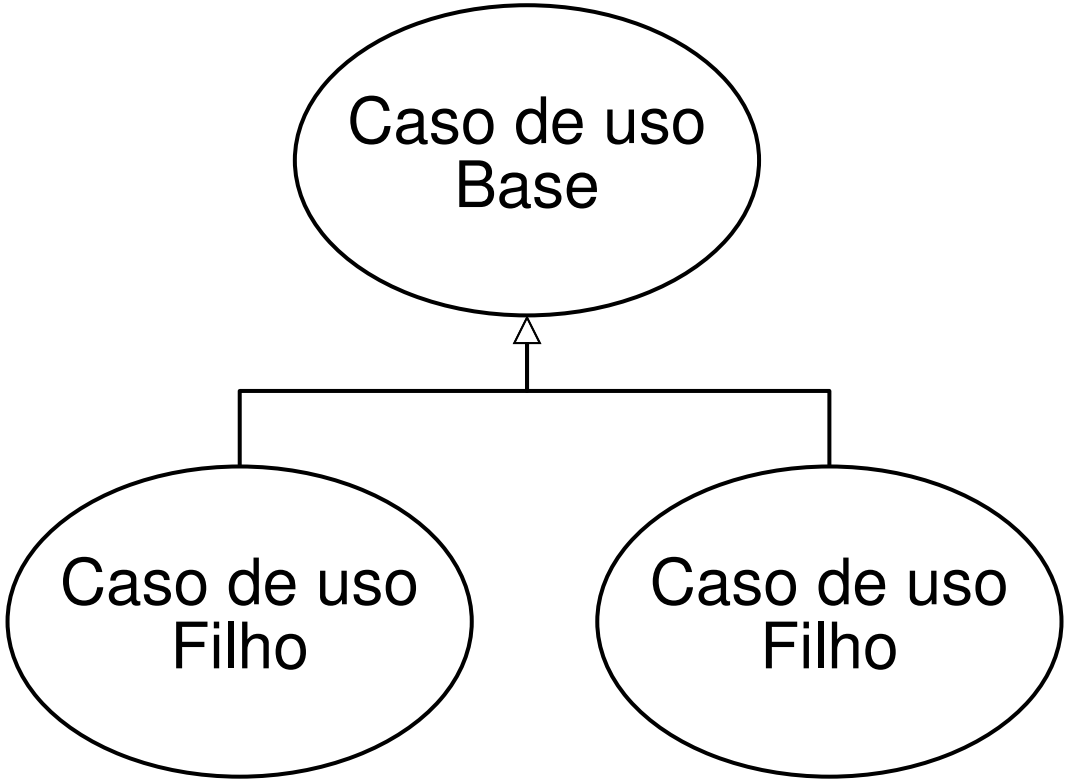


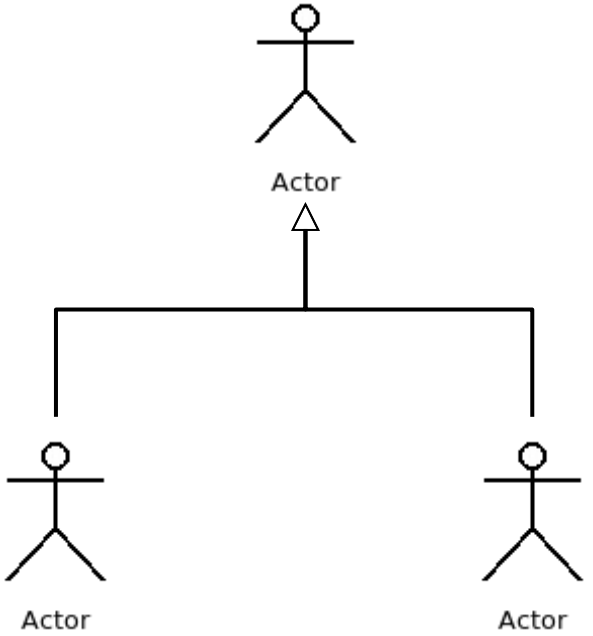
Relacionamento entre Casos de Uso e Atores

- Generalização
  - Ocorre entre casos de uso ou entre atores
  - É considerado quando existem dois elementos semelhantes, mas com um deles realizando algo a mais
  - É similar ao conceito de orientação a objeto, quando uma classe herda de outra
  - Ex: ator genérico Aluno é especializado nos atores Aluno Matriculado e Aluno Ouvinte

Relacionamento entre Casos de Uso e Atores

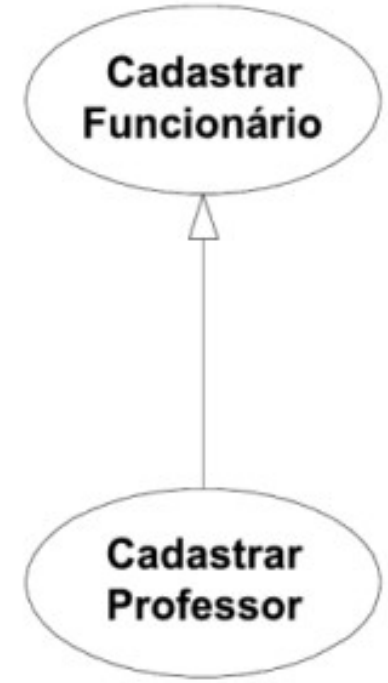
- Generalização



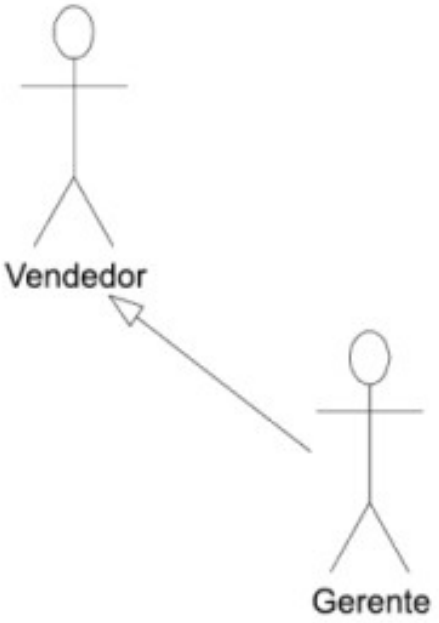


Relacionamento entre Casos de Uso e Atores

- Generalização



Generalização entre casos de uso



Generalização entre atores

Relacionamento entre Casos de Uso e Atores

- Extensão (extend)
  - Ocorre entre casos de uso e indica que um deles terá seu procedimento acrescido, em um ponto de extensão, de outro caso de uso, identificado como base
  - O mesmo caso de uso de extensão pode estender mais de um caso de uso
  - Os pontos de extensão são rótulos que aparecem nos cenários do caso de uso base, exemplo:

Cenário Principal

...

5. O usuário escolhe a forma de pagamento.

6. Se cliente VIP, o sistema calcula o desconto especial. **Extend** (desconto ClienteVip).

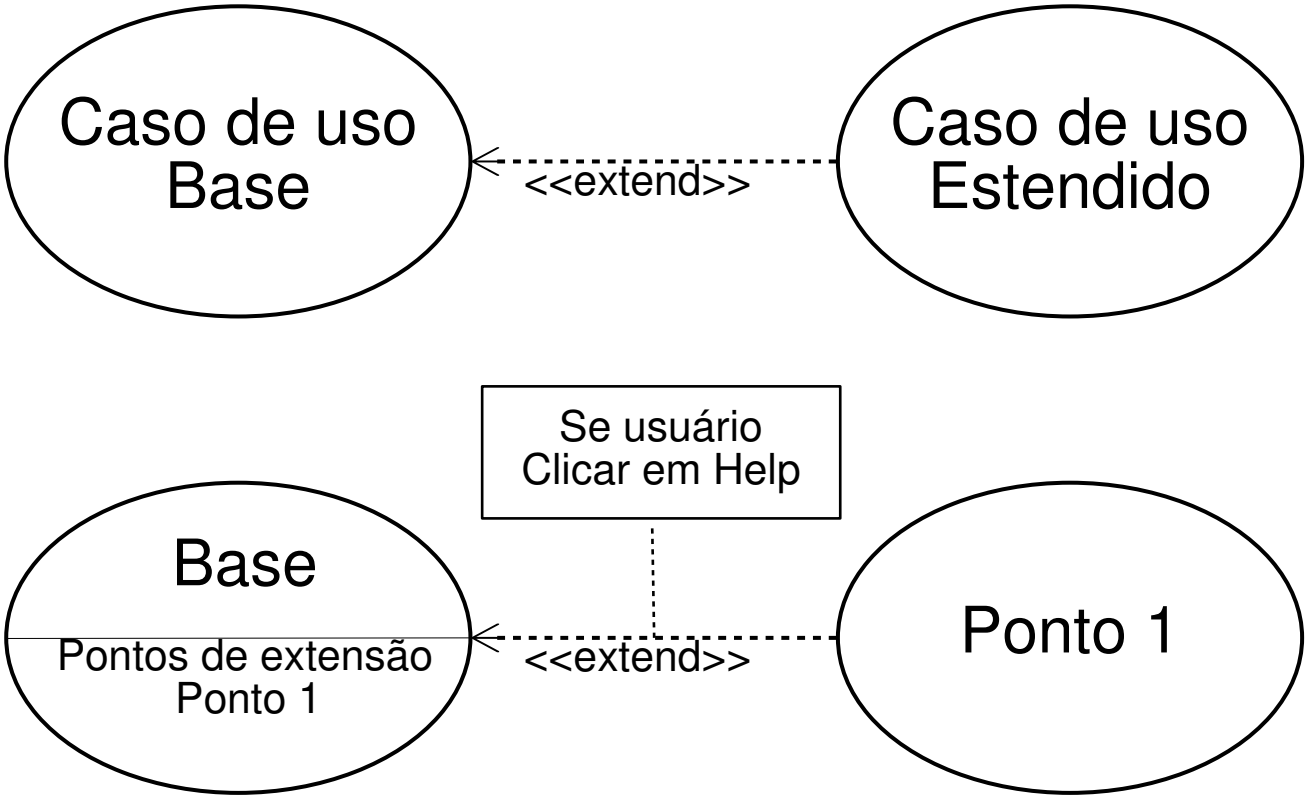
...

Relacionamento entre Casos de Uso e Atores

- Um caso de uso de extensão é muito utilizado para:
  - Expressar rotinas de exceção ou desmembramento de um caso de uso (cenário alternativo com fluxo grande ou que mereça uma atenção especial)
  - Separar um comportamento obrigatório de outro opcional
  - Separar um trecho do caso de uso que será executado apenas em determinadas condições
  - Separar trechos que dependam da interação com um determinado ator (ex: rotina de desconto só pode ser executada pelo gerente)

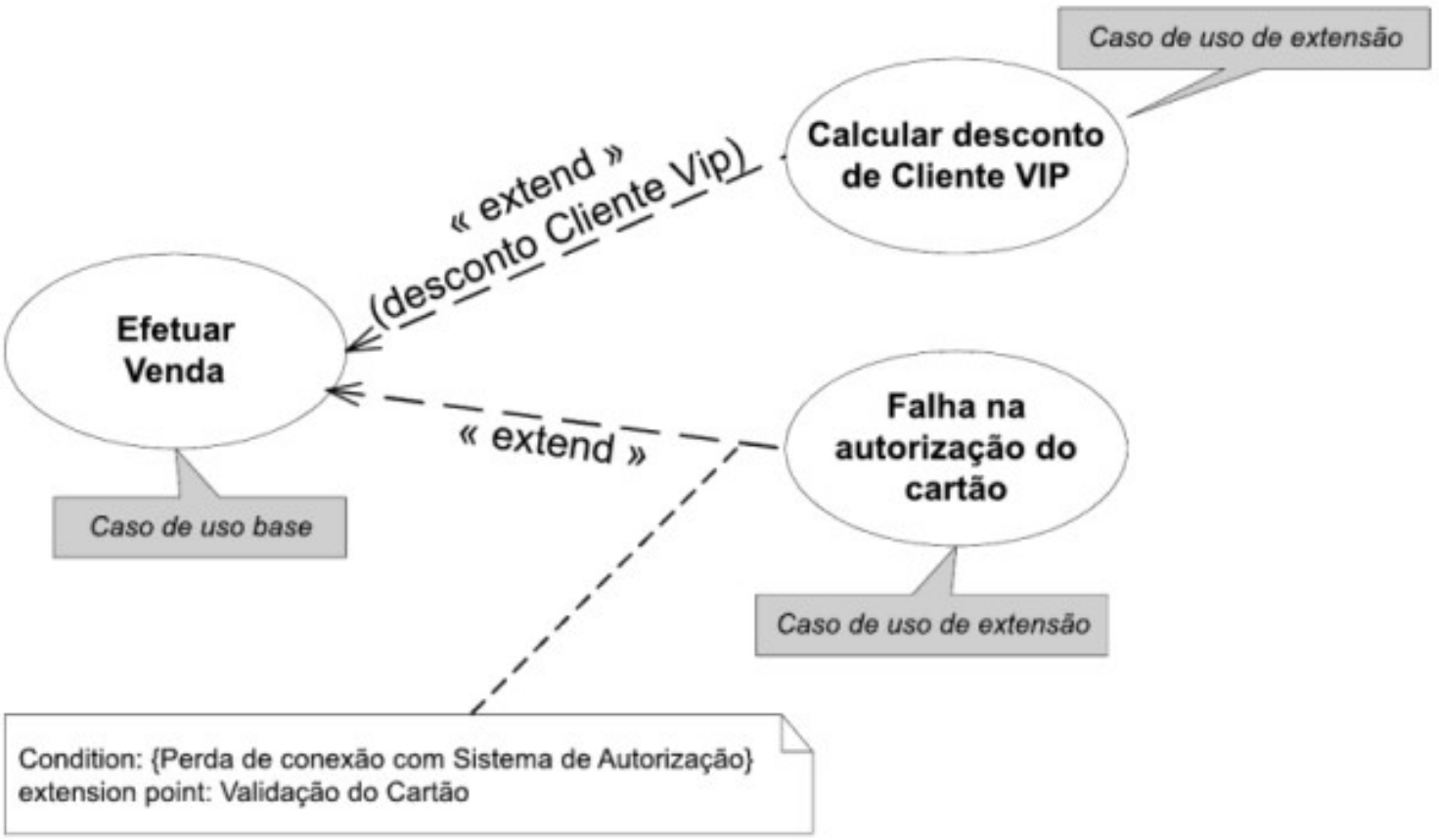
Relacionamento entre Casos de Uso e Atores

- Extensão (extend)



Relacionamento entre Casos de Uso e Atores

- Extensão (extend)



Relacionamento entre Casos de Uso e Atores

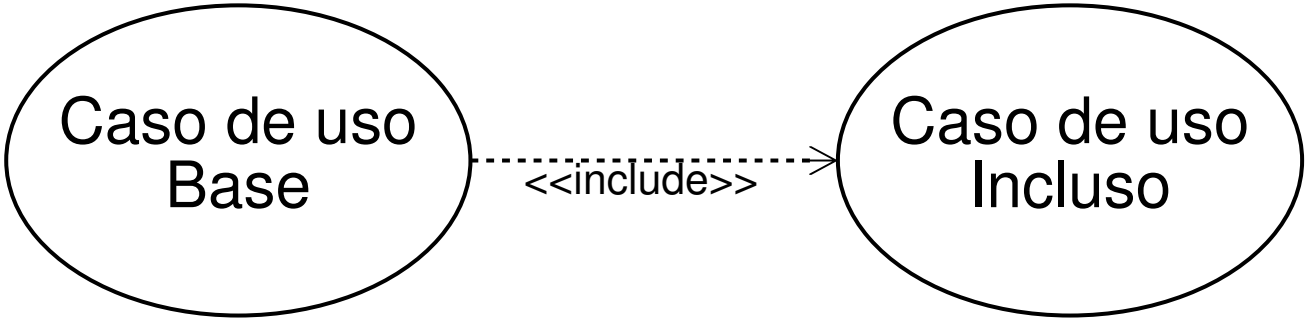
- Inclusão (include)
  - Entre dois casos de uso e significa que o comportamento definido em um caso de uso de inclusão é incluído no comportamento de um caso de uso base
  - O relacionamento de inclusão tem a intenção de ser usado quando há partes comuns de um comportamento a dois ou mais casos de uso
  - Semelhante à chamada de uma função

Cenário Principal

1. O aluno informa sua matrícula. O sistema verifica se a matrícula é válida e ativa. **Include (Validar Matrícula).** (...)

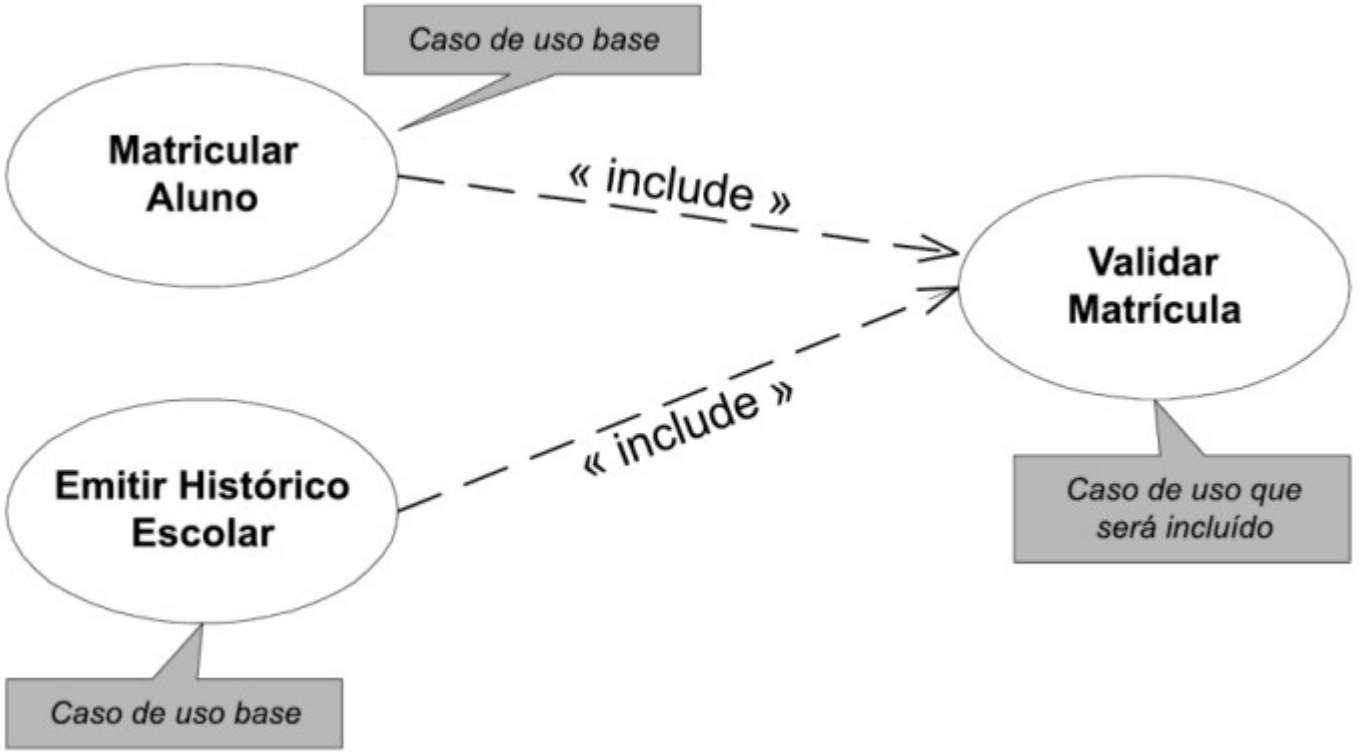
Relacionamento entre Casos de Uso e Atores

- Inclusão (include)



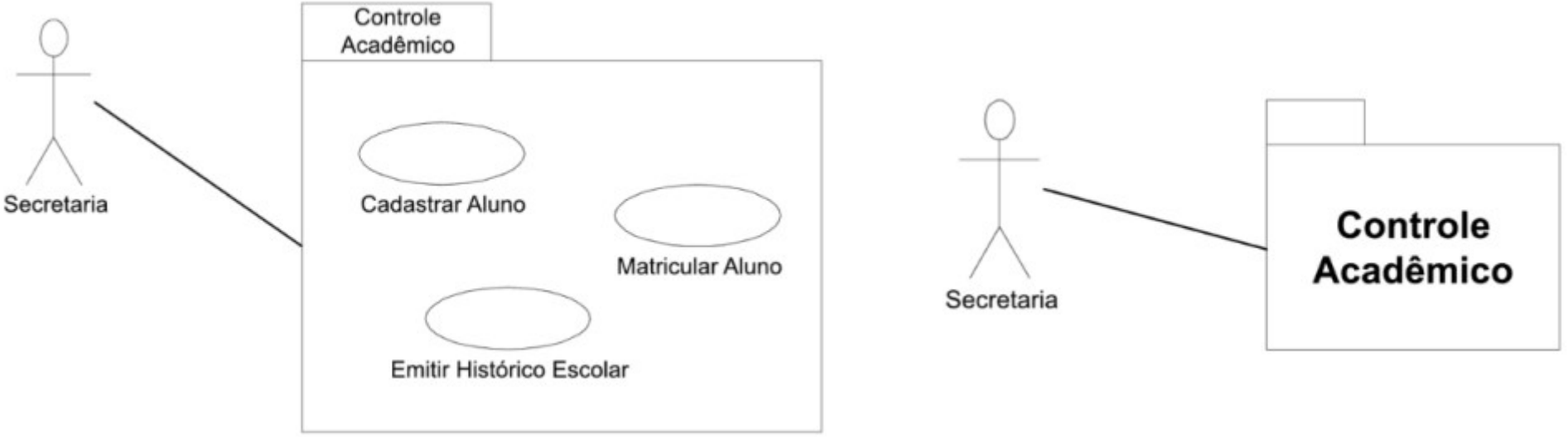
Relacionamento entre Casos de Uso e Atores

- Inclusão (include)



Casos de Uso com Auxílio de Pacotes

- Quando o sistema é muito complexo, é comum existir dezenas de casos de uso
- Neste caso, a representação gráfica é difícil
- Pacotes auxiliam na visualização

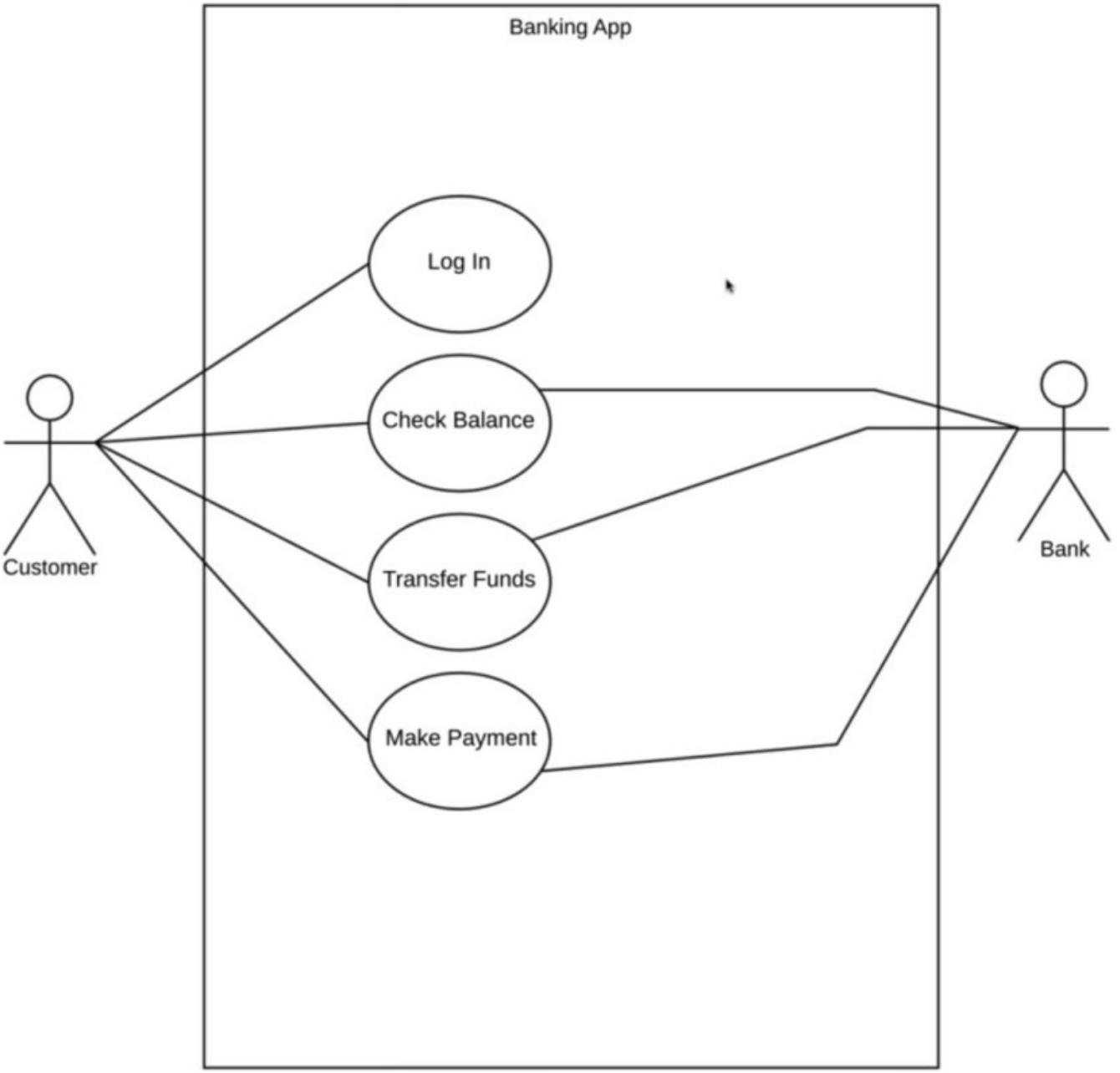


Diagramas de Casos de Uso

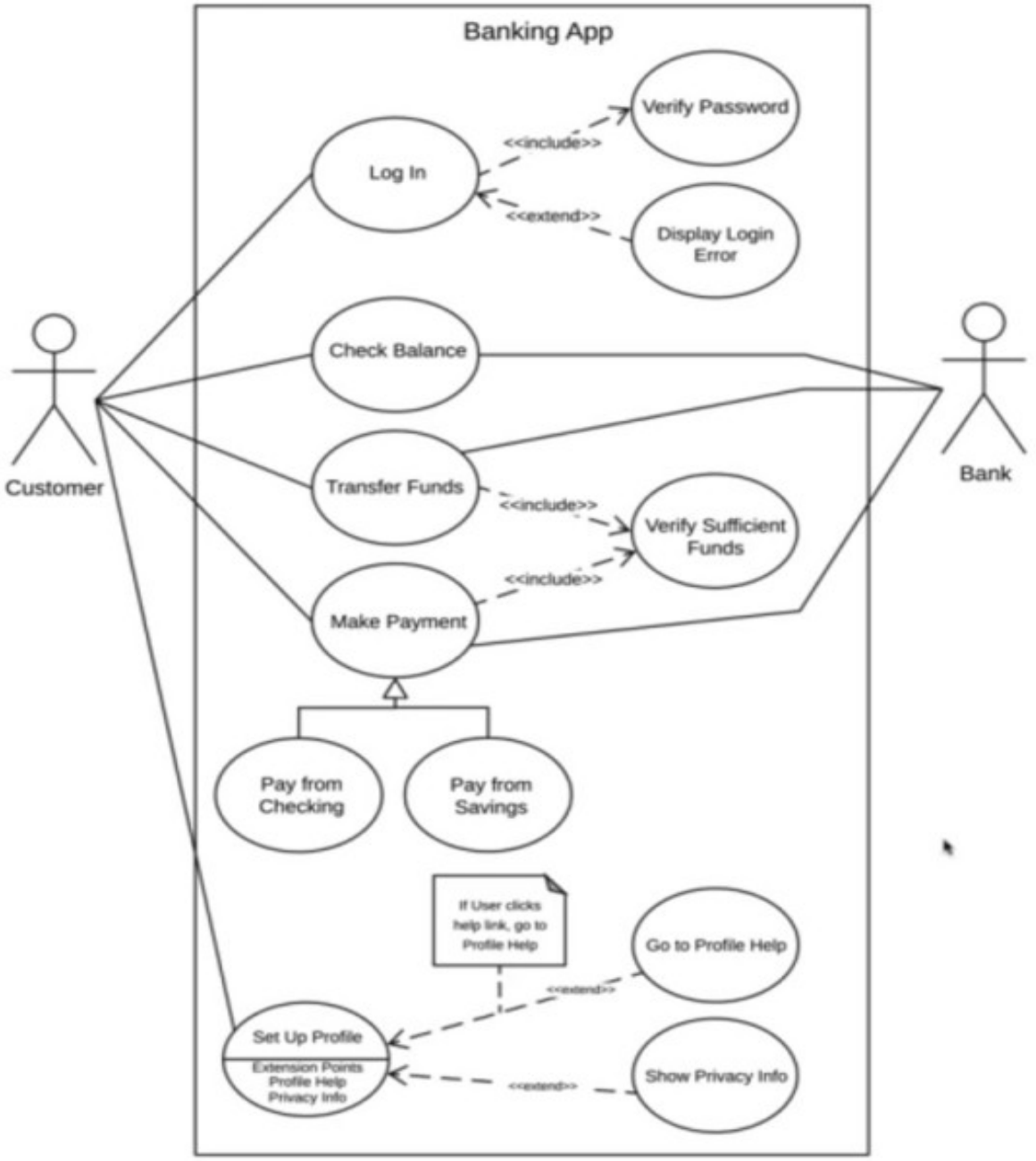
- Exercício
- Modelar os casos de uso de um app de banco
- O app deve prover as seguintes funcionalidades:
  - Logar no app
  - Verificar o saldo
  - Transferência
  - Fazer pagamento

15 minutos

Diagramas de Casos de Uso - Exercício



Diagramas de Casos de Uso - Exercício



Diagramas de Casos de Uso

- Não existe na documentação da UML uma forma pré-definida de se escrever um caso de uso
- Boas práticas:
  - Caso de uso não deve incluir detalhes de como será implementado, exceto se for um caso de uso de projeto (ex: manter cópia de backup)
  - Ex: Detalhes como usuário clicará em um botão ou selecionará de uma listbox. Diga apenas que usuário selecionará uma opção
  - Após escrever uma primeira versão, analise-o novamente. Verifique se alguma parte é reaproveitável por outro caso de uso
  - Um caso de uso pode se transformar em uma função ou método de classe

Diagramas de Casos de Uso

- Modelo descritivo sugerido

NOME DO CASO DE USO	
Descrição:	**Texto descrevendo o caso de uso**
Ator:	**Lista dos atores que se associam ao caso de uso**

Pré-condição:
Descrição da pré-condição se existir (ex. Receber informações de X, Y, Z)
Cenário Principal:
Descrição textual do cenário principal
Cenário Alternativo:
Descrição textual de cada cenário alternativo
Pós-condição:
Descrição da pós-condição se existir

Diagramas de Casos de Uso

- Exercício:
  - Alvaró está fazendo a amplicação de sua residência. Todo dia existe demanda de compra de material. Sendo assim, ele desenvolveu uma aplicação que controla essa demanda de solicitações e as compras efetuadas.
  - A aplicação possui um cadastro de produtos, contendo: nome, descrição, medida de venda do produto (kg, ml, etc) e o valor da medida de venda (ex 1,5)
  - A cada solicitação de compra cadastram-se os itens desta solicitação. Cada item possui: produto e a quantidade. Quando cada item é adquirido, atualiza-se a solicitação com o preço unitário de compra, a forma de pagamento, a data e o local da compra

Diagramas de Casos de Uso

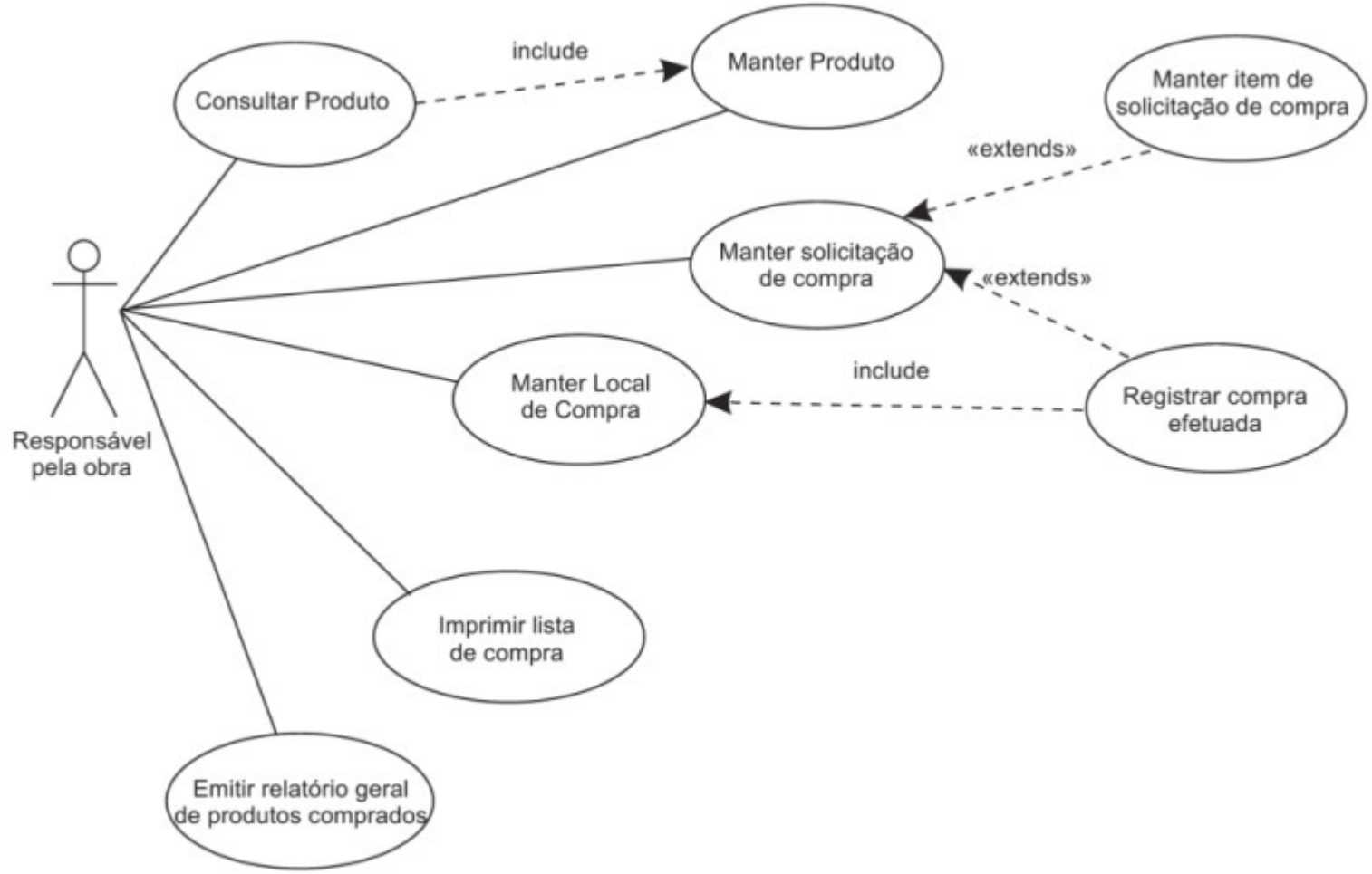
- São controles oferecidos pela aplicação:
  - Quando há uma nova solicitação, é possível obter de cada item a lista dos 3 menores preços que já foram pagos para o referido produto, incluindo na listagem o local onde foi comprado
  - A lista de compras é impressa a partir dos itens que não foram fechados, de todas as solicitações de compra que estejam com status em aberto
  - Uma solicitação pode ser cancelada (status = cancelado)
  - Quando todos os itens de uma solicitação tiverem sido comprados, o sistema atualiza automaticamente o status desta solicitação para “fechado”
  - Deve ser emitida uma listagem de todos os produtos já comprados, com seu somatório de quantidade e valor

Diagramas de Casos de Uso

- A partir do cenário descrito, desenha o diagrama de casos de uso deste sistema e escreva o cenário para consultar produto. Considere que todas as operações são feitas pelo Álvaro, que pode ser identificado como Responsável pela Obra.

20 minutos

Solução do exercício



Solução do exercício

Consultar Produto	
Descrição:	**Tem como objetivo apresentar os produtos cadastrados e habilitar a inclusão, alteração ou exclusão de produtos.**
Ator:	**Responsável pela obra.**

Cenário Pricipal

1. Prepara lista de produtos cadastrados.
2. O sistema oferece ao usuário
   - 2.1 selecionar produto para alterar o cadastro
   - 2.2 localizar um produto
   - 2.3 selecionar a opção de inserir produto
3. Pesquisa de produto
   - 3.1 Usuário deve inserir o nome
   - 3.2 Exibe a lista de produtos com nome e descrição para cada
4. Inserção de produto
   - 4.1 [Include Caso de Uso Manter Produto]
5. Seleção de produto
   - 5.1 Após selecionar um produto, o sistema habilita as opções “alterar produto” e “excluir produto”
   - 5.2 Se o usuário selecionar uma dessas opções o sistema aciona o cadastro de produto. [Include Caso de Uso Manter Produto]

Tarefa 1 (disponível no Moodle)

- Sistema de ar condicionado automotivo
  - Representação do sistema em um PDF no Moodle (agradecimentos ao Roberto Costa da Ford)
  - Descrever todos os cenários existentes usando o modelo apresentado
  - Fazer o diagrama de casos de uso
  - Entrega no dia 15/07 via Moodle e discussão em “sala”

Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Diagrama de Classes

- Através do diagrama de casos de uso, são mapeadas as classes e alguns métodos do sistema
- O diagrama de classes mostra como as diferentes entidades (classes) do sistema modelado se relacionam. Base para qualquer projeto orientado à objetos
- Assumo que os participantes já possuem conhecimentos básicos de classes e objetos e algum contato com uma linguagem orientada à objetos (como C++)

Diagrama de Classes

- Diagrama de classes conceitual
  - Contém apenas classes de conceito formando os modelos de análise (o que o sistema fará, não como fará)
  - Sem detalhes de implementação
- Diagramas de classes de implementação
  - Representa todos os detalhes necessários para a implementação do sistema
  - São bastante extensos e complexos por apresentarem todos os detalhes

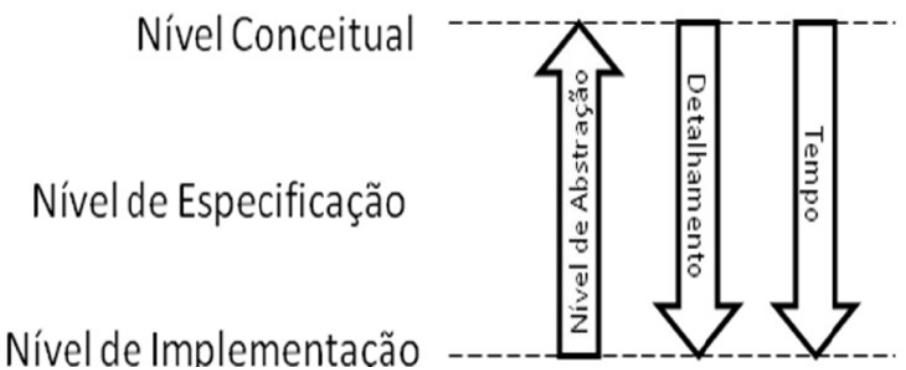


Diagrama de Classes - Conceitos

- Visibilidade: identifica por quem uma propriedade (atributo ou operação) pode ser utilizada

+ ou **public** (público)	A propriedade será vista e usada dentro da classe na qual foi declarada, em qualquer elemento externo (incluindo objetos instanciados a partir desta classe) e nas classes descendentes.
# ou **protected** (protegido)	A propriedade será vista e usada apenas dentro da classe na qual foi declarada e pelas classes descendentes.
– ou **private** (privado)	A propriedade será vista e usada apenas dentro da classe na qual foi declarada.
~ ou **package** (pacote)	A propriedade poderá ser vista e usada por elementos que estejam declarados dentro do mesmo pacote no qual está inserida a classe que a declarou. É como a visibilidade pública, só que não generalizada a qualquer elemento externo, mas apenas aos elementos externos localizados no mesmo pacote.

Diagrama de Classes - Conceitos

- Multiplicidade: indica a quantidade de instâncias possíveis em um relacionamento

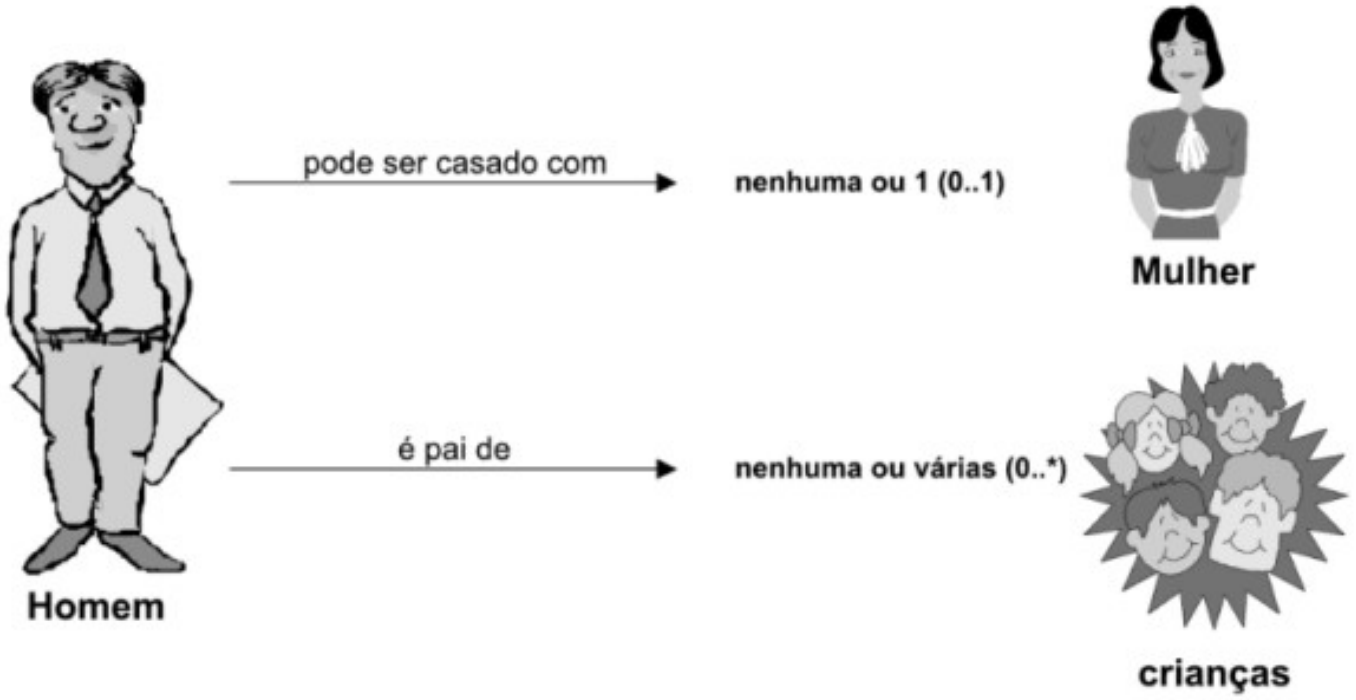


Diagrama de Classes - Conceitos

- Multiplicidade: é demonstrada pela forma

limite-inferior .. limite-superior

- Multiplicidades comuns:
  - 0 .. 1 (valor opcional)
  - 1 ou 1 .. 1 (exatamente um)
  - * ou 0 .. * (qualquer valor inteiro não negativo)
  - 1 .. * (qualquer valor inteiro positivo)
- Algumas normas
  - Adequado: 2 . . 5, 8, 10 .. 15 Inadequado: 10 .. 15, 2 .. 5, 8
  - Adequado: 3 .. 8 Inadequado: 3 .. 5, 6 .. 8

Diagrama de Classes - Conceitos

- Estático: membros da classe estáticos (atributos ou métodos) devem ser sublinhados

- Estereótipo: é uma extensão da UML que permite distinguir um uso específico para um elemento
  - O nome do estereótipo é colocado entre << >>
  - Ex: <<interface>>

- Notas ou comentários: símbolo gráfico contento informação textual

Diagrama de Classes - Conceitos

- Restrições: podem aparecer em diversos elementos da UML
  - É uma condição expressa textualmente
  - Uma restrição é mostrada através de chaves
    Por exemplo: {valor > 1000}
- No diagrama de classes, podemos incluir restrições em associações ou atributos
  - Ex: restrição ligada ao atributo
    qtdSolicitada: int {qtdSolicitada > 0}

Diagrama de Classes

- Representação de classe

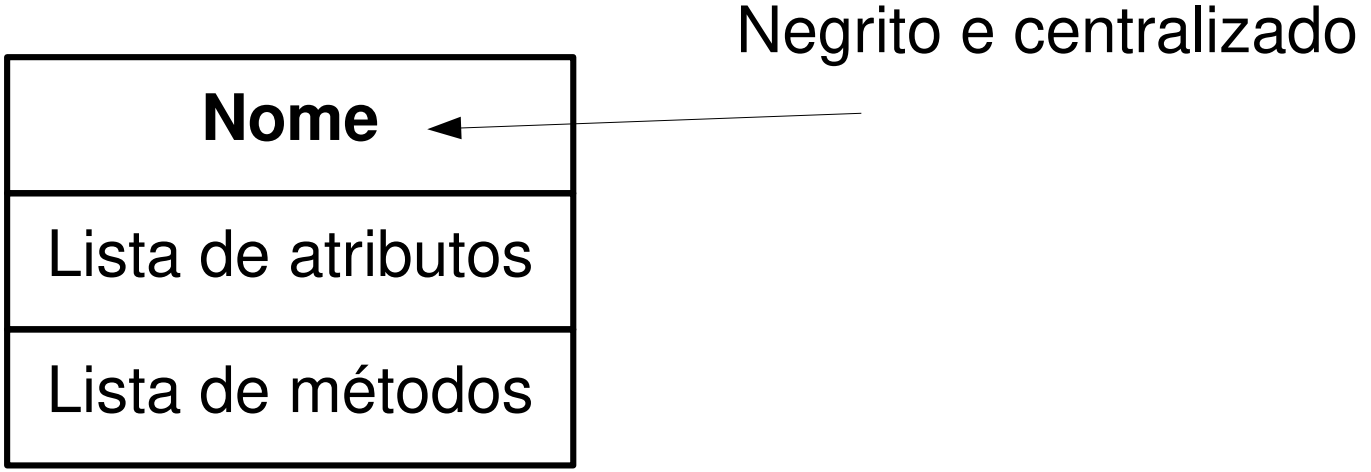


Diagrama de Classes

- Representação de atributos

  [visibilidade] [/] nome : tipo [multiplicidade] [= valor padrão]

- O “/” indica que o atributo é derivado
  - Ex: idade e dataDeNascimento
  - O atributo idade pode ser derivado da dataDeNascimento
- Exemplos

  - dataDeNascimento : Date;

  - / idade : int;

  + nomeContato : String[0 .. 2];

Diagrama de Classes

- Representação dos métodos (operações)

  [visibilidade] nome([lista de parâmetros]) : tipo de retorno

- Onde lista de parâmetros segue o formato

  [direção] nome : tipo

- Direção é opcional e pode ser “in” ou “out” ou “inout”
- Exemplos

  + getIdade() : int;
  + setDataDeNascimento(in dataNascimento : Date);

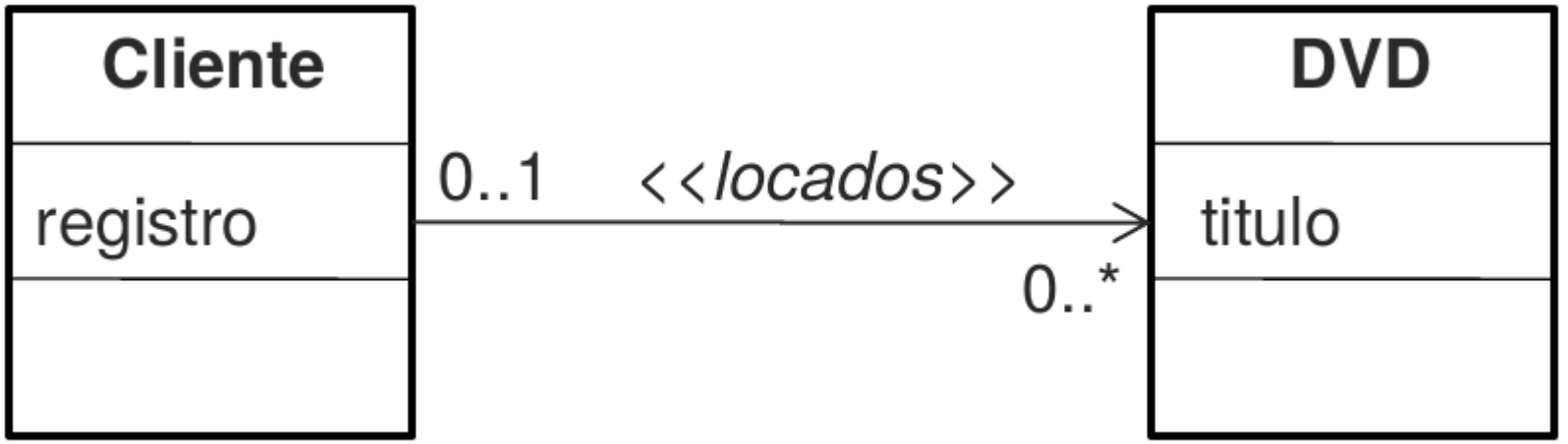
Relacionamento entre Classes

- Classes possuem relacionamentos entre elas
  - Compartilham informações
  - Colaboram umas com as outras

- Principais tipos de relacionamento
  - Associação
  - Agregação / Composição
  - Herança (generalização / especialização)
  - Dependência

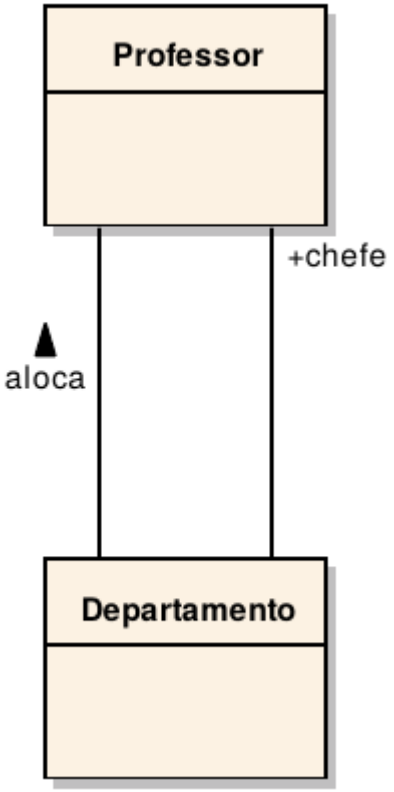
Associação

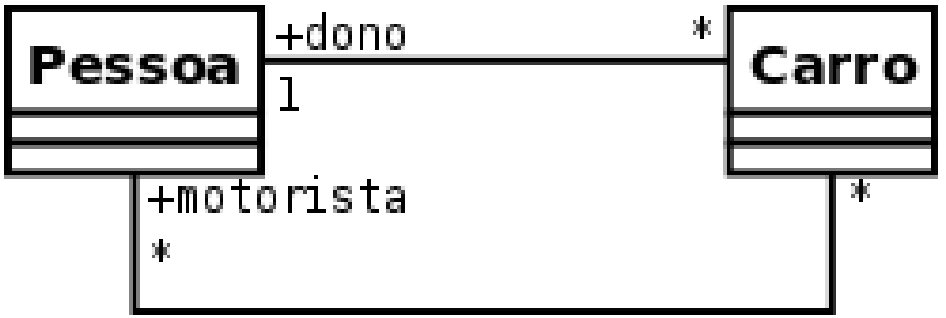
- Descreve um vínculo entre duas classes
- Determina que as instâncias de uma classe estão de alguma forma ligada às instâncias de outra classe
- Pode ser opcionalmente rotulada com o nome da associação



Associação

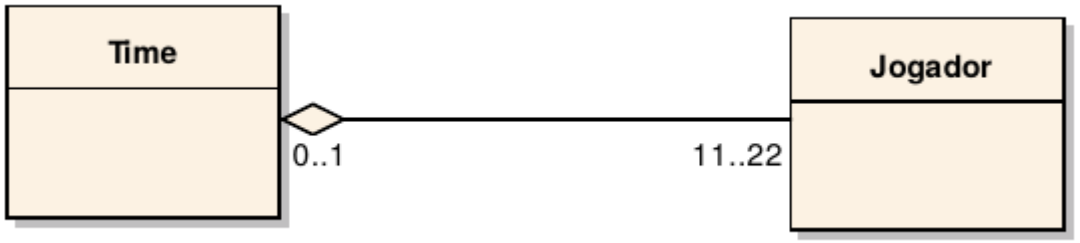
- Papel na associação
  - As pontas das associações chamam-se papéis
  - Podem ser usados para dar nomes aos papéis que as classes representam nas associações





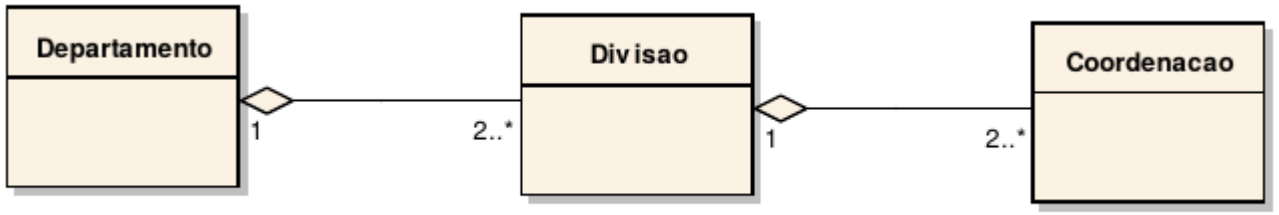
Agregação

- É um tipo especial de associação
- Demonstra que as informações de um objeto precisam ser complementadas por um objeto de outra classe
  - Relacionamento todo-parte, representando o “todo” associado às “partes” que o compõem
  - Representada por um losango na extremidade da classe que contém a outra



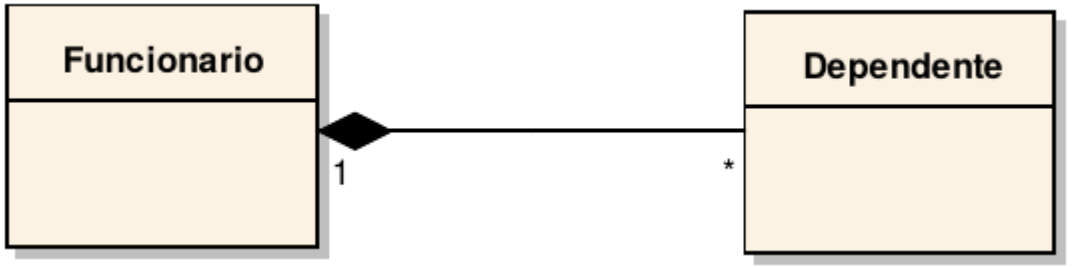
Agregação

- As agregações são bem lidas com o verbo “tem”. Ex: Time têm jogadores
  - Ex.Cada departamento é subdividido em pelo menos duas divisões e estas são divididas em no mínimo duas coordenações
  - Ex. Cada coordenação está associada à sua respectiva divisão e cada divisão ao seu respectivo departamento



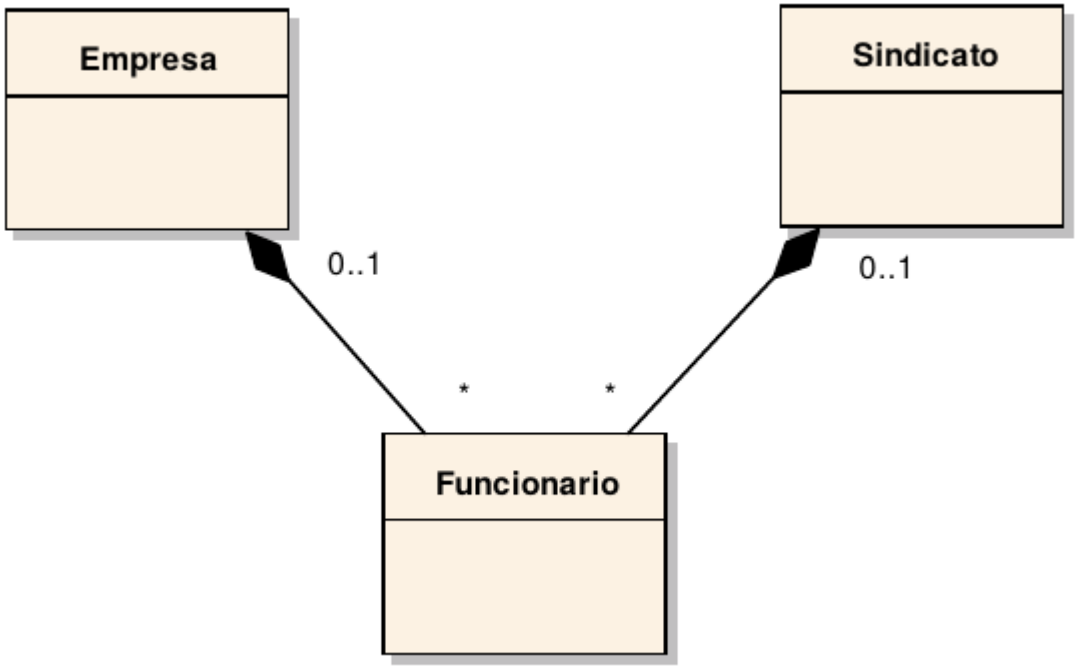
Composição (ou agregação composta)

- Uma variação do tipo de agregação
- Representa um vínculo mais forte entre as classes
  - As partes não podem pertencer, em um mesmo instante, a mais do que um todo
  - Se o todo deixa de existir, as partes também
- Representado graficamente por um losango preenchido



Composição

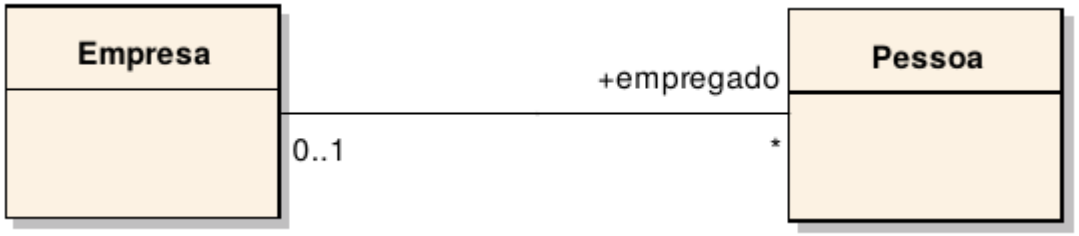
- O que o exemplo abaixo nos permite concluir?



- Se um determinado funcionário está associado a uma empresa, não pode estar associado ao mesmo tempo a um sindicato

Classes de Associação

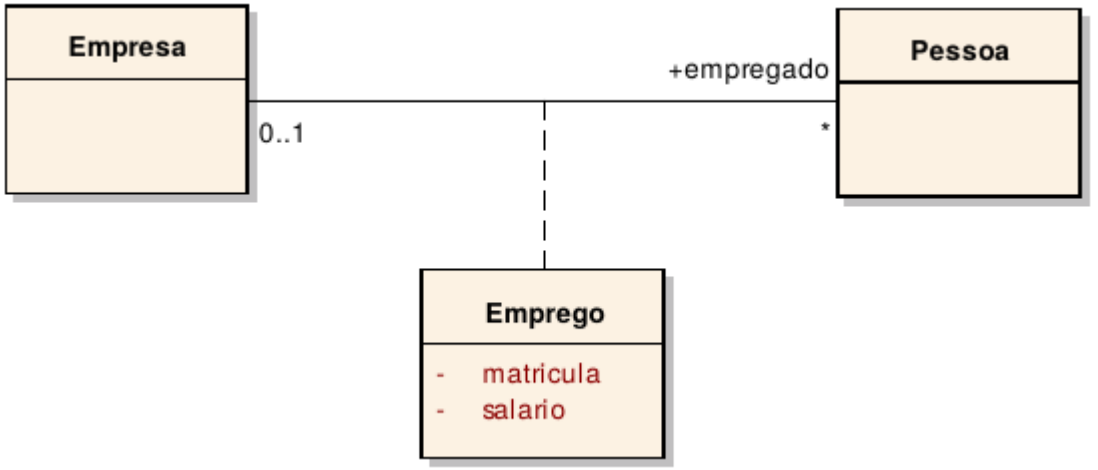
- Considerando a associação abaixo



- Onde seriam armazenados os atributos matrícula e salário de um empregado? Na classe Pessoa? Na classe Empresa?
- Esse atributos passam a ser necessários somente quando uma pessoa se associa a uma empresa
- Portanto, os atributos matrícula e salário são atributos da associação entre as classes

Classes de Associação

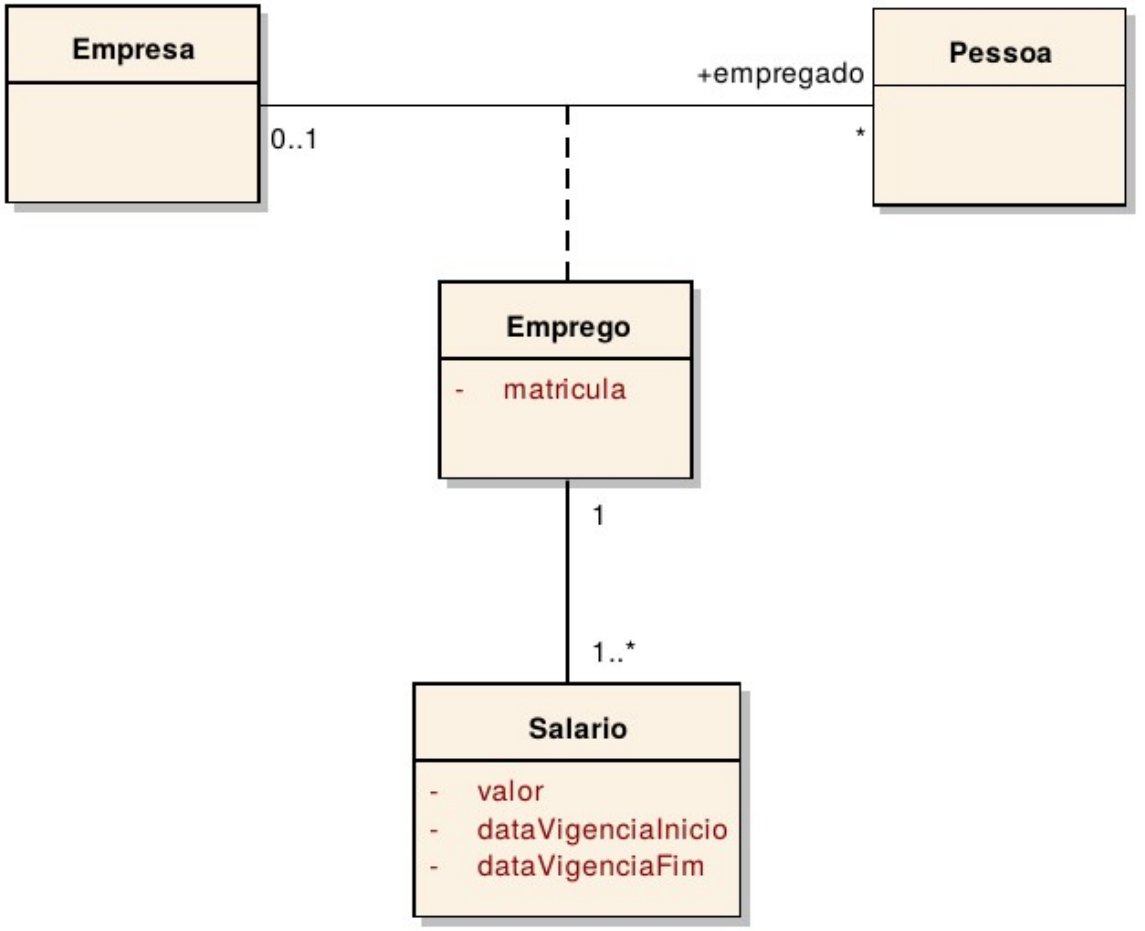
- Classes de associação são representadas conforme a figura abaixo



- A classe Emprego não permite guardar o histórico do salário

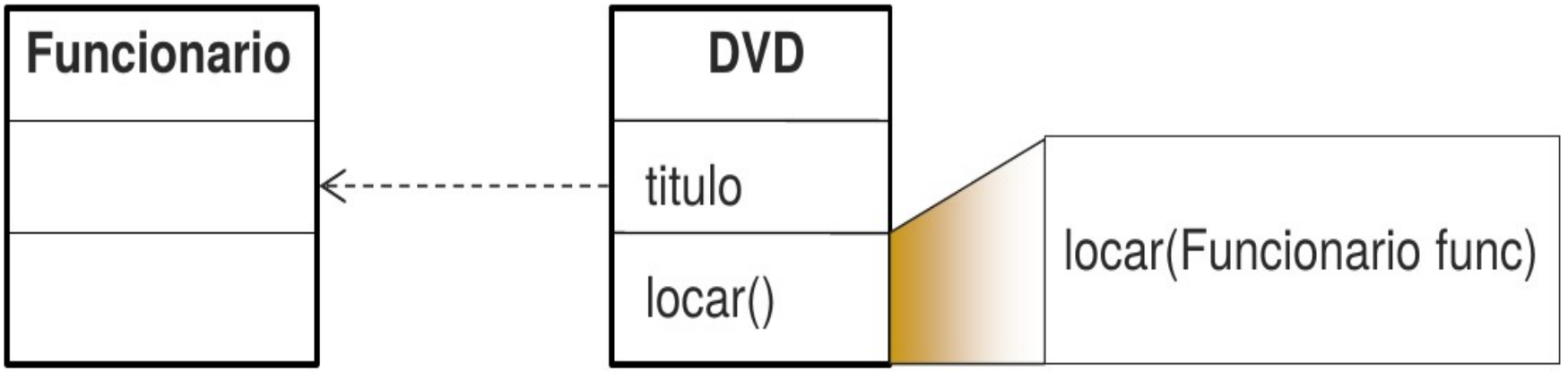
Classes de Associação

- Para guardar o histórico de atributos, não se deve usar classes de associação e sim classes normais



Dependência

- Tipo menos comum de relacionamento
- Identifica uma relação fraca entre objetos de duas classes
- Reta tracejada entre duas classes
- Uma seta na extremidade indica o dependente

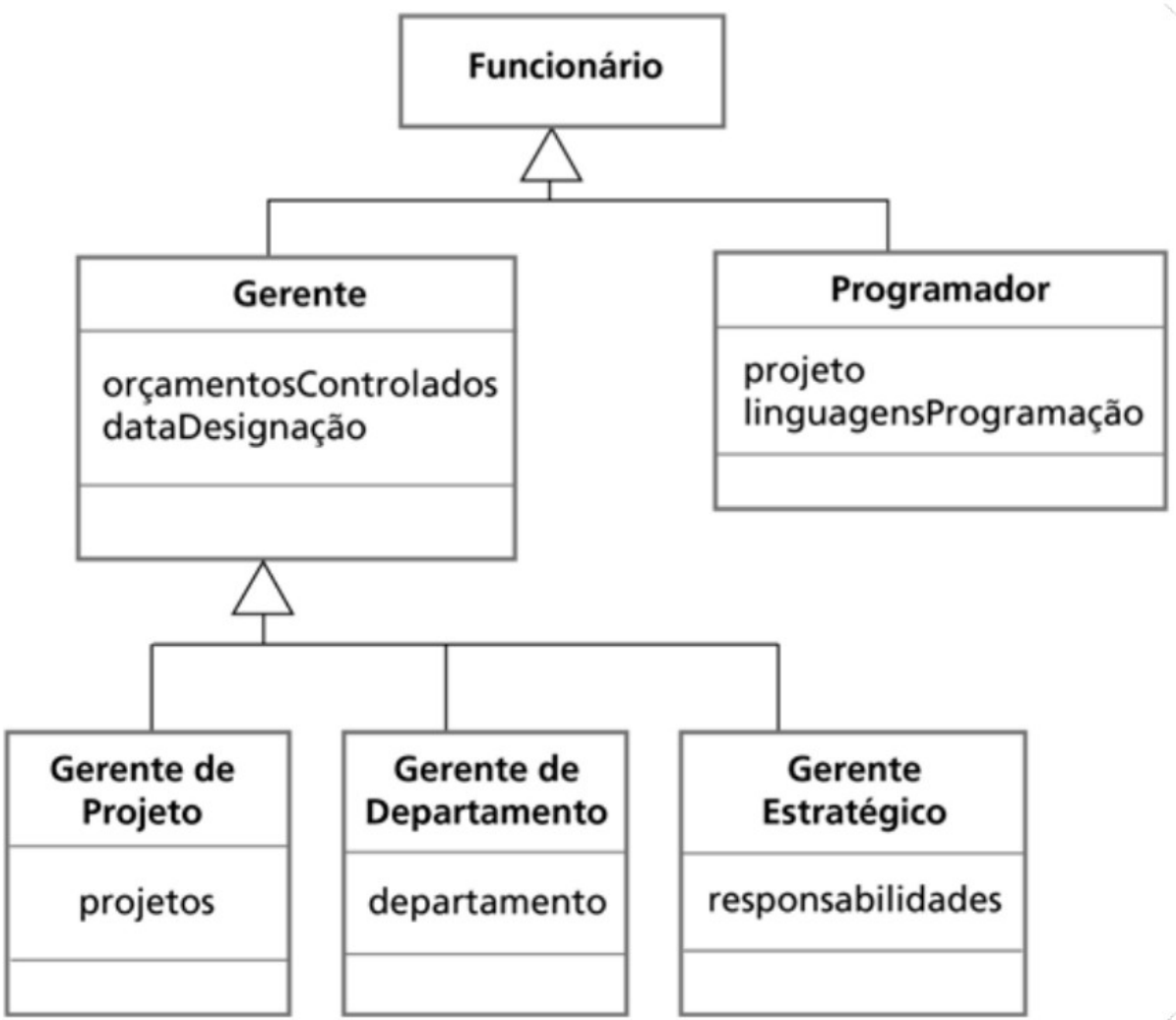


Especialização / Generalização

- Identifica uma hierarquia de classes através de super-classe (geral ou classe pai) e sub-classes (especializadas ou classe filhas)
  - Tudo que a classe pai pode fazer, as filhas também podem
- Constitui uma relação de herança
  - A hierarquia representando a herança é uma fonte de conhecimento sobre o domínio do sistema
  - Abstração
  - Mecanismo de reuso de software bastante eficiente

Especialização / Generalização

- Especializações de funcionários

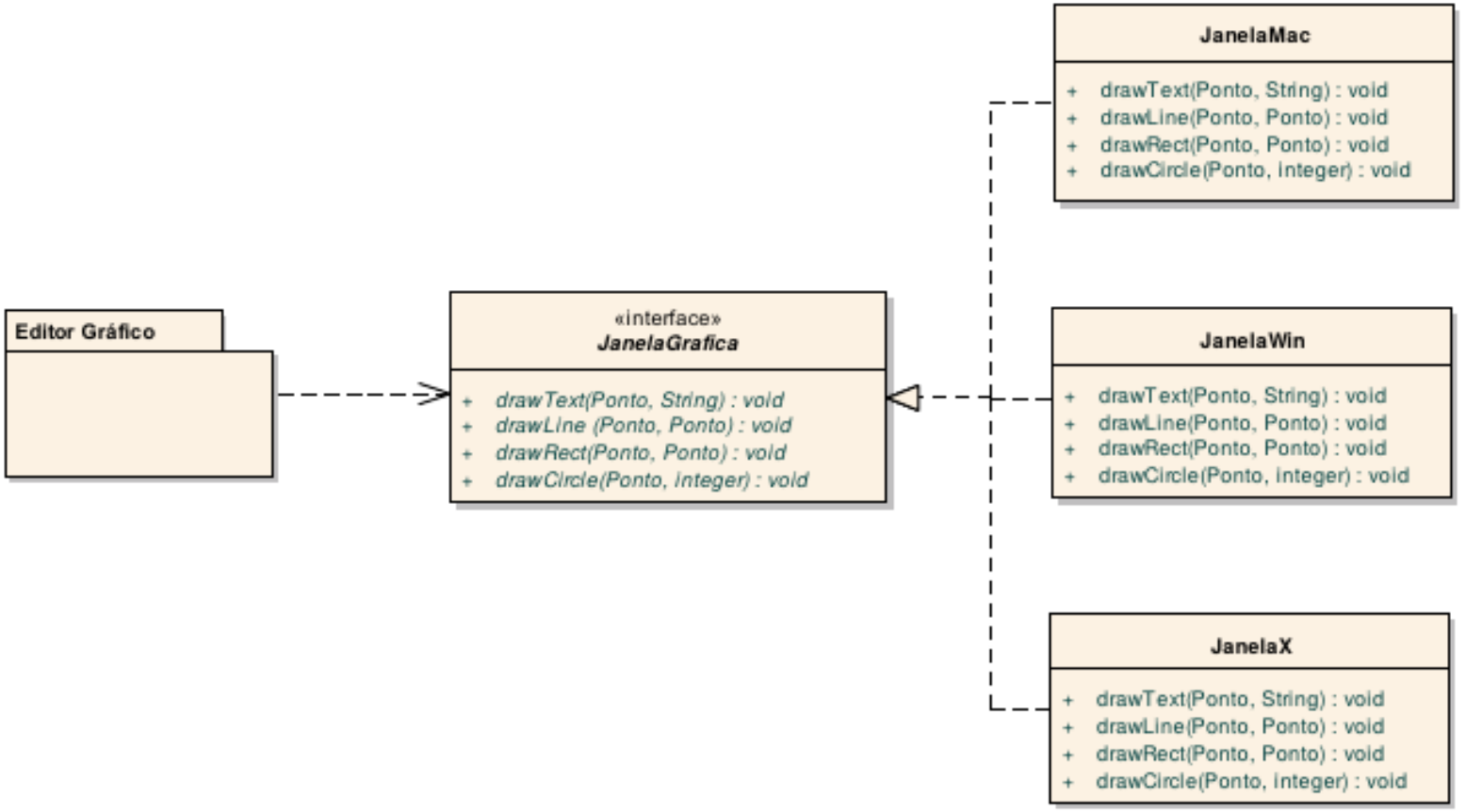


Classe Abstrata

- Possuem apenas a declaração das operações, sem implementação
  - Funcionam como uma interface na hierarquia de classes
  - Toda classe derivada deve implementar os métodos abstratos da classe pai
- Esse tipo de classe nos leva a dois novos conceitos da UML
  - Interface: representada através do estereótipo <<interface>> em cima do nome da classe
  - Realização da Interface: identifica que as classes filhas implementam a interface (linha tracejada com a seta preenchida)

Classe Abstrata

- Exemplo:



Exercício

- Controle de comanda eletrônica de uma padaria
- O atendente registra na comanda (que possui numeração) o produto e a quantidade
- Ao passar no caixa, são lidos os gastos da comanda, verificando-se o valor unitário de cada produto e o valor total da compra
- Identifique as classes, atributos e métodos deste cenário

5 minutos

Exercício: solução

Classe	**Atributos**	**Métodos**
Produto	codigo : String descricao : String precoUnitario : float	cadastrar
Comanda	numero: int /valorTotal : float /itens : coleção de ItemComanda	RegistrarItem(item: ItemComanda) fecharComanda
ItemComanda	/produto : Produto quantidade : int	cadastrar

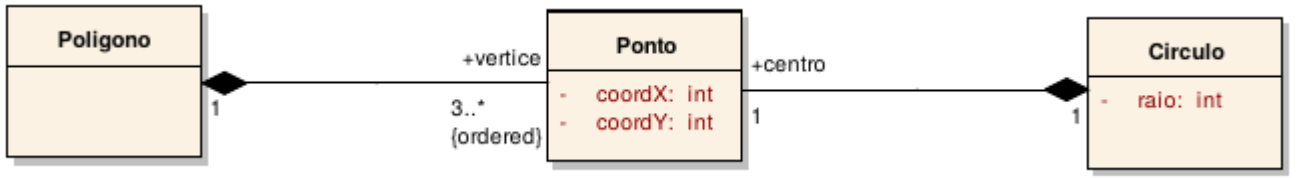
Exercício

- Estamos desenvolvendo um editor de texto para permitir o desenho de polígonos (especificados por conjuntos ordenados de 3 ou mais pontos ligados por segmentos de reta) e círculos (especificados por centro e raio) na tela
- Faça a modelagem do Polígono e Círculo utilizando as seguintes restrições
  - Um polígono possui 3 ou mais vértices
  - Um vértice de um polígono não pode ser ao mesmo tempo vértice de outro polígono nem centro de um círculo
  - O centro de um círculo não pode ser centro de outro círculo
  - Ao remover um polígono, remove-se também seus vértices (da mesma forma com círculo)

15 minutos

Exercício: solução

- Classe Ponto que representa o par ordenado P(x,y)
- Se um ponto está associado a um polígono (fazendo o papel de vértice) não pode estar associado, ao mesmo tempo, a um círculo
- Multiplicidade 1 garante que se o Polígono ou Círculo forem removidos, seus Pontos também serão



Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Diagrama de Componentes

- Um componente em UML corresponde a um agrupamento físico de elementos do modelo que provê um conjunto de funcionalidades
- Os componentes se relacionam com a arquitetura do sistema
  - Como ele é projetado e implementado
- Mostram a estrutura do modelo de implementação
- Um componente pode corresponder a um subsistema ou a qualquer outra porção reutilizável
  - Mostra apenas o que é relevante do ponto de visto do resto do sistema e encapsula o resto

Diagrama de Componentes

- Componentes são utilizados para
  - Dividir o sistema em módulos
  - Facilitar o reúso
  - Facilitar a manutenção do sistema
- Notação
  - Comunicação entre componentes feita através de interface fornecida e exigida
  - Fornecida: círculo
  - Exigida: semicírculo externo ao da interface fornecida

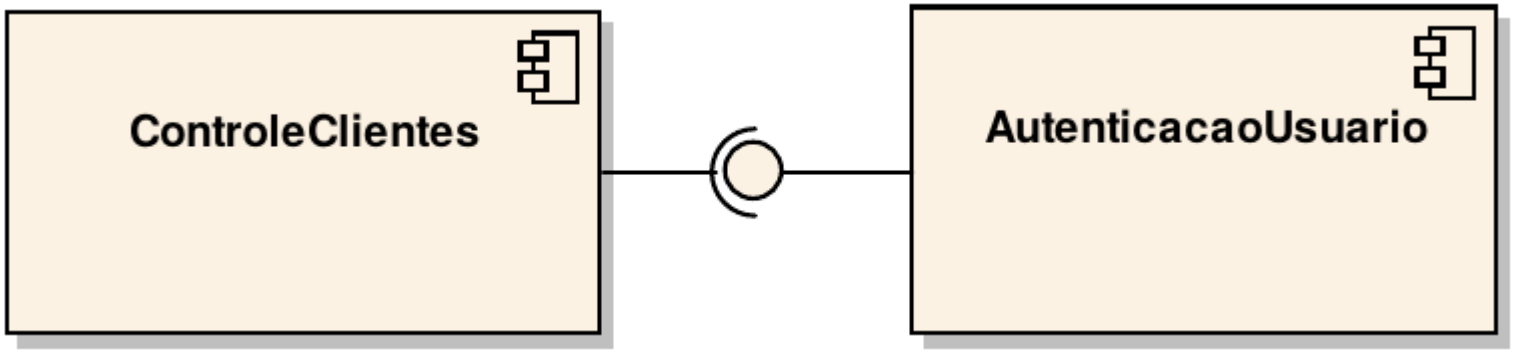
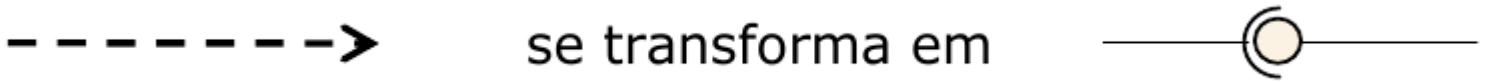


Diagrama de Componentes

- Em um diagrama de pacotes, se usa a relação de dependência entre interfaces e pacotes
- A relação de dependência é substituída pela notação de interface fornecida e exigida



- Um componente pode ainda definir nome para portas, dando diferente significados para a interação com o mundo exterior

Diagrama de Componentes

- Portas podem ser uni ou birecionais
- Ex: Student tem 3 portas, duas unidirecionais e uma birecional

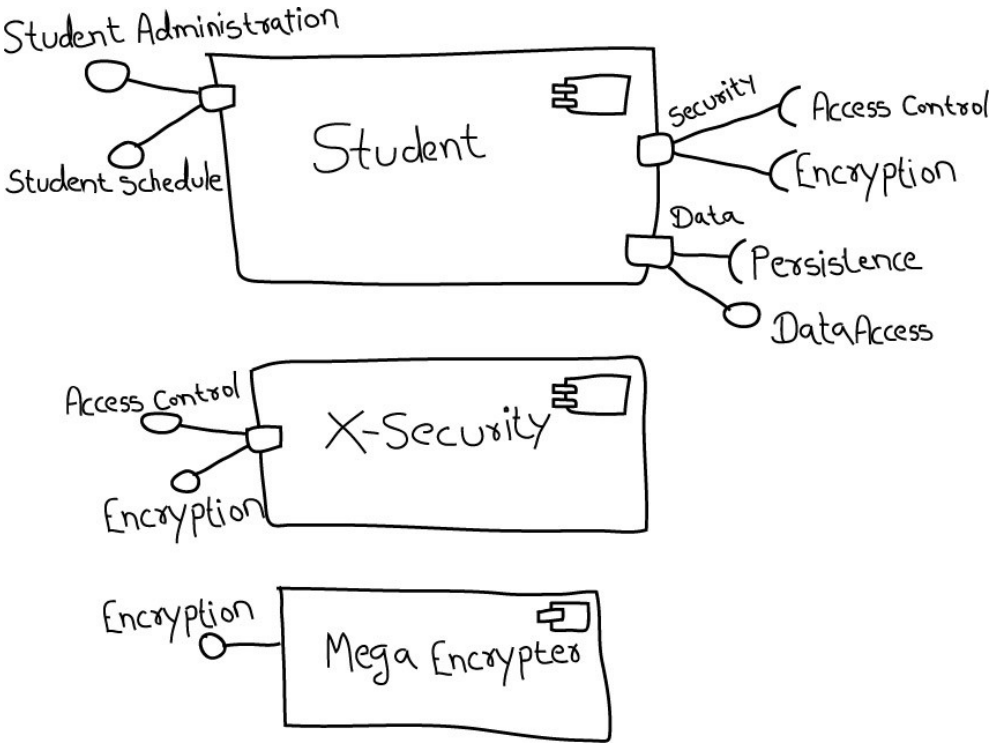


Diagrama de Componentes

- Como projetar um componente?
- Conjunto de classes
  - Identificar as interfaces exigidas e fornecidas
  - Isso é feito usando as relações de dependência (flecha aberta) e realização de interface (flecha fechada)

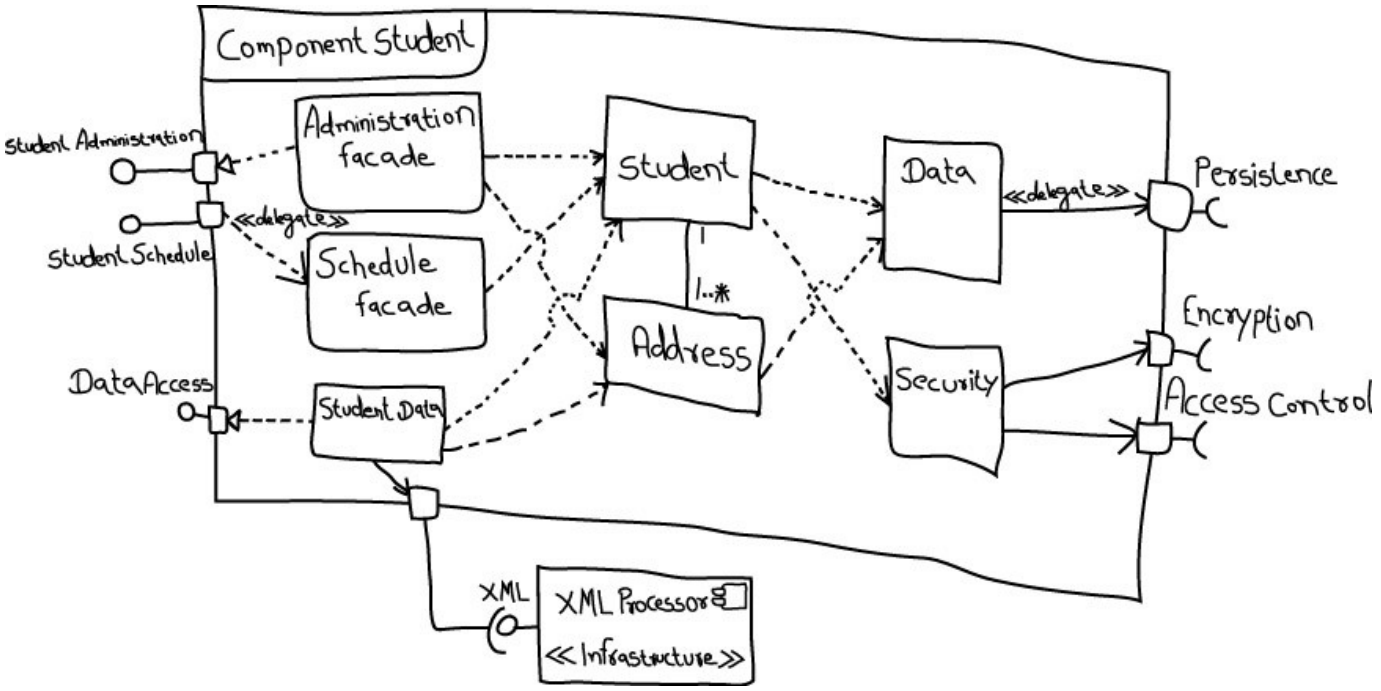


Diagrama de Componentes

- “There are several advantages to components that promote agility. First, components are reusable building blocks from which you can build software, increasing your productivity as a developer. Second, components can improve your testing productivity because they can be treated as elements, which you can black-box unit and integration test.”
  - Agile Model-driven Development with UML 2.0. Scott W. Ambler

Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Diagrama de Máquina de Estados

- Uma determinada classe pode possuir instâncias com diferentes estados
- Conta corrente:
  - Superavitária, credora, bloqueada
- Diagrama de Máquina de Estados (DME) especifica não só os estados pelos quais os objetos podem passar e as possíveis transições entre os estados, mas também as reações dos objetos aos eventos que os atingem

Diagrama de Máquina de Estados

- Considere uma classe Pedido de uma empresa
- Queremos especificar as possíveis situações (estados) em que os pedidos (instâncias da classe Pedido) podem estar
- Para isso precisamos considerar alguns exemplos:
  - O pedido número 346 está na rua para entrega
  - O pedido número 349 está aguardando a reposição de estoque de determinado produto
  - O pedido 330 foi devolvido
  - O pedido 390 está sendo verificado para determinar se todos os itens que o compõem estão disponíveis em estoque

Diagrama de Máquina de Estados

- Com base nas informações, o seguinte DME foi desenvolvido

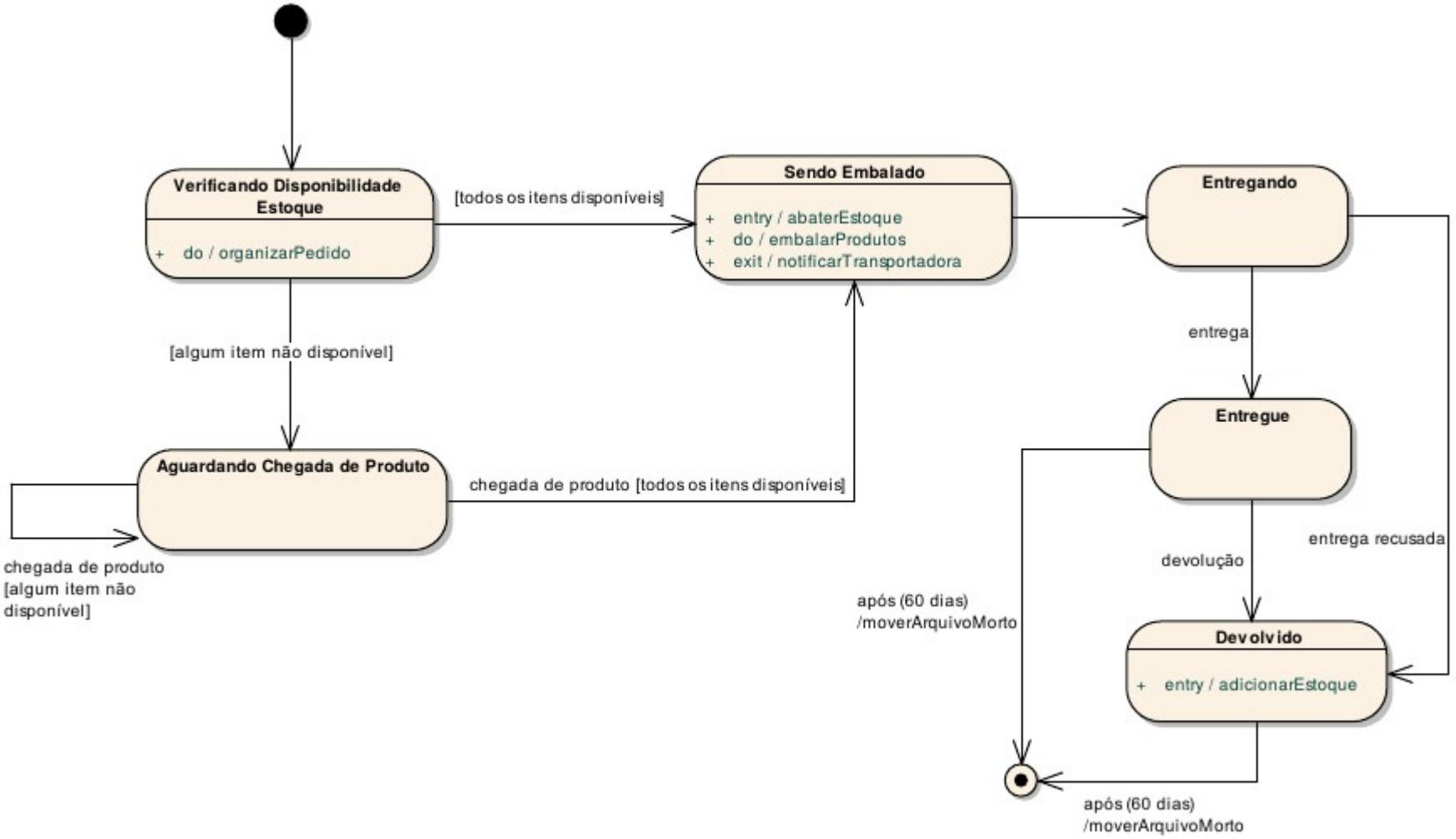
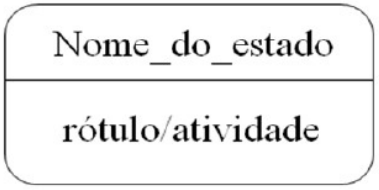
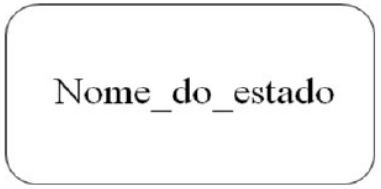


Diagrama de Máquina de Estados

- Representação de estados: retângulos com os vértices arredondados, com um ou dois compartimentos
  - Nome do estado é colocado centralizado
  - O segundo compartimento é opcional e é utilizado para relacionar as atividades a serem executadas assim que o objeto entrar no estado, permanecer e antes de sair



ou



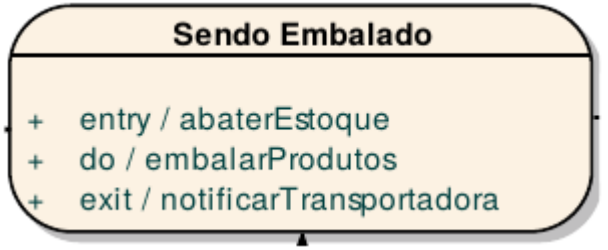


Diagrama de Máquina de Estados

- Qualquer estado é necessariamente de um dos tipos descritos a seguir:
  - Estado de atividade (ex: verificando disponibilidade estoque, sendo embalado e entregando). Objeto executa alguma atividade obrigatoriamente. Nome no gerúndio.
  - Estado de espera (ex: aguardando chegada de produto)
  - Estados que refletem que o objeto atingiu determinada condição (ex: Entregue e Devolvido). Verbos no particípio (tal coisa calculada, tal coisa pesquisada)

Diagrama de Máquina de Estados

- Estado inicial

- Estado final

- Eventos, condições e ações em transições

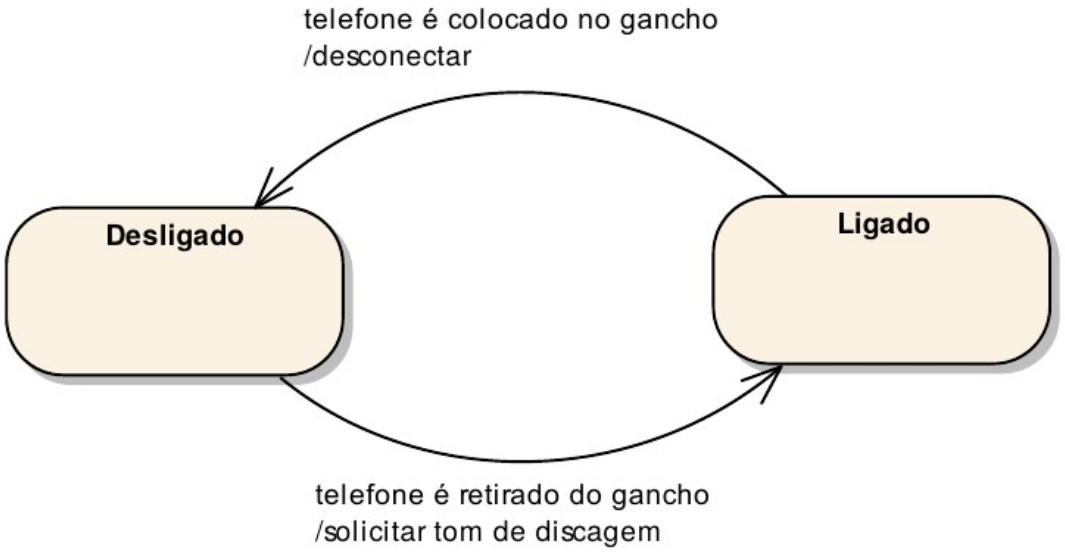


Diagrama de Máquina de Estados

- A sintaxe dos rótulos das transições é

  evento [condição] / ação

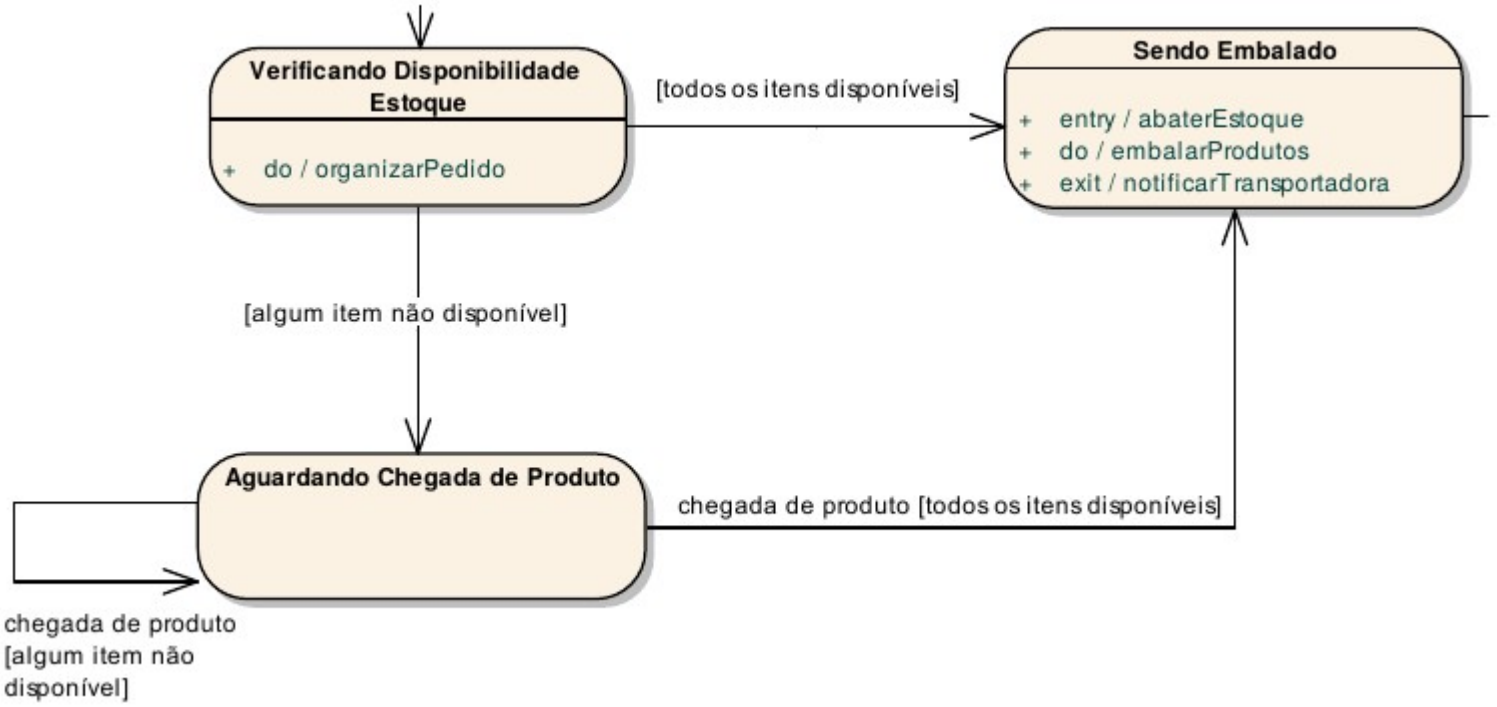


Diagrama de Máquina de Estados

- Estados compostos
  - Quando um estado contém outros estados, concorrentes ou independentes
- Ex: estado ligado do telefone

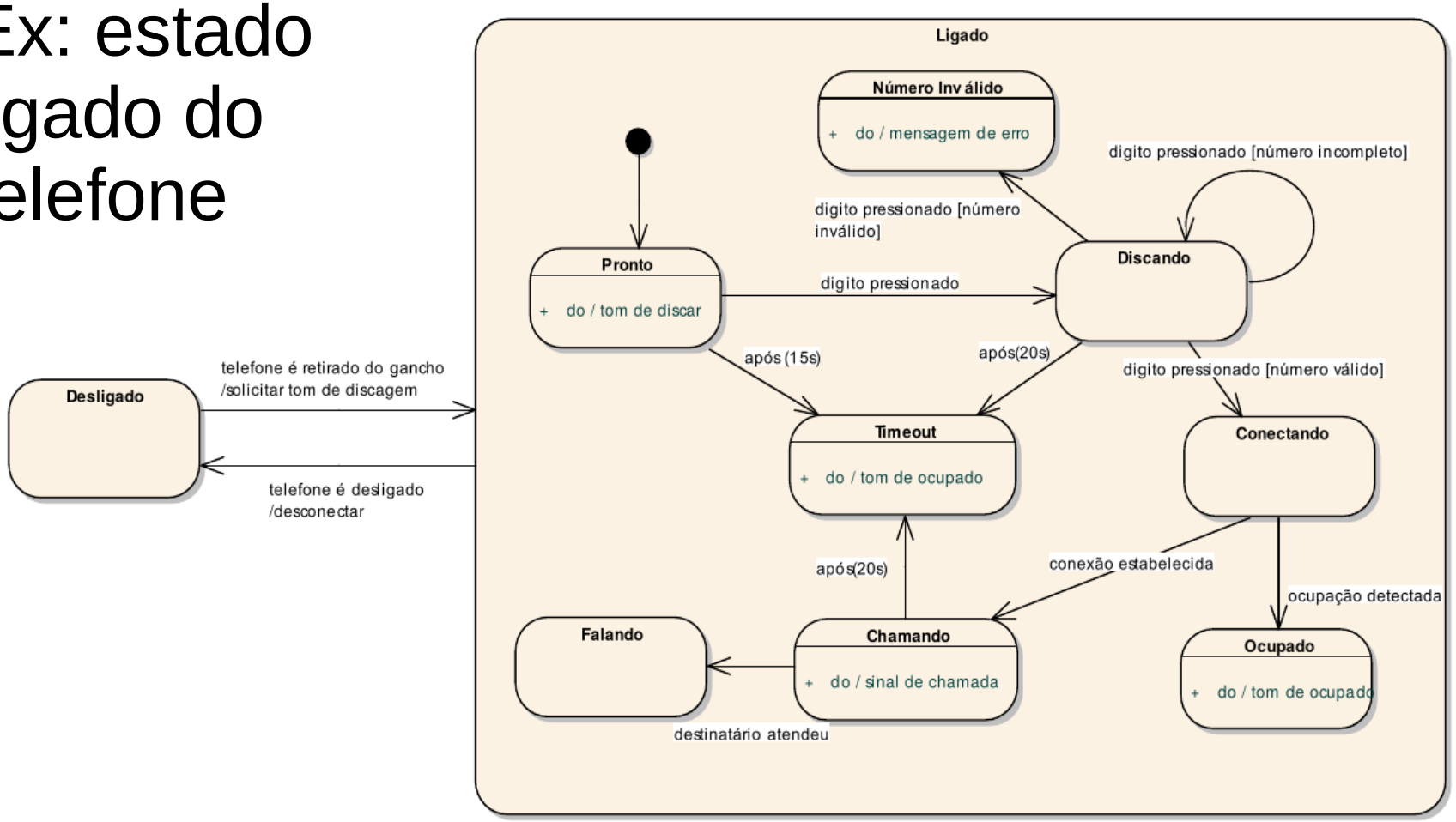


Diagrama de Máquina de Estados

- Pagamento do Pedido (estados concorrentes)

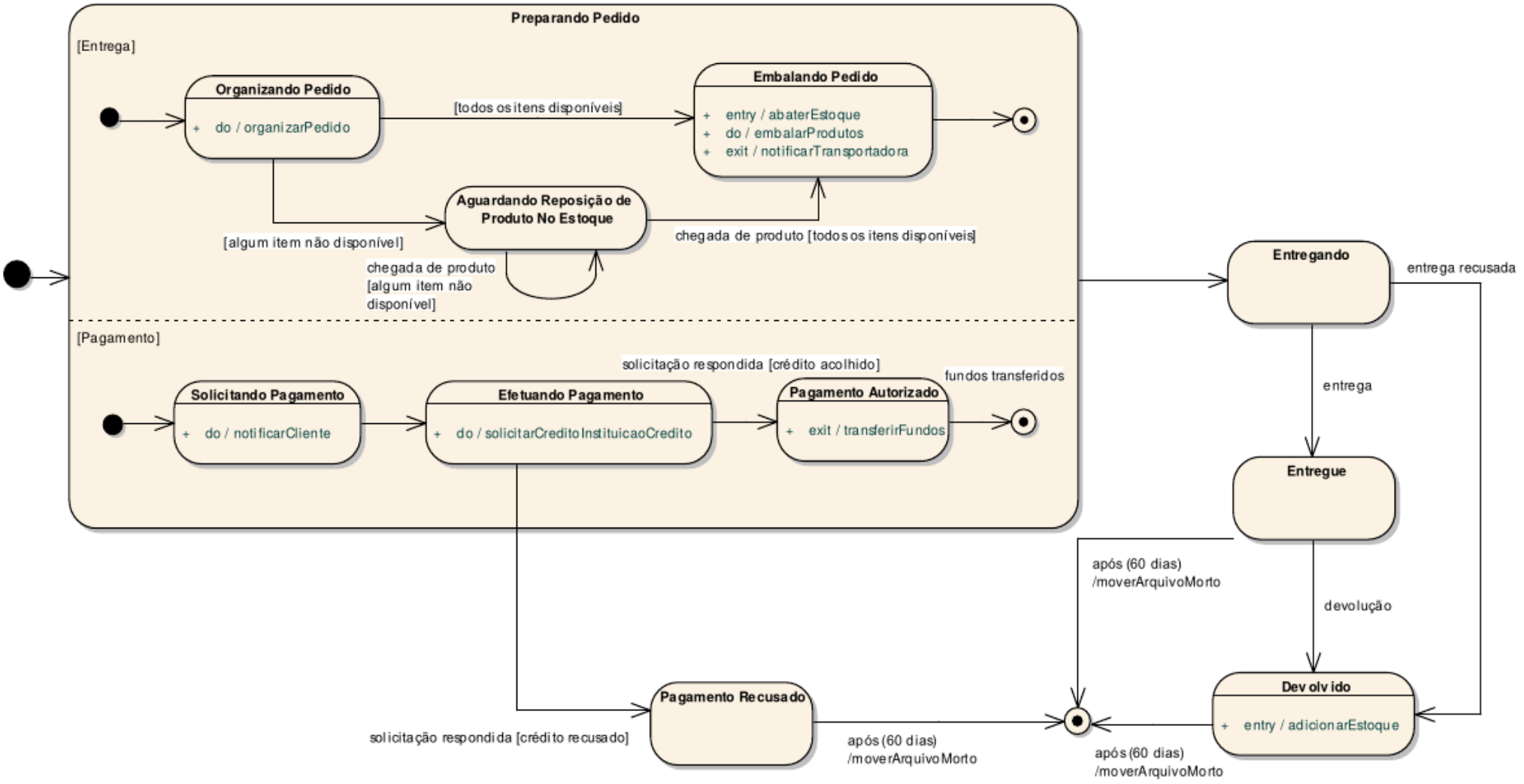


Diagrama de Máquina de Estados

- Ajudam a descobrir casos de uso pois as mudanças de estados estão associadas aos casos de uso
- Verificar os eventos que provocam as transições entre os estados nos DMEs pode ajudar a analisar a completude do diagrama de casos de uso
- Os DMEs também podem ser usados dentro dos diagramas de sequência, na fase do projeto/implementação

Agenda

- Introdução a UML e SysML
  - Diferenças e histórico
- Diagramas de modelagem UML
  - Casos de uso
  - Classes
  - Componentes
  - Estados
  - Sequência
- Exemplos
- Exercícios

Diagrama de Sequência

- Usado na modelagem comportamental
- Especifica colaborações entre objetos, descrevendo a sequência de mensagens passadas de objeto a objeto necessária para a realização de uma determinada operação (como uma funcionalidade do sistema)
- Diagramas de sequência são melhores que diagramas de comunicação na apresentação das responsabilidades de cada objeto, especialmente quando o aspecto de ordenação temporal é relevante
- Usados para descobrir detalhes de implementação (como parâmetros dos métodos)

Diagrama de Sequência

- Os diagramas de sequência (DS) juntamente com o diagrama de classe com detalhes suficientes podem ser suficientes para gerar código automaticamente (depende de uma boa ferramenta)

- Formam o tripé da análise
  - Casos de uso: o que é para ser feito
  - Classes: com quem faremos o que é para ser feito
  - DS: como faremos o que é para ser feito, considerando com quem faremos

Diagrama de Sequência

- Relação entre os 3 diagramas

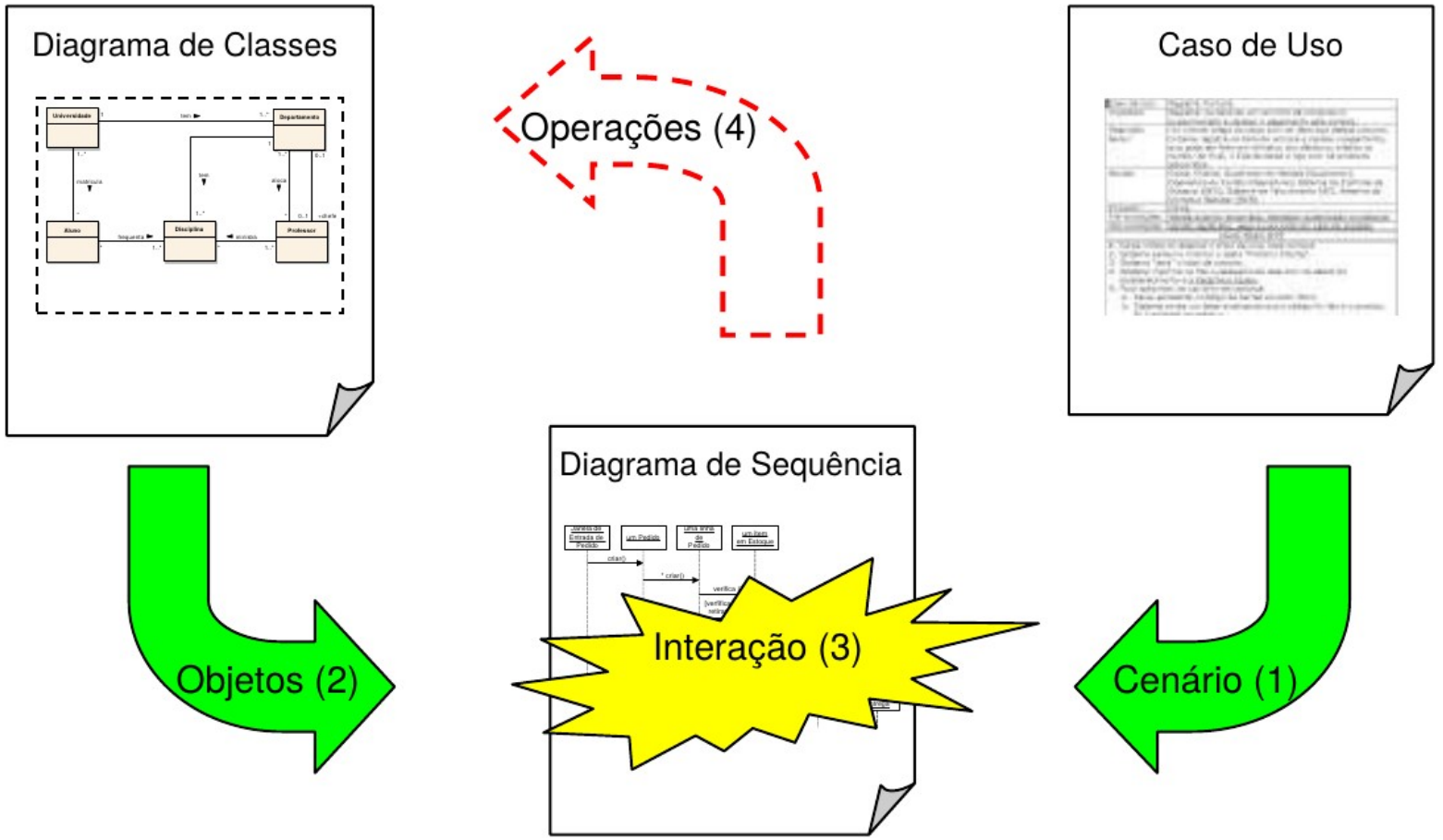


Diagrama de Sequência

- Ciclo de vida de um objeto em um diagrama de sequência
  - Nível de detalhes: documentação x geração de código

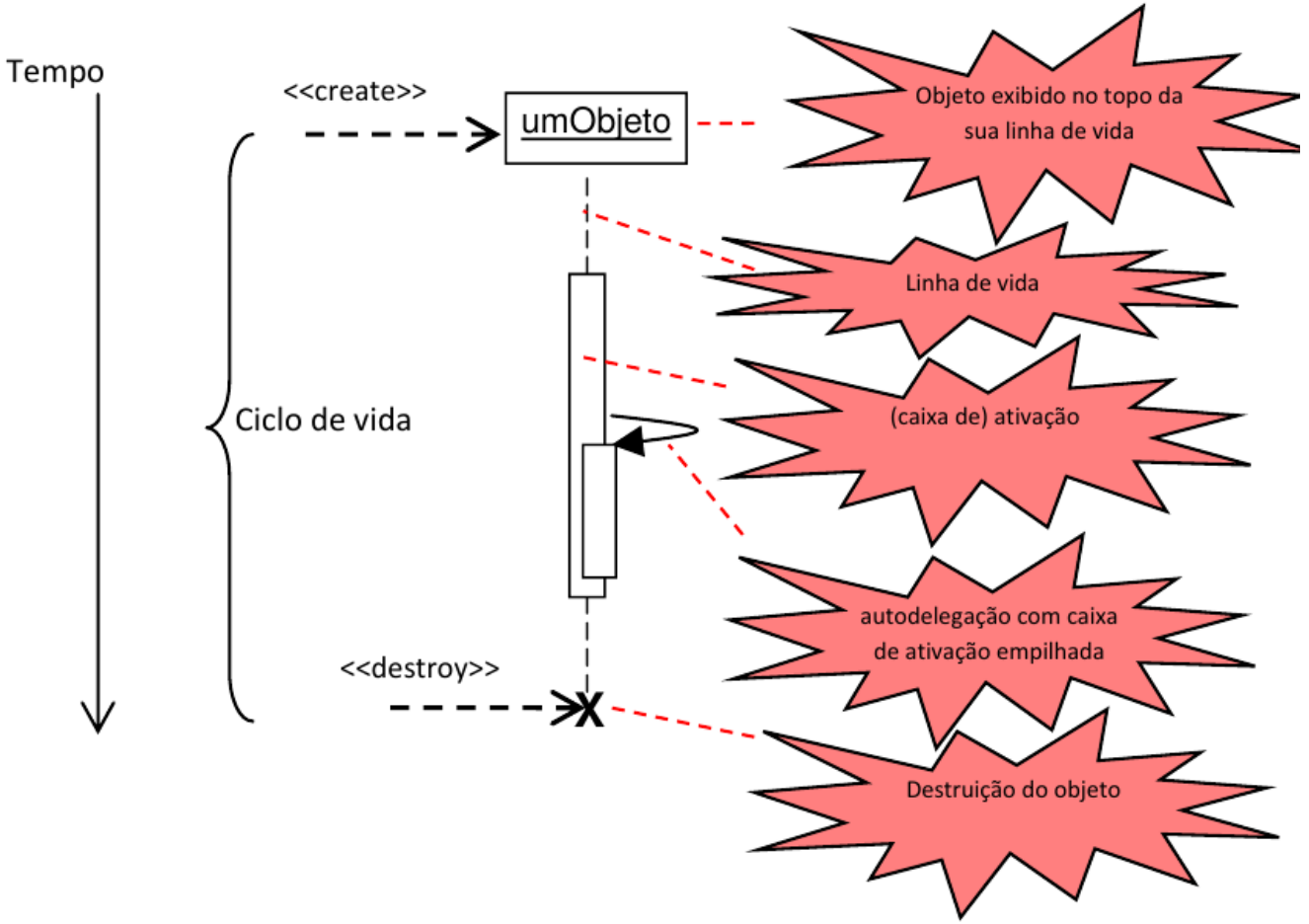
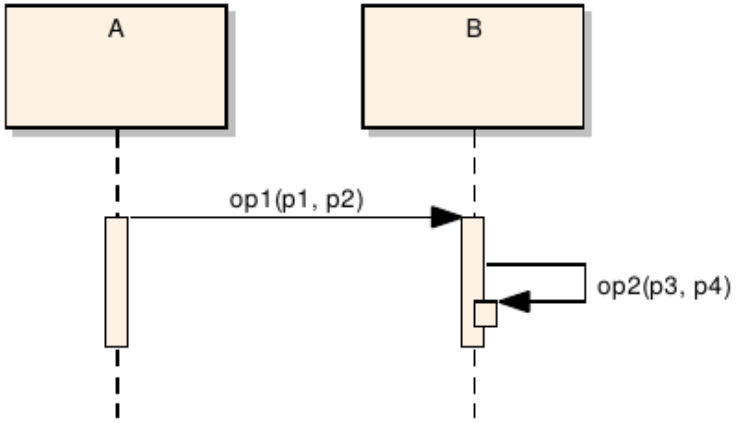
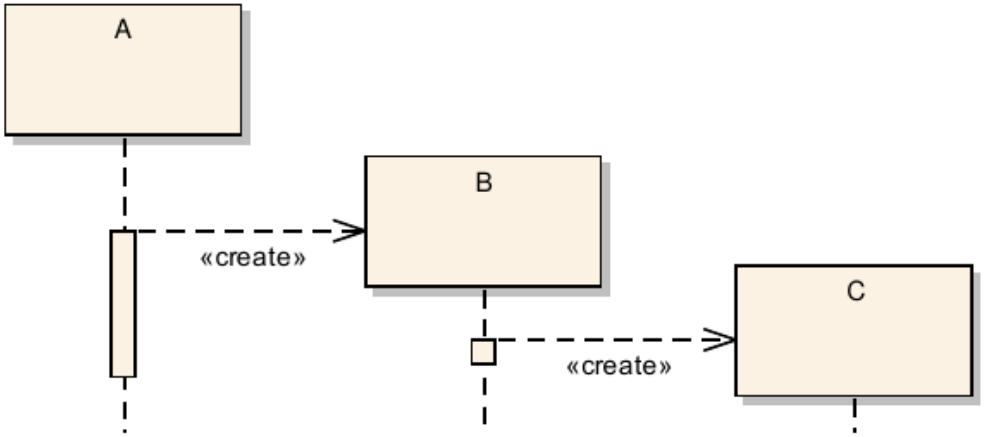


Diagrama de Sequência

- Mensagens de chamadas



- Mensagem de criação e destruição



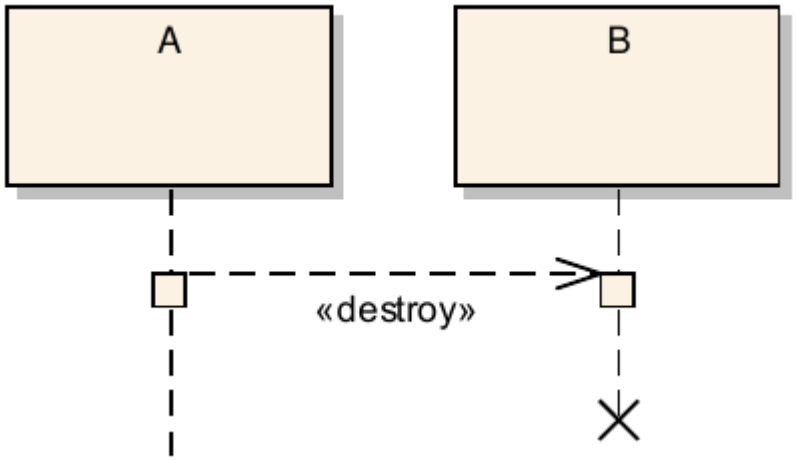
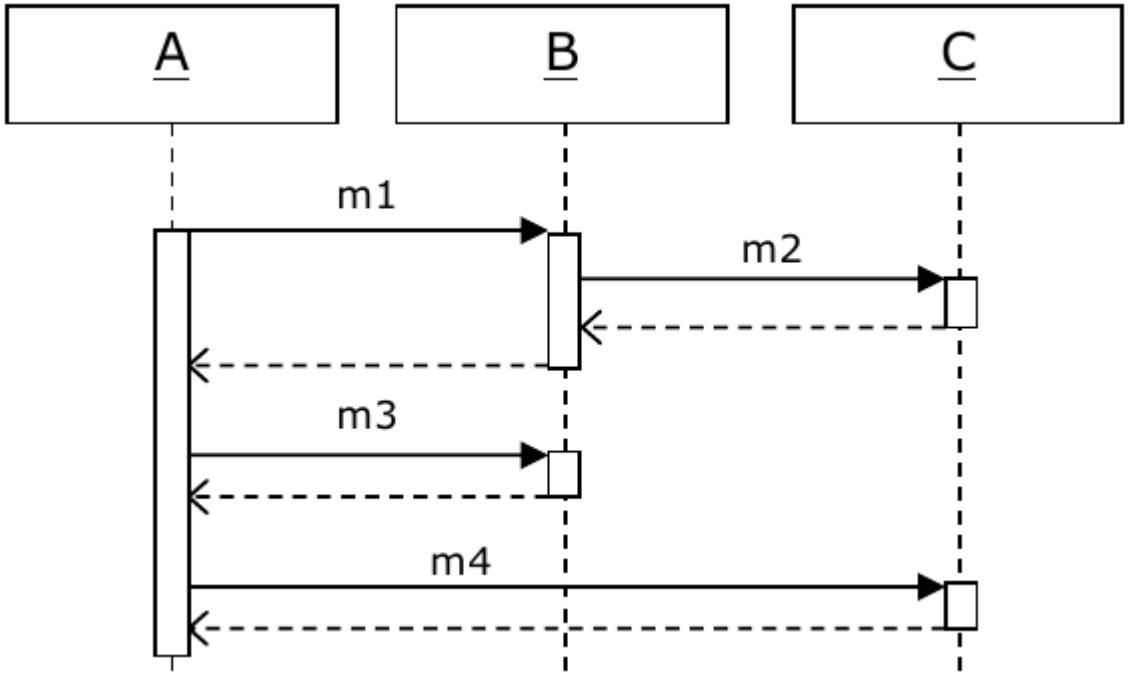


Diagrama de Sequência

- Mensagem de retorno



- Mensagens síncronas representadas pelas setas fechadas

Diagrama de Sequência

- Mensagens assíncronas (uso de Threads por exemplo)
- Representadas por setas abertas

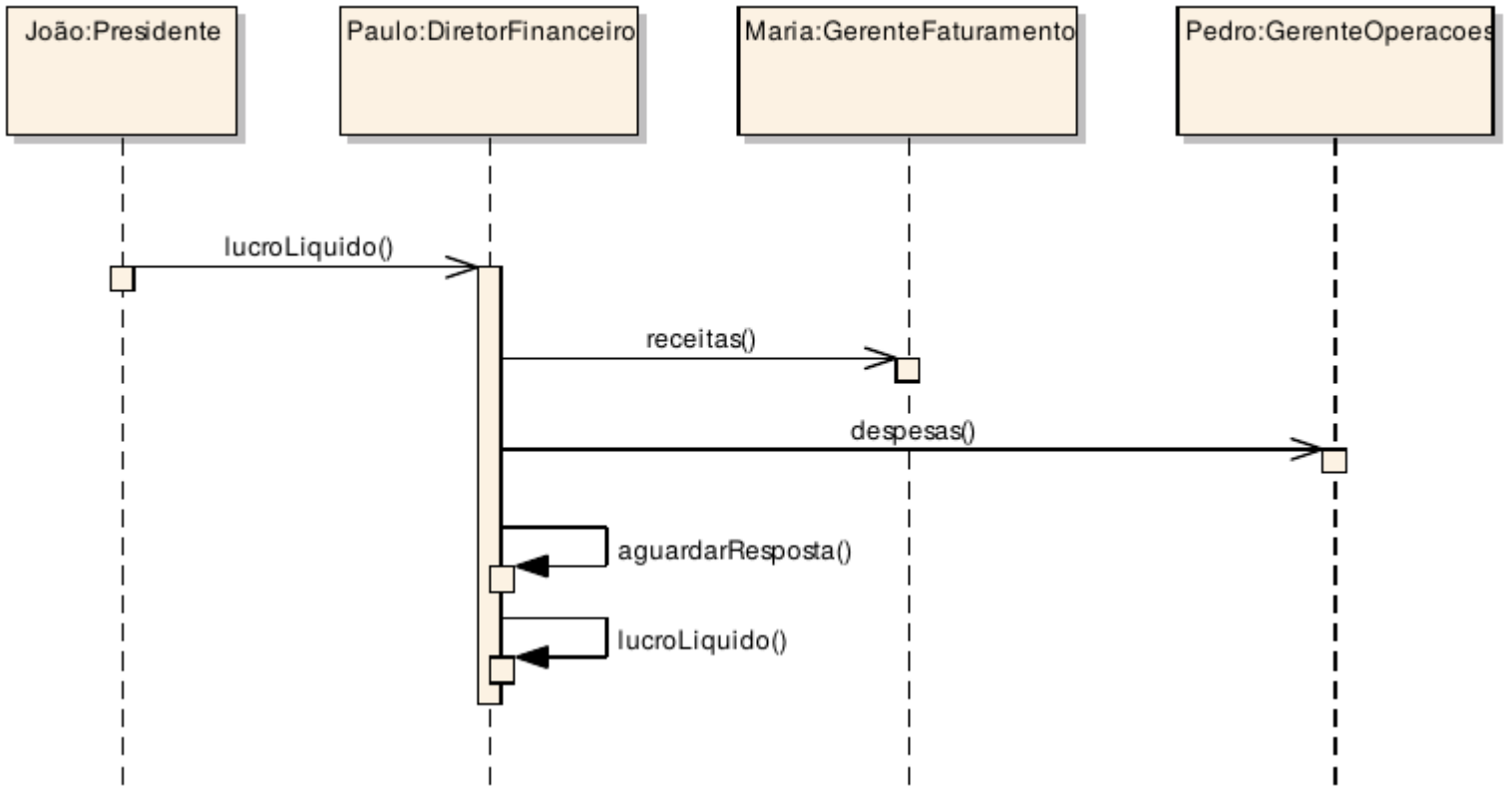


Diagrama de Sequência

- Representando repetições e condições

Operador	Significado
`alt`	Especifica múltiplos fragmentos alternativos em um quadro. A condição em que um fragmento é executado é colocada no topo do fragmento entre colchetes. Os fragmentos do quadro são separados por linhas tracejadas, com as condições em que são executados em seus topos entre colchetes.
`opt`	Especifica um quadro executado opcionalmente. O quadro corresponde a um único fragmento. A condição em que o fragmento é executado é colocada no topo do quadro, entre colchetes.
`par`	Especifica múltiplos fragmentos executados concorrentemente.
`loop`	Especifica um quadro (um único fragmento) executado iterativamente.
`ref`	Especifica um rótulo para um quadro, que pode ser referenciado em outro lugar ou em outro diagrama, como uma chamada.
`sd`	De "*sequence diagram*", que opcionalmente rotula um quadro que contém totalmente um diagrama de sequência.

Diagrama de Sequência

- Repetição

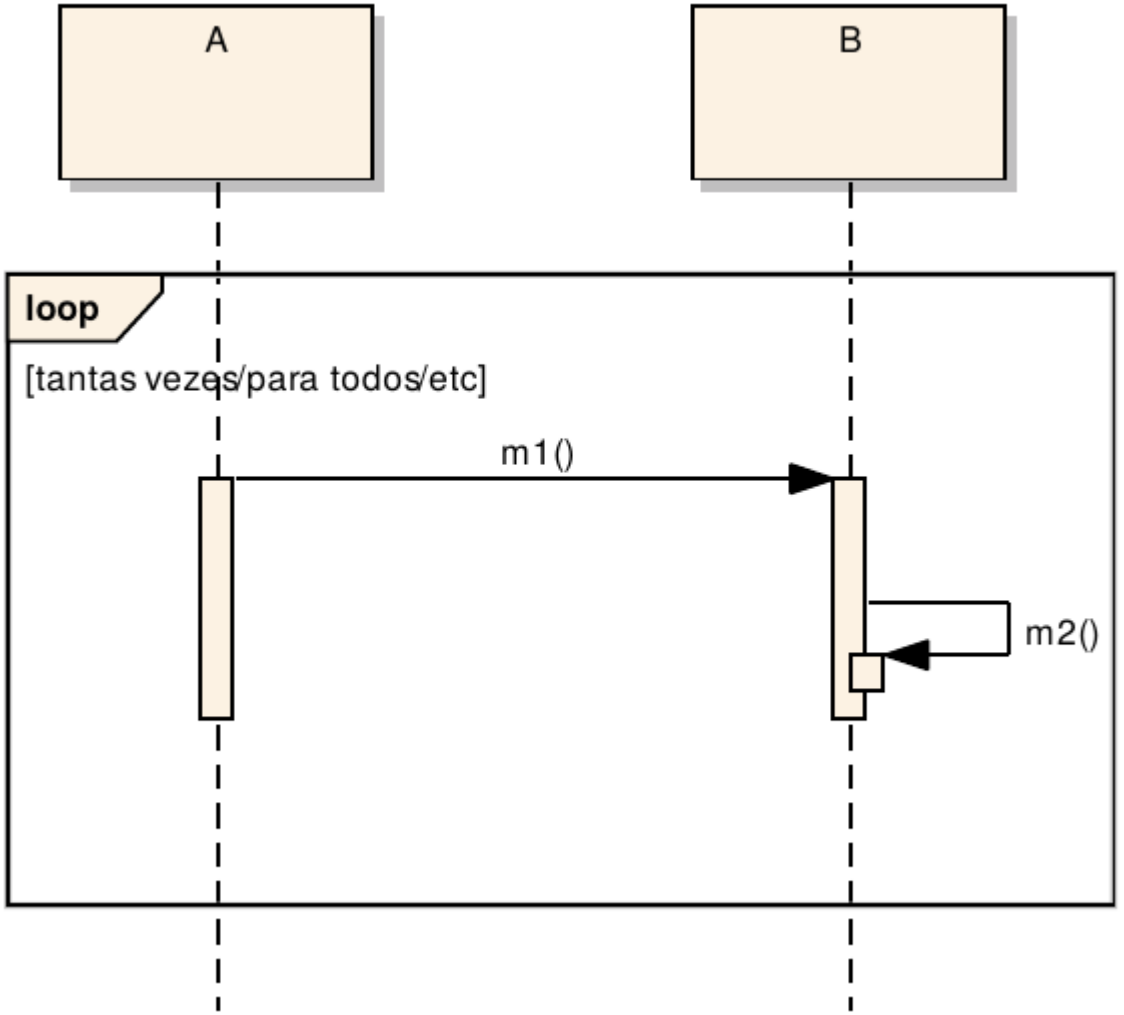


Diagrama de Sequência

- Se (if)

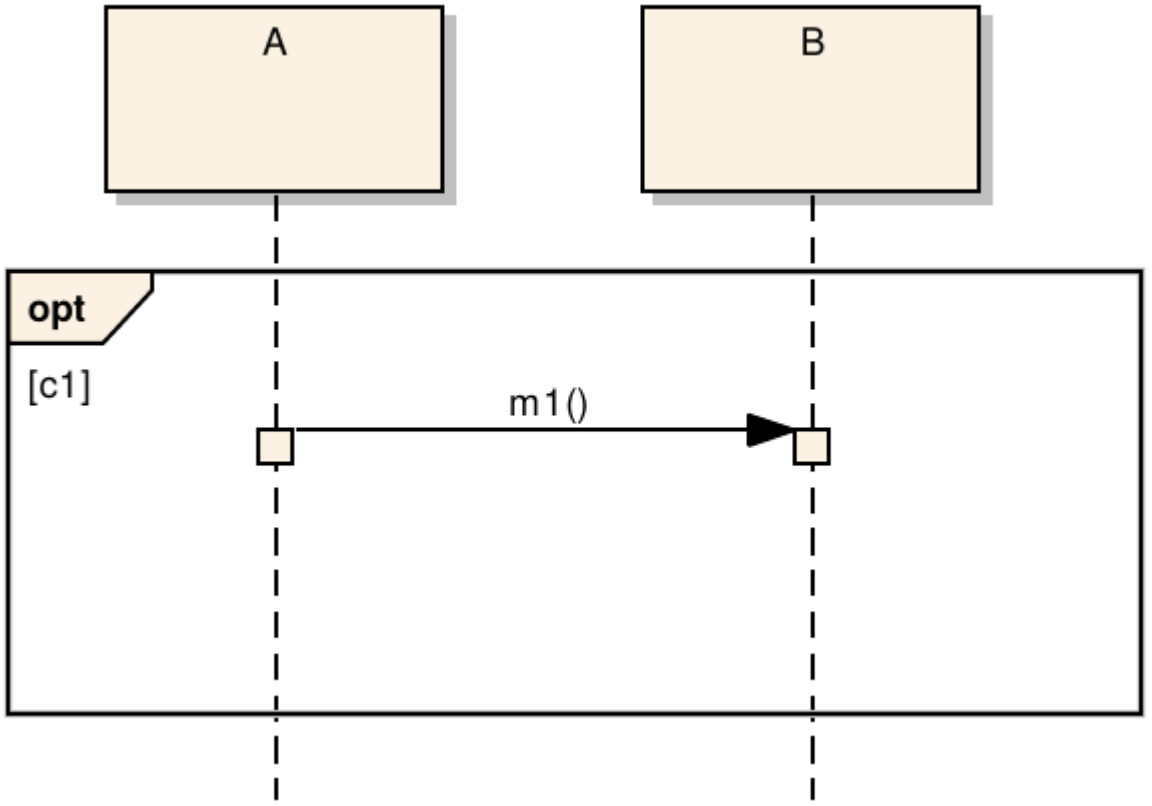


Diagrama de Sequência

- Se-senão (if-else)

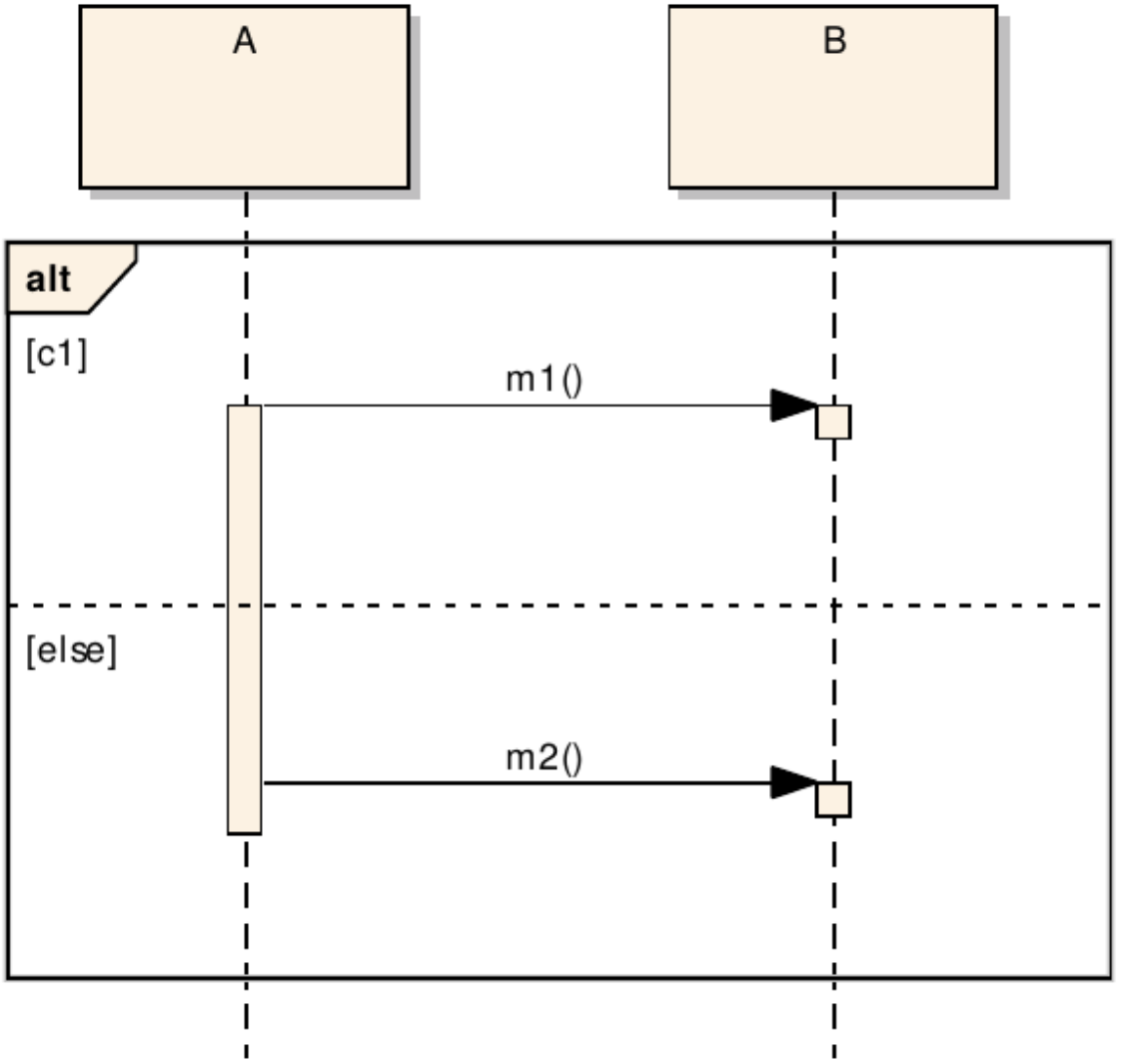
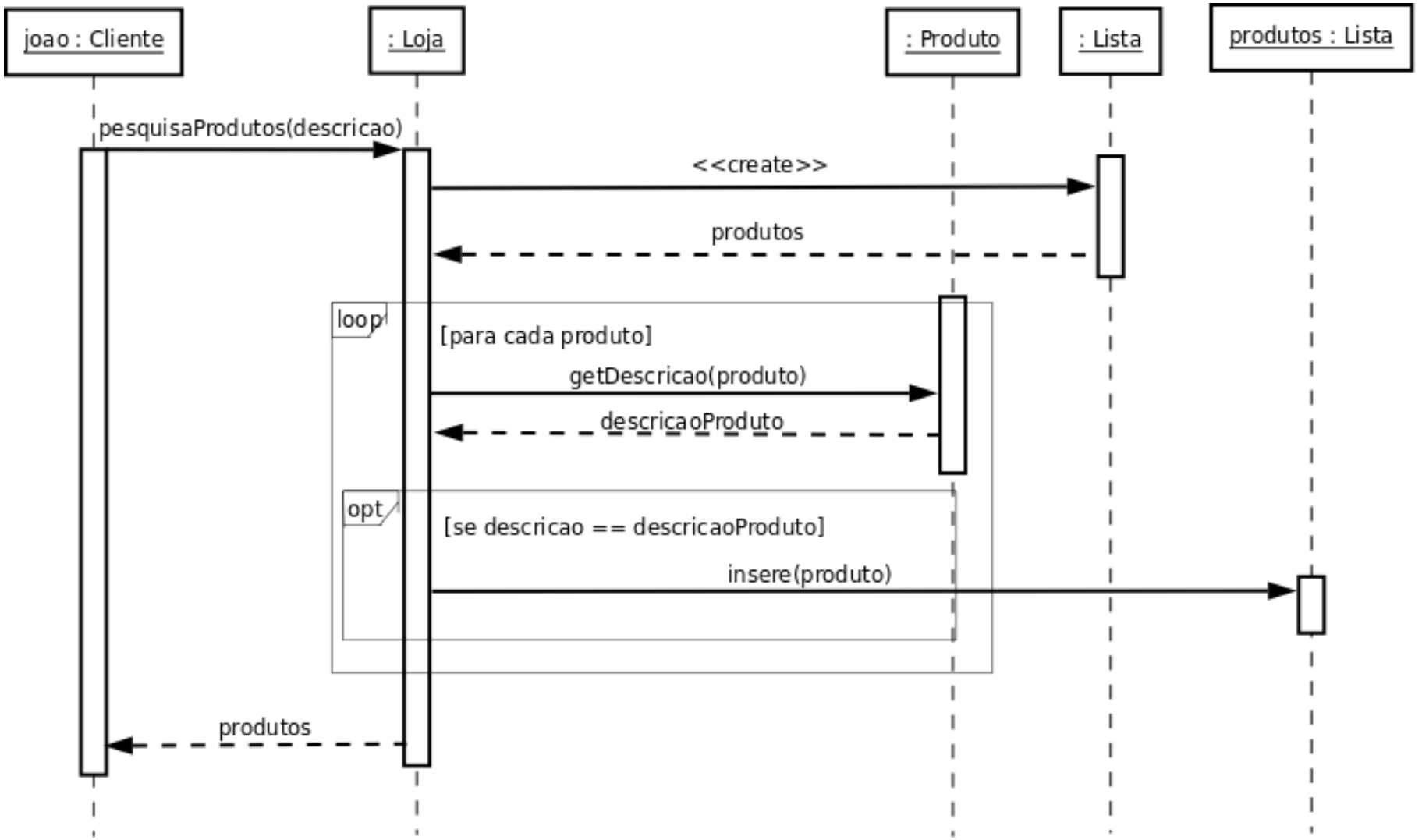


Diagrama de Sequência

- Exemplo



Referências

- Scott W. Ambler. The object primer 3rd edition: Agile Model-Driven Development with UML 2.0. 2004.
- Ana Cristina Melo. Desenvolvendo Aplicações com UML 2.2. 3ª edição. 2010.
- https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/
- https://ecs.wgtn.ac.nz/foswiki/pub/Events/MODELS2011/Material/MODELS_2011_T2-Roques-SysML_UML2.pdf

Obrigado!