Análise e Desenvolvimento de Sistemas
Engenharia de Software I

Revisão de Análise Orientada a Objetos

Parte II

Slides da disciplina Laboratório de Engenharia de Software
Curso de Análise e Desenvolvimento de Sistemas
Faculdade de Tecnologia de São Caetano do Sul – FATEC SCS
Prof. Marco Antonio Furlan de Souza
marco.furlandesouza@fatec.sp.gov.br

Agenda

- Diagrama de classes
- Diagrama de pacotes
- Diagrama de sequência
- Diagrama de atividade

Agenda

- Diagrama de classes
- Diagrama de pacotes
- Diagrama de sequência
- Diagrama de atividade

Diagrama de classes

- **Representação de classes**
  - O **símbolo UML** para **classes** é uma **caixa** com **três compartimentos**:

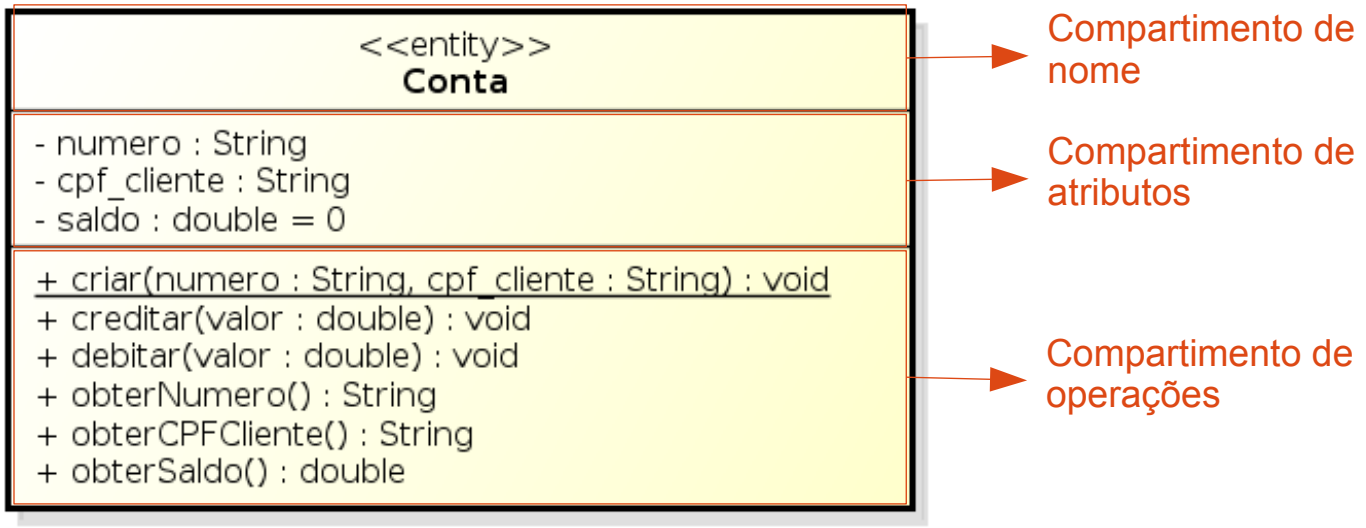


Diagrama de classes

- **Representação de objetos**
  - O **símbolo UML** para **objetos** é uma **caixa** com **dois compartimentos**:
    - No **compartimento** superior escreve-se o **identificador** do **objeto**, que é sempre **sublinhado**;
    - No **segundo compartimento**, **escrevem**-se os **nomes** dos **atributos** no **formato** `nome:tipo[multiplicidade]=valor`;
    - O **nome** do **atributo** é **obrigatório**. A **multiplicidade indica** que o **atributo** é na realidade uma **coleção** de valores.

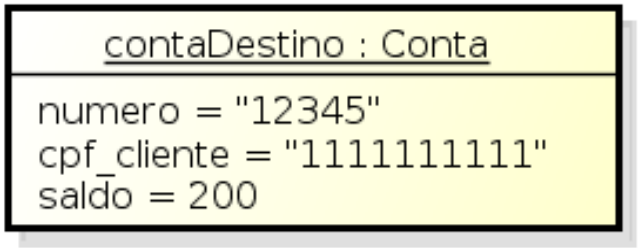


Diagrama de classes

- **Representação de classes**
  - **Compartimento de nome**
    - **Exibe** o **nome** da **classe** (no **exemplo**, `Conta`) e **opcionalmente** seu **estereótipo**, que é um **classificador adicional**;
    - **Normalmente** o **nome** da **classe** é um **substantivo** escrito com a **primeira letra** em **maiúscula**;
    - O **nome** do **estereótipo** é **livre**, mas **nomes como** `<<entity>>` (classifica a **classe** como de **entidade**), `<<boundary>>` (**classe** de **fronteira**) e `<<control>>` (**classe** de **controle**) são **muito utilizados** na **análise**;
    - **Classes abstratas** têm seu **nome escrito** em **itálico**.

Diagrama de classes

- **Representação de classes**
  - **Classes de controle** (JACOBSON et al., 1992)
    - Seus objetos **agem** como "**cola**" entre **objetos** dos **outros tipos**, **evitando** que se **particularize** o **controle** do **comportamento** do **sistema**, **embutindo-o tanto** em **objetos** do tipo **entidade** quanto de **interface**;
    - São anotadas com esterótipo `<<control>>` e possuem uma representação icônica.

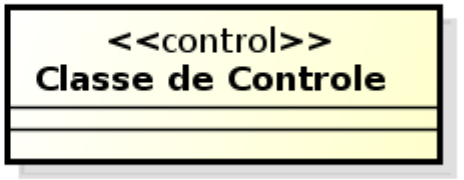
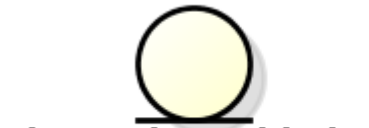


Diagrama de classes

- **Representação de classes**
  - **Classes de entidade** (JACOBSON et al., 1992)
    - Seus **objetos modelam** a **informação** que o sistema deverá tratar;
    - São **dependentes** do **domínio** do problema e representam seus conceitos-chave;
    - São anotadas com esterótipo `<<entity>>` e possuem uma representação icônica.



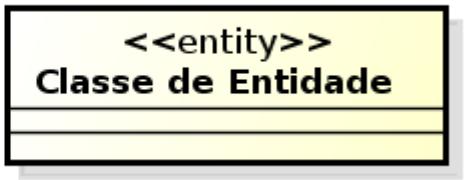


Diagrama de classes

- **Representação de classes**
  - **Classes de fronteira** (JACOBSON et al., 1992)
    - Os **objetos** de **fronteira recebem** toda **funcionalidade** dos **casos** de **uso** e **dependem diretamente** dependente do **ambiente** do **sistema**;
    - **Classes** de **fronteira** são **facilmente identificadas** nos casos de uso **por que são diretamente operadas** pelos **atores** do sistema;
    - São anotadas com esterótipo `<<boundary>>` e possuem uma representação icônica.

Diagrama de classes

- **Compartimento de atributos**
  - A **regra** para se **escrever atributos** é a **mesma** daquela **utilizada** para **objetos**, mas **prefixada** por um **adorno** (símbolo) que **representa** a **visibilidade** do **atributo**;
  - A **visibilidade** do **atributo**:
    - **Privado**: símbolo "–" (o mais comum, torna o **atributo acessível** apenas para **operações da classe** em que foi definido – princípio da **ocultação** da **informação**);
    - **Protegido**: símbolo "#" (o **atributo** é **acessível apenas** para **operações** da **classe** em que **foi definido** e **suas subclasses**);
    - **Público**: símbolo "+" (**sem restrição** de **acessibilidade – não comum**);
    - **Pacote**: símbolo "~" (**sem restrição** para as **classes presentes** em um **mesmo pacote**, mas **não acessíveis** ao **restante**).

Diagrama de classes

- **Compartimento de atributos**
  - Depois da definição do atributo, pode-se ainda acrescentar uma **restrição**, que é uma **expressão lógica** que **afeta** o **atributo** e é escrita entre chaves;
  - **Restrições podem seguir** a **OCL** (*Object Constraint Language*) da OMG, e está definida em padrão da UML[2];
  - Por **exemplo**, se o **saldo** de uma **conta** deve ser **sempre positivo**, pode-se **definir** o **atributo saldo assim**:

```
saldo: double {>0}
```

[2] http://www.omg.org/spec/OCL/

Diagrama de classes

- **Compartimento de operações**
  - As **operações** são **escritas** com os **seguintes elementos**, da **esquerda** para a **direita**:
    - **Adorno**:
      - **Símbolo** "+" representa uma **operação pública** (**mais comum**, pois torna a **operação acessível** em **operações** de **quaisquer objetos**),
      - **Símbolo** "#" representa uma **operação protegida** (**acessível apenas** para **operações desta classe** e suas **subclasses**);
      - **Símbolo** "–" representa uma operação **privada** (**acessível apenas** para **operações desta classe**);
      - **Símbolo** "~" uma **operação** de **pacote** (**sem restrição** para as **classes presentes** em um **mesmo pacote**, mas **não acessíveis** ao **restante**).

Diagrama de classes

- **Compartimento de operações**
  - As **operações** são **escritas** com os **seguintes elementos**, da **esquerda** para a **direita** (cont.):
    - **Nome da operação**: normalmente é um **verbo** que **identifica** seu **propósito** na **classe**;
    - **Lista** de **parâmetros**: entre **parênteses** se **escrevem** os **parâmetros** no **formato**:

      ```
      nome: tipo[multiplicidade] = valor_inicial
      ```

      São **separados** por **vírgula**. Se **não** houver **parâmetros**, basta **escrever parênteses vazios**;
    - **Tipo** de **retorno**: **após dois pontos** se **escreve** o **tipo** de **retorno**, se **existir**;
    - A **UML define apenas tipos primitivos** de **dados** (integer, boolean, string e real). **Outros tipos** de dados **podem** ser **adicionados** e **qualquer classe** do **modelo** é **automaticamente** um **tipo** de **dado** que pode ser **utilizado**.

Diagrama de classes

- **Compartimento de operações**
  - Pode-se **determinar** se os **parâmetros das operações serão utilizados apenas** para **entrada**, **saída** ou **entrada** e **saída**;
  - **Prefixam-se** os **parâmetros** com os **modificadores**:
    - **in**: parâmetro de apenas entrada;
    - **out**: parâmetro de apenas saída;
    - **inout**: parâmetro de entrada/saída.
  - Por **exemplo**, pode-se **reescrever** a **operação** `debitar()` assim, na **classe** `Conta`:

```
+debitar(in valor: double): void
```

Diagrama de classes

- **Exemplo**
  - Classes que poderiam ser obtidas do problema do **Caixa Eletrônico**:

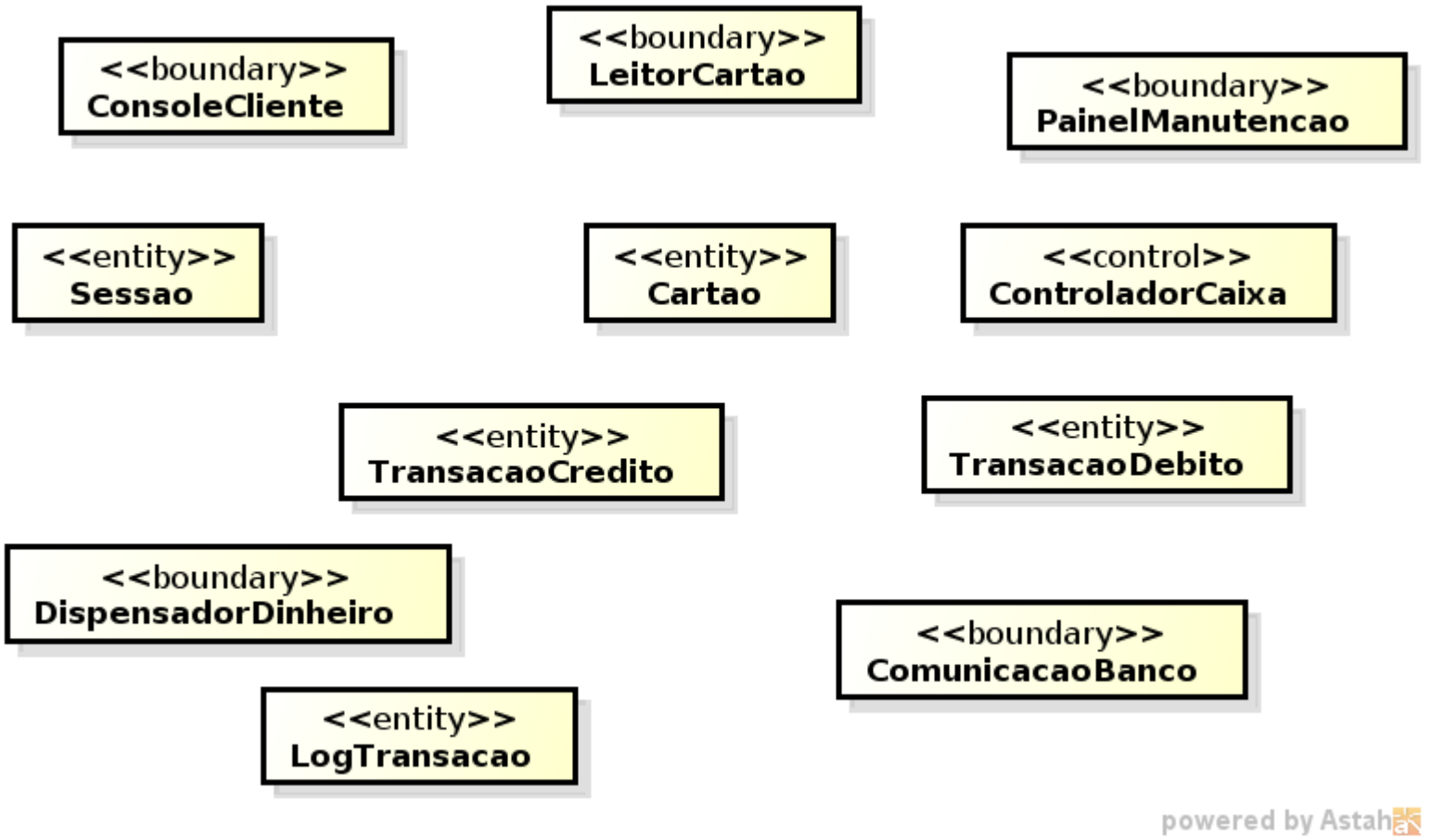


Diagrama de classes

- **Exemplo**
  - Classes que poderiam ser obtidas do problema do **Caixa Eletrônico** (versão estereotipada com ícones):

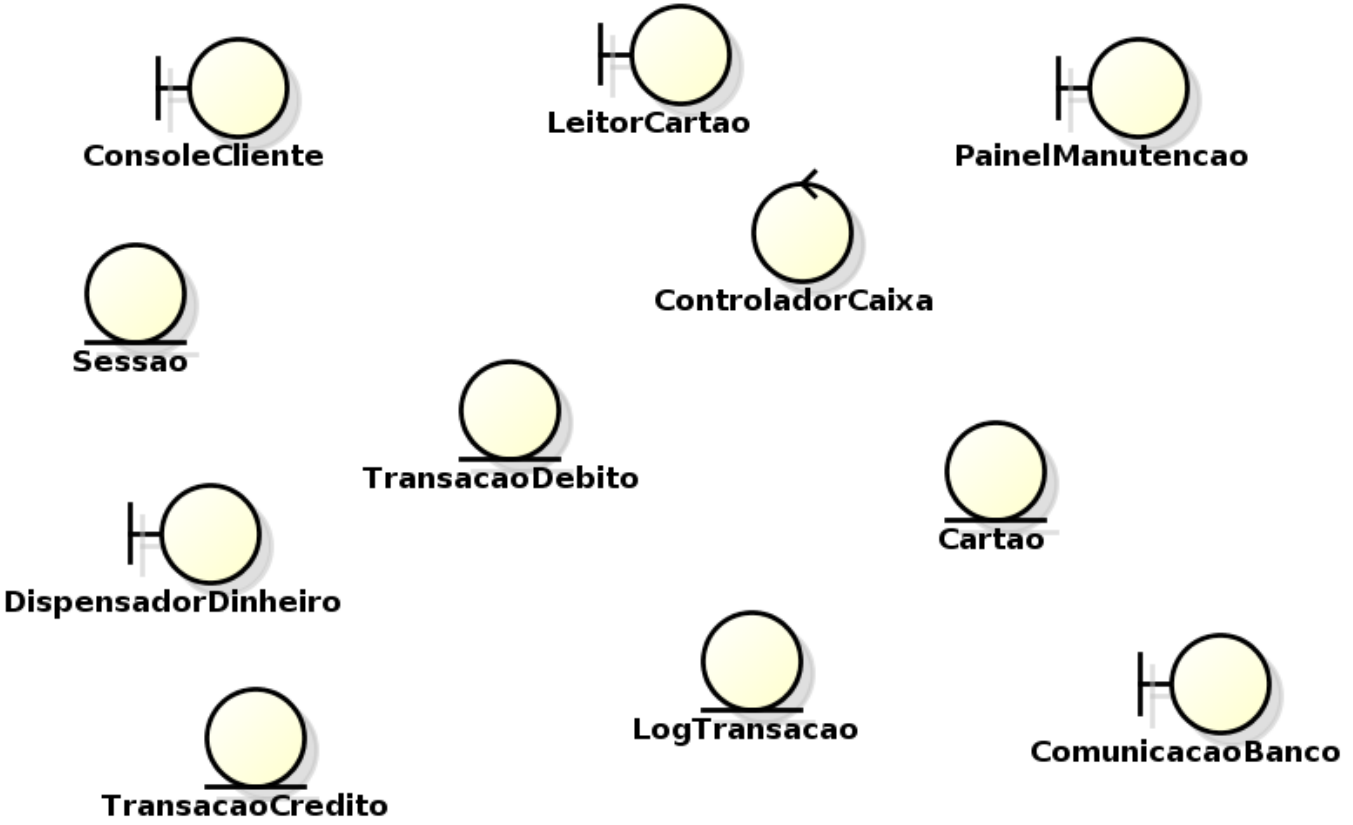


Diagrama de classes

- **Exemplo**
  - Algumas classes do sistema do **Caixa Eletrônico** com atributos e operações apresentados:

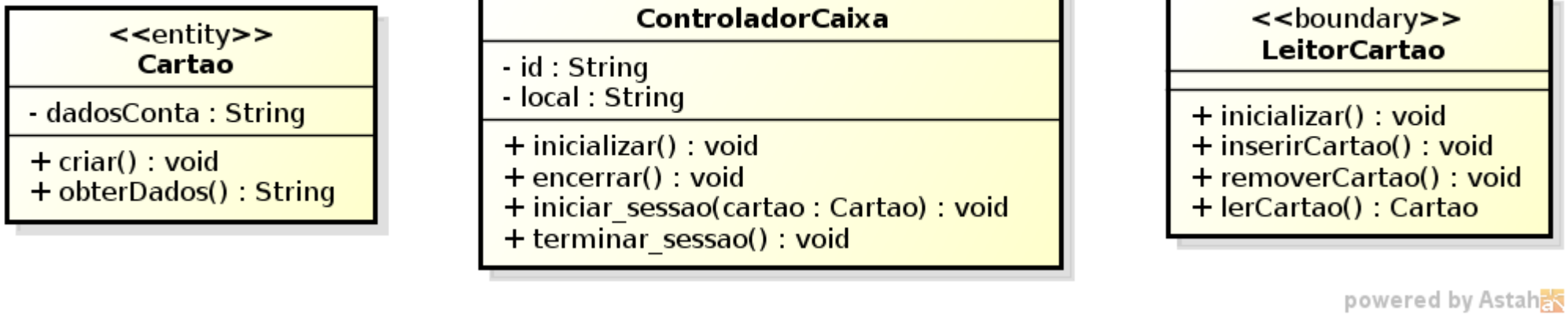


Diagrama de classes

- **Escopo**
  - Na **notação UML** para **classes**, pode-se **sinalizar**, ainda, o **escopo** dos **atributos** e **operações**;
  - Entende-se por **escopo** o **lugar** no qual um **atributo** ou **operação** está **definido**:
    - **Escopo de instância** (**padrão**): **cada objeto distinto** possui uma **própria cópia** de seu **atributo** e que no **caso** de **operações significa que** a **operação** pode **manipular** os **atributos** de **cada objeto** que a **invocou**;
    - **Escopo de classe**: no **caso** de **atributos significa** que o **atributo** em **questão** será **global** à **classe**, com um **único valor compartilhado** por todos seus **objetos** e que no **caso** de **operações** significa dizer que a **operação** é **global** à **classe**, **não estando ligada** a um **objeto particular** e **não** sendo **possível acessar** seus **atributos diretamente**.

Diagrama de classes

- **Escopo**
  - Na UML, **atributos** e **operações** com **escopo** de **classe** são **sublinhados**;
  - **Exemplo**:

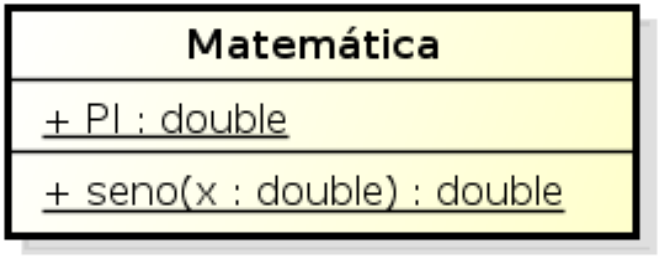


Diagrama de classes

- **Relacionamentos**
  - **Relacionamentos** são **ligações** que **possuem** um **significado** entre os **elementos** de um **modelo**;
  - É o **modo UML** de **conectar** “**coisas**”;
  - Em um **sistema orientado** a **objetos**, os **objetos** não **operam sozinhos**, em **isolamento**;
  - Eles **precisam** se **conectar** para **realizar** um **trabalho útil** em **benefício** aos **usuários** do **sistema**;
  - **Conexões** entre os **objetos** são **denominadas** de **ligações**, e quando os **objetos trabalham** em **conjunto**, diz-se que eles **colaboram entre si**.

Diagrama de classes

- **Relacionamentos**
  - **Se há** uma **ligação** entre **dois objetos então também deve haver** alguma **ligação semântica** entre suas **classes**;
  - **Para** que os **objetos** possam se **comunicar diretamente** uns com os **outros**, as **classes desses objetos** devem ter **conhecimentos entre si**, de alguma forma;
  - Essas **conexões** entre as **classes** são **denominadas** de **associações**;
  - Então, as **ligações** entre **objetos** são, na verdade, **instâncias** de **associações** entre suas **classes**.

Diagrama de classes

- **Relacionamentos**
  - O **significado** de uma **associação** é **simples** – ela **indica** que **existirão ligações** entre **objetos** das **classes associadas**:

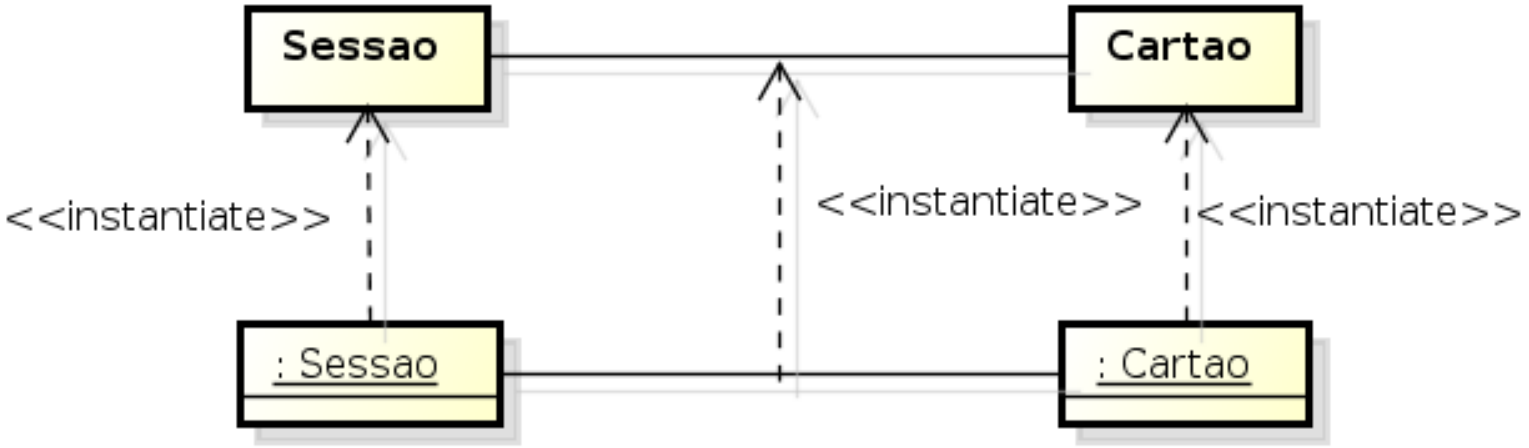


Diagrama de classes

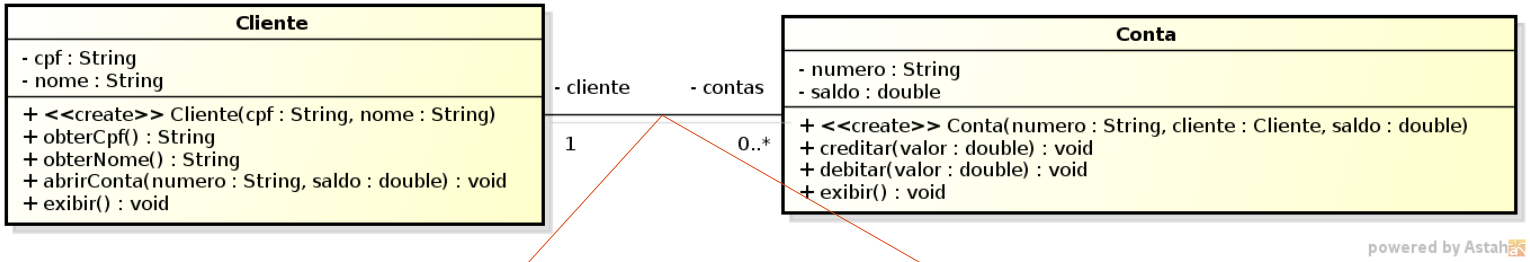
- **Ligações entre objetos**
  - Um **programa orientado** a **objetos** é o **resultado** da **execução** de uma **comunidade** de **objetos cooperando entre si**;
  - Uma **ligação** é uma **conexão semântica entre dois** objetos que **permite** que as **mensagens sejam enviadas** de um **objeto** para o outro;
  - Em um **sistema orientado** a **objetos**, durante a **execução** muitos **objetos aparecem** e **desaparecem** e **as ligações** que **unem** esses objetos **também**;
  - As **mensagens** (**execução** de **operações**) são **passadas** entre **objetos** sobre essas **ligações**. Quando um **objeto** recebe uma **mensagem**, ele **executa** uma **operação correspondente**.

Diagrama de classes

- **Ligações entre objetos**
  - Pode-se **implementar** as **ligações** de **diversas formas**, nas **linguagens orientadas** a **objetos**;
  - Em **Java**, **ligações** são **referências** de **objeto** e **em C++** são **ponteiros**, **referências**, ou pela **inclusão direta** de um **objeto** por **outro**;
  - Um **requisito mínimo** para uma **ligação existir** é que, **no mínimo**, **um dos objetos deva ter** uma **referência para** outro **objeto**.

Diagrama de classes

- **Associações e ligações**
  - **Exemplo** em Java




```java
package banco;

import java.util.ArrayList;
import java.util.List;

public class Cliente {
        private List<Conta> contas;
        String cpf;
        String nome;

        public Cliente(String cpf, String nome) {
                this.cpf = cpf;
                this.nome = nome;
                this.contas = new ArrayList<Conta>();
        }

...
```

```java
package banco;

public class Conta {
        private String numero;
        private double saldo;
        private Cliente cliente;

        public Conta(String numero, Cliente
                        cliente, double saldo) {
                this.numero = numero;
                this.cliente = cliente;
                this.saldo = saldo;
        }

...
```

Diagrama de classes

- **Associações e ligações**
  - **Exemplo** em Java

```
Cliente c = new Cliente("1234567890", "João Carlos");
c.abrirConta("0001", 1000);
c.abrirConta("0002", 2000);
c.exibir();
```

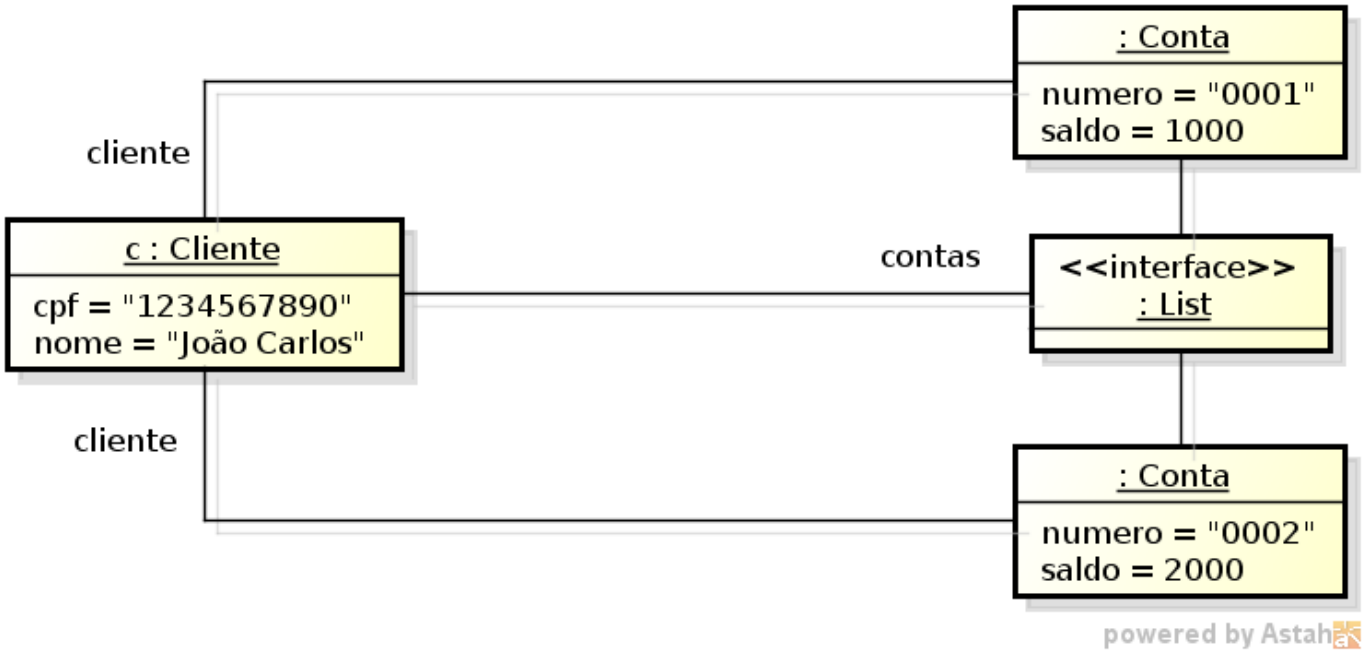


Diagrama de classes

- **Representação de objetos e ligações**
  - Permite exibir **objetos** e suas **conexões** em um certo **ponto** no **tempo**;
  - É “um **instantâneo**” de **parte** de um **sistema orientado** a **objetos** em **execução**, **explicando** os **objetos** e as **ligações** entre **eles**.
  - As **ligações** são representadas por **linhas simples** entre dois objetos;
  - Esta **ligação** pode ser **anotada** com o **nome** de **um papel** que o **objeto possui** naquele **instante**.

Diagrama de classes

- **Representação de objetos e ligações**
  - **Exemplo**

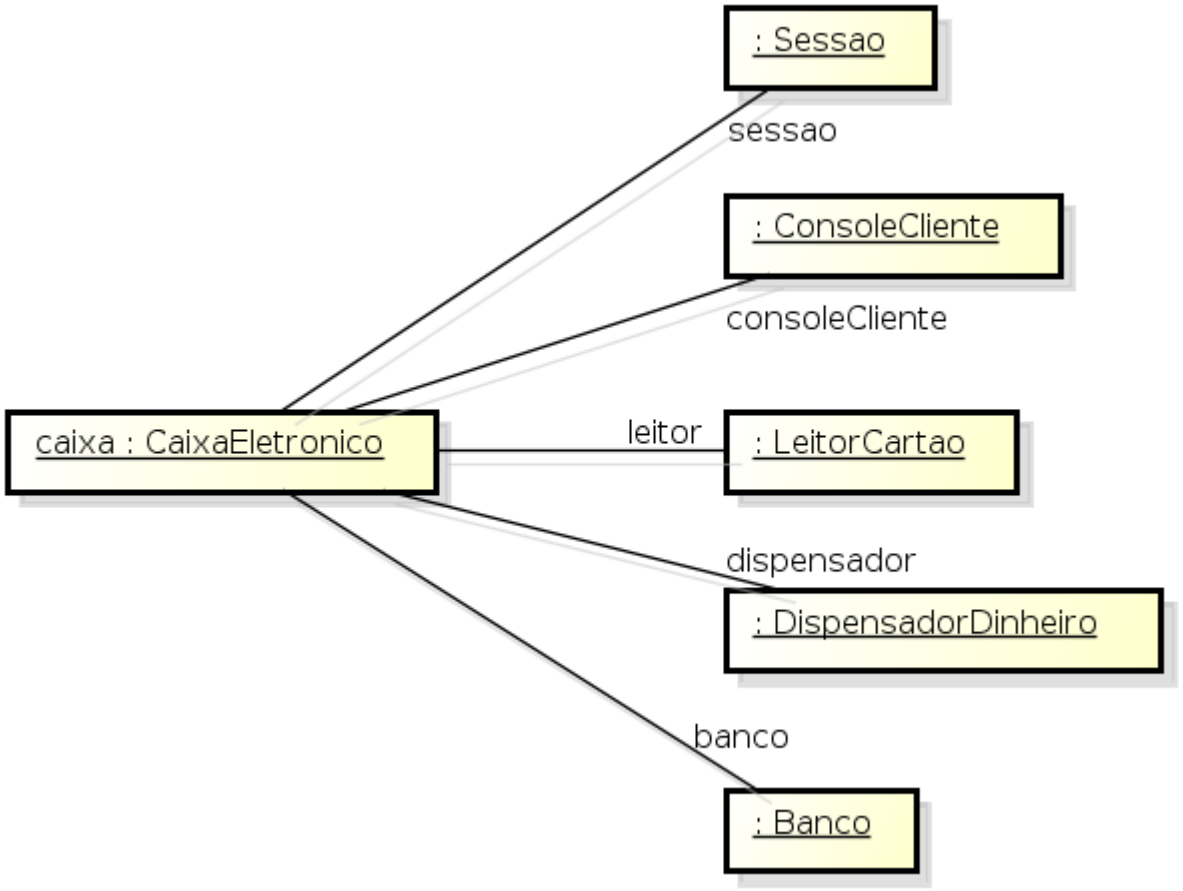


Diagrama de classes

- **Representação de classes e relacionamentos**
  - É utilizada para **exibir classes** e **seus relacionamentos** na **visão lógica** de um **sistema**;
  - **Representa** a **estrutura** de **classes** de um **sistema**;
  - Na **Análise**, **indicam** os **papéis comuns** e as **responsabilidades** das **entidades responsáveis** pelo **comportamento** do **sistema**;
  - No **Projeto**, são **utilizados** para **capturar** a **estrutura** das **classes** que formam a **arquitetura** do **sistema**;
  - Existem **diversos tipos** de **relacionamentos** entre classes, **além** da **associação**.

Diagrama de classes

- **Relacionamento de associação**
  - **Elementos**
    - **Nome**: são **verbos** que **indicam** uma **ação** que o **objeto** de **origem** está **realizando** sobre o **objeto alvo**.
    - O **nome** também **pode** ser **prefixado** ou **pós-fixado** com **uma** pequena **seta preta** para indicar a **direção** em que o **nome** de **associação deve** ser **lido**.

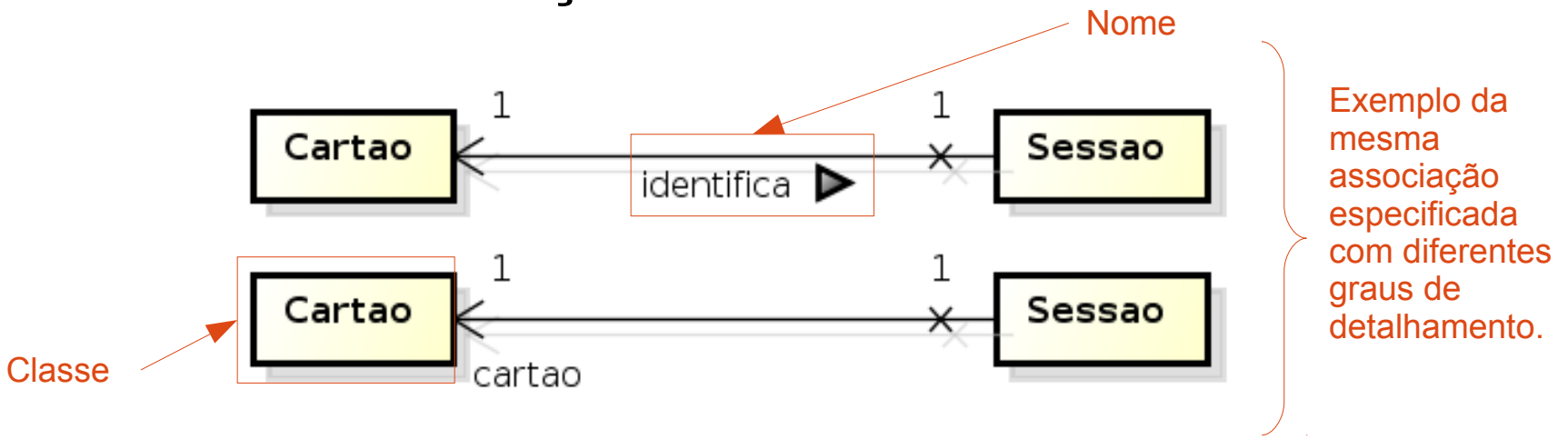


Diagrama de classes

- **Relacionamento de associação**
  - **Elementos**
    - **Nomes de papéis**: nomes de papéis **representam** a **função** que **objetos** das classes **envolvidas** na **associação** deverão **representar** no **sistema**;
    - O nome do papel **pode** ser **prefixado** pelo **símbolo** de **visibilidade** (“-”, “+”, “#”, “~”) já visto anteriormente para se **denotar** que ele **futuramente** será um **atributo** do objeto **associado**.

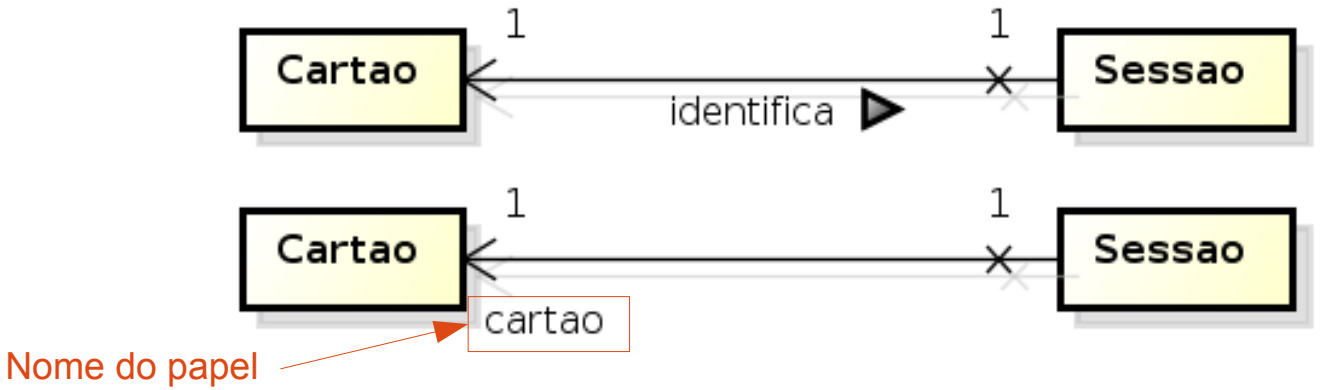


Diagrama de classes

- **Relacionamento de associação**
  - **Elementos**
    - **Multiplicidade**: **restringe** a **ligação** entre **objetos** pois **indica** uma **quantidade exata ou** um **intervalo** que **especifica** a **mínima** e a **máxima quantidade** de **objetos** que **estarão ligados**.
    - É um símbolo **escrito como** um **número fixo** (0, 1 etc) ou o **símbolo** "*" que **indica** "**vários**" (zero ou mais) **ou ainda** na **forma** `mínimo..máximo` onde `mínimo` é a mínima multiplicidade exigida e `máximo` é a máxima multiplicidade exigida.

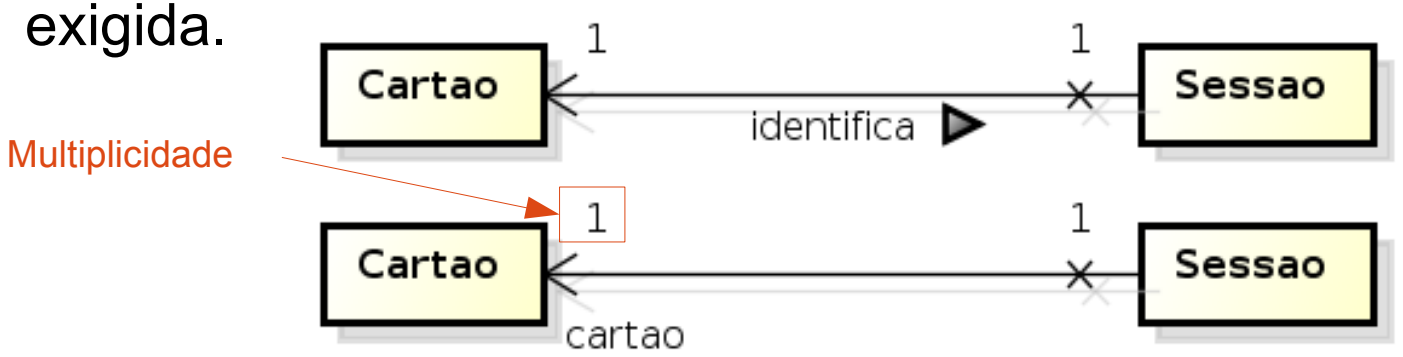


Diagrama de classes

- **Relacionamento de associação**
  - **Elementos**
    - **Navegabilidade**: a navegabilidade **indica** a **direção** que as **mensagens fluirão**. **Pode ser não definida**, para **esquerda**, para **direita**, **ambos** ou **nenhuma**.

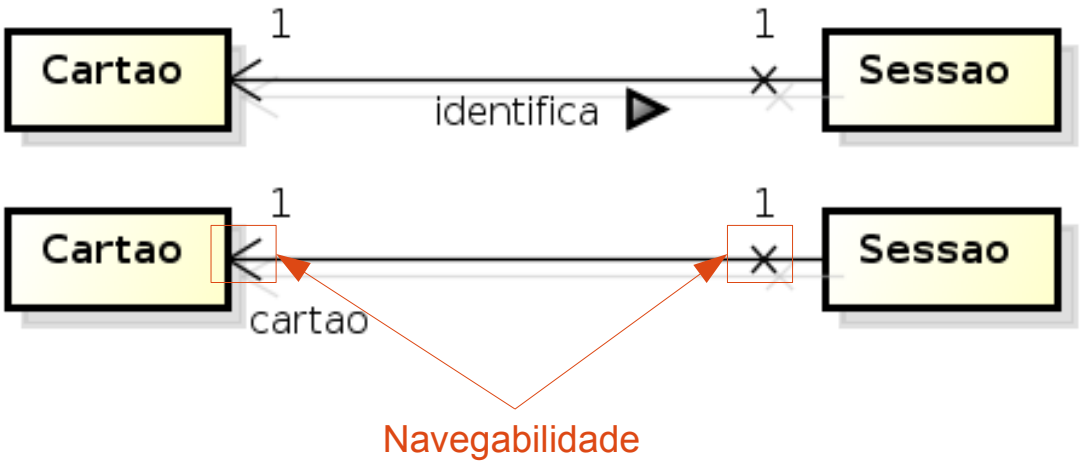


Diagrama de classes

- **Relacionamento de associação**
  - **Atributos de uma associação**
    - Existem **situações** nas quais é **necessário qualificar** com **atributos** uma **associação**;
    - Por **exemplo**, **representar** que uma **sessão** está **associada** a um **cartão** em um dado **instante** e que essa **associação** foi **autorizada** pelo **banco** por um **número** de **autorização**:

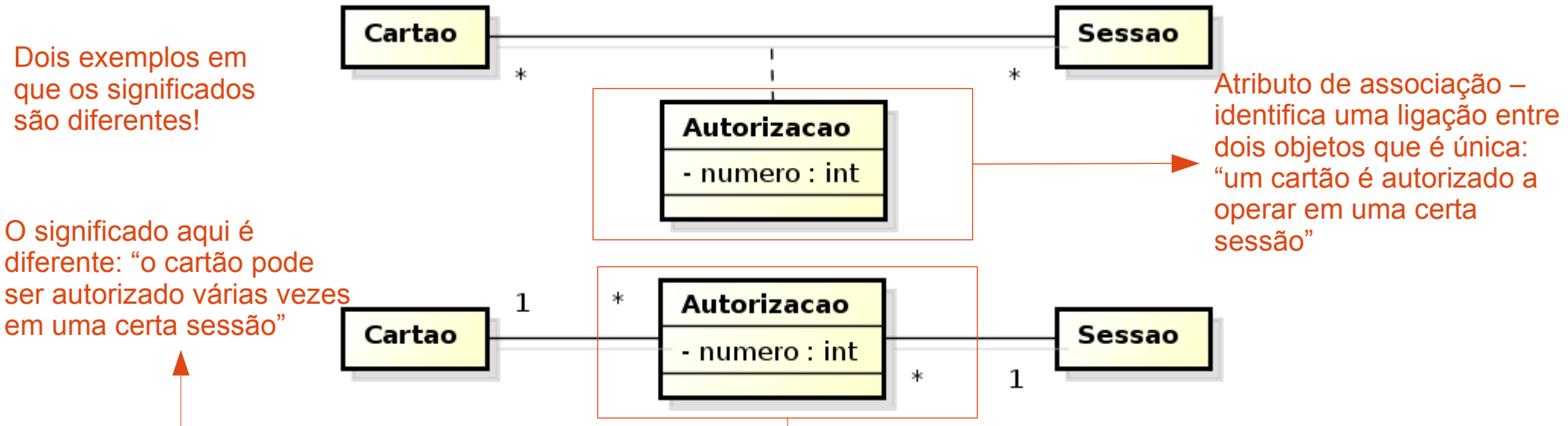


Diagrama de classes

- **Relacionamento de associação**
  - **Associações reflexivas**
    - Quando uma **associação existe** entre o **mesmo elemento**, diz-se que se tem uma **associação reflexiva**;
    - Por **exemplo**, o conceito de **sistema** de **arquivos** onde uma **pasta** pode **conter** outras **pastas**, poderia ser assim modelado (o exemplo descreve uma **hierarquia**):

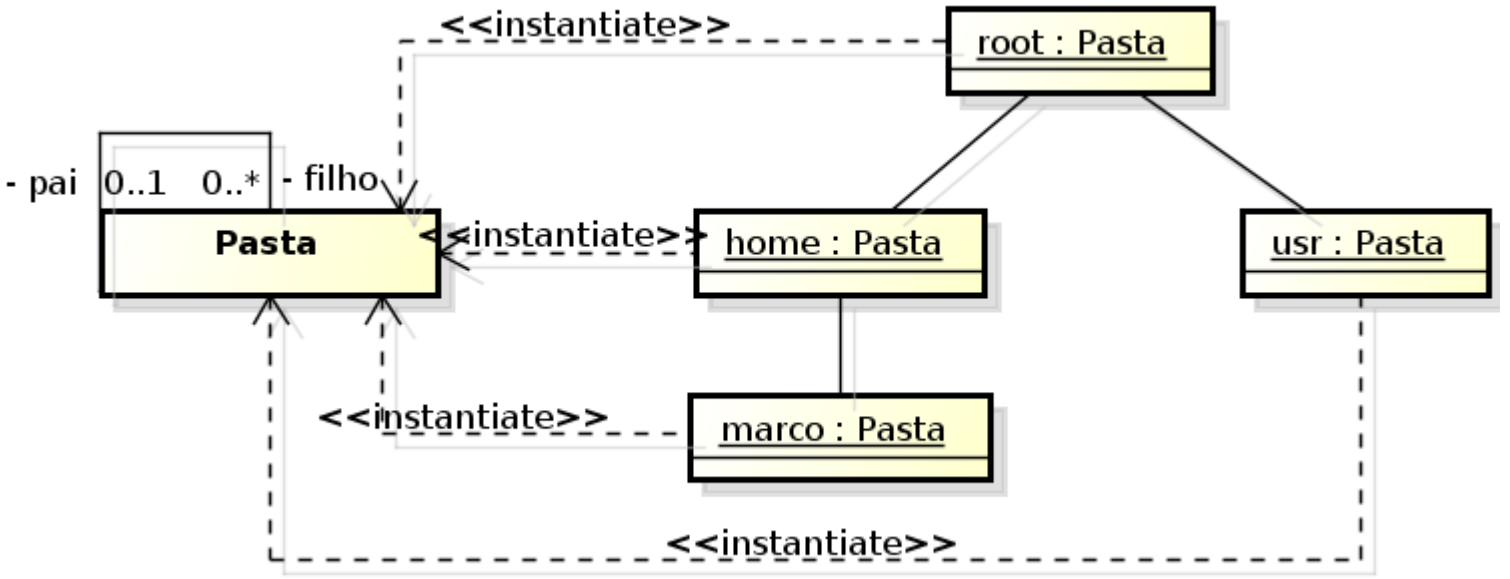


Diagrama de classes

- **Relacionamento de associação**
  - **Associações reflexivas**
    - Por **exemplo**, o conceito de **funcionários** que podem ser **subordinados** a um ou mais **funcionários** (neste caso, tem-se uma **rede**):

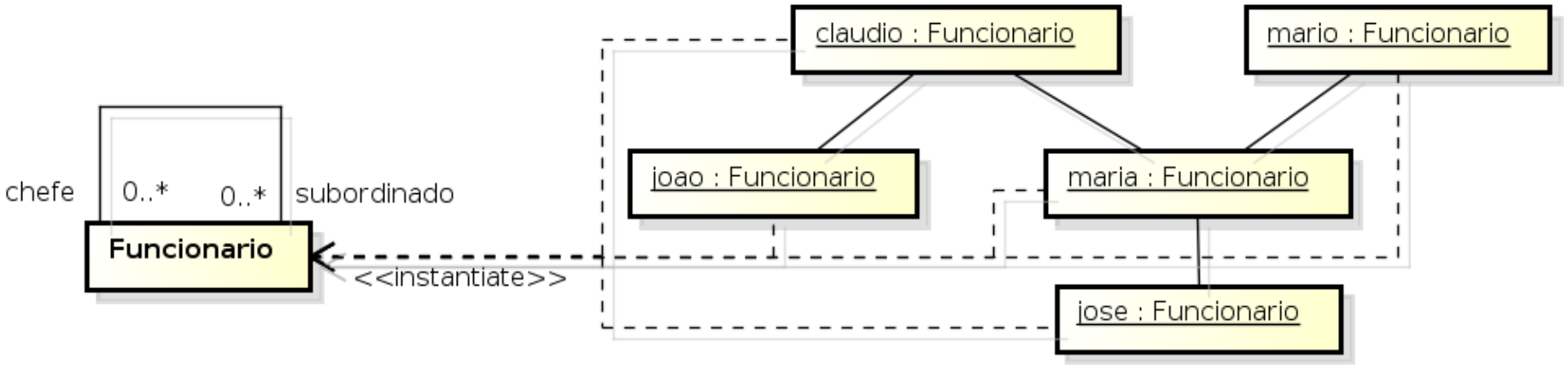


Diagrama de classes

- **Relacionamento de agregação e de composição**
  - **Conceitos**
    - São **refinamentos** de **associações** e **normalmente** são **considerados** em **Projeto Orientada** a **Objetos**;
    - Quando uma **associação** tem um **caráter mais para** "**todo-parte**" **do que** uma **associação** mais "**equilibrada**", pode-se **pensar** no **conceito** de **agregação**;
    - Na **agregação**, o "**todo**" **contém uma** ou **mais** "**partes**", sendo que tanto a **linha** de **vida** do **objeto** "**todo**" quanto as **linhas** de **vida** de suas "**partes**" **podem ser independentes**;
    - A **composição** tem a **mesma definição** que a **agregação**, mas a **linha** de **vida** das "**partes**" é **controlada** pela **linha** de **vida** do **todo**, ou seja, a **parte não existe sem** o **todo**.

Diagrama de classes

- **Relacionamento de agregação e de composição**
  - **Exemplos**
    - Um **cesto** de **compra** é um **agregado** de **produtos**: **tanto** o **cesto** quanto os **produtos** são **independentes**;
    - Já a uma **venda** é um **agregado** de **itens** de **venda**, onde cada **item** de **venda** pode estar **associado** a um **produto** . Observe que um **item** de **venda não** existe **sem** a **venda** logo, **trata**-se de **uma composição**.

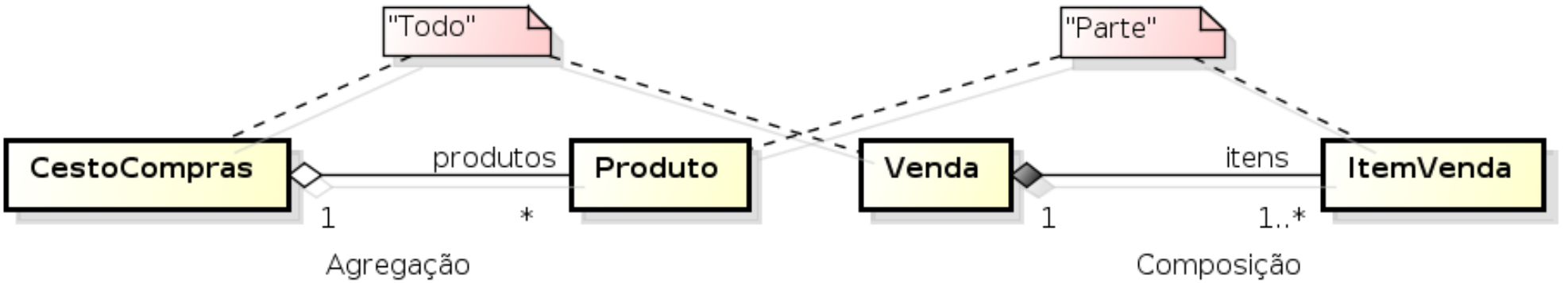


Diagrama de classes

- **Relacionamento de dependência**
  - **Conceitos**
    - Uma **dependência** indica uma **relação** entre **dois** ou **mais elementos** do modelo em que uma **mudança** em um **elemento** (o **fornecedor**) pode **afetar** ou **fornecer informações** necessárias para o outro **elemento** (o **cliente**);
    - **Dependências** são **utilizadas** para **modelar relações** entre **classificadores**, onde um **classificador** (**cliente**) **depende** de algum **modo** de **outro** (**fornecedor**), mas a **relação não** é **verdadeiramente** uma **associação** ou **generalização**.
    - **Dependências** podem ser **utilizadas não apenas** entre **classes**, mas **também** entre **pacotes** e entre **objetos** e **classes**.

Diagrama de classes

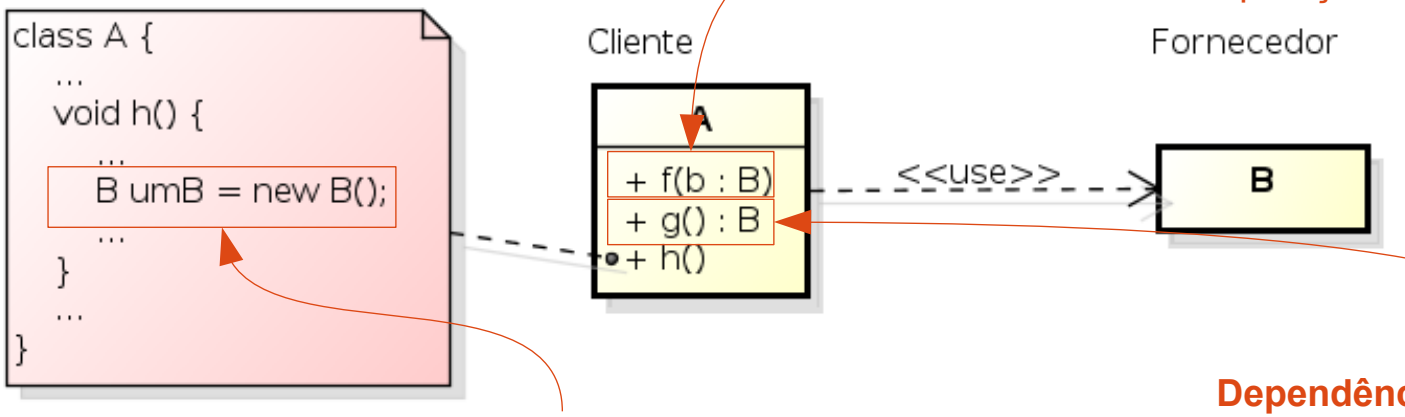
- **Relacionamento de dependência**
  - **Três tipos básicos de dependência**
    - **Uso**: o **cliente utiliza** alguns dos **serviços** disponibilizados pelo **fornecedor** para **implementar** seu próprio **comportamento**.
      - Existem **cinco tipos** de **dependência** de **uso**, identificadas pelos **estereótipos** `<<use>>`, `<<call>>`, `<<parameter>>`, `<<send>>` **e** `<<instantiate>>`;
    - **Abstração**: **indica** uma **relação** entre **cliente** e **fornecedor**, onde o **fornecedor** é mais **abstrato** do que o **cliente**, isto é, em um **ponto** de **desenvolvimento diferente** do **cliente**.
      - Existem **quatro tipos** de **dependência** de **abstração**, identificadas pelos estereótipos `<<trace>>`, `<<substitute>>`, `<<refine>>` **e** `<<derive>>`;

Diagrama de classes

- **Relacionamento de dependência**
  - **Três tipos básicos de dependência**
    - **Permissão**: o **fornecedor concede algum tipo** de **permissão** para o **cliente** para **acessar** seu **conteúdo**. Esta é uma **forma** do **fornecedor controlar** e **limitar** o **acesso** ao seu **conteúdo**.
      - Existem **três tipos** de **dependência** de **permissão**, identificadas pelos **estereótipos** `<<access>>`, `<<import>>` e `<<permit>>`.

Diagrama de classes

- **Relacionamento de dependência**
  - **Dependência de uso tipo <<use>>**
    - A **relação** de **dependência** de uso o tipo <<use>> é a mais **comum** de se utilizar na **modelagem**. **Três situações** de **dependência de uso** <<use>> de uma **classe A** para uma **classe B**:



Dependência por parâmetro: um objeto da classe A depende de um objeto da classe B que é passado como parâmetro em uma ou mais de suas operações.

Dependência por variável local: um objeto da classe A depende de um objeto da classe B que é criado internamente em uma ou mais de suas operações.

Dependência por retorno: um objeto da classe A depende de um objeto da classe B que é retornado por uma ou mais de suas operações.

Diagrama de classes

- **Relacionamento de herança**
  - Para se entender o conceito de herança é **importante entender** o que significa **generalização**;
  - **Generalização** é uma **relação** entre um **elemento** mais **geral** e um elemento mais **específico**, em que o elemento **mais específico** é inteiramente **consistente** com o elemento **mais geral**, mas **contém mais informação**;
  - Quando há generalização, os **dois elementos** devem **obedecer** ao **princípio** da **substituição** – pode-se **utilizar** o **elemento** mais **específico** em **qualquer lugar** onde o **elemento** mais **geral** é **esperado**, sem danificar o sistema;
  - A **generalização** é um tipo **mais forte** do que a **associação** e **implica** uma **maior dependência** (portanto, **acoplamento**) entre os dois elementos envolvidos.

Diagrama de classes

- **Relacionamento de herança**
  - Na UML a **generalização** se aplica a **todos** os **classificadores**, (por exemplo, com casos de uso e atores);
  - **Exemplo**
    - A classe `Forma`, que faz o papel de uma **classe mais geral**, a partir da qual **pode-se criar classes** mais **específicas**, denominadas de **classes-filha**, **subclasses** ou **descendentes**, formando uma **hierarquia** entre as classes:

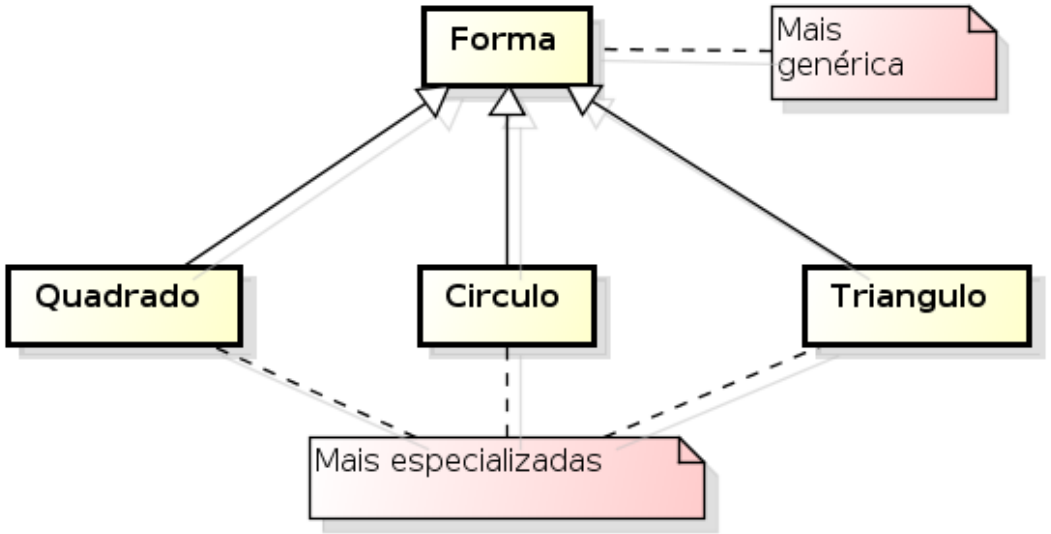


Diagrama de classes

- **Relacionamento de herança**
  - Quando se organiza as classes em uma **hierarquia** de **generalização**, **implicitamente** há uma **herança** entre as **classes**, de modo que as **subclasses herdam** todas as **características** de suas **superclasses**, que são:
    - **Atributos**;
    - **Operações**;
    - **Relacionamentos**;
    - **Restrições**.

Diagrama de classes

- **Relacionamento de herança**
  - **Conceitos**
    - As **subclasses** também **podem adicionar novos atributos** e **operações** e também **substituir operações** da **superclasse**:

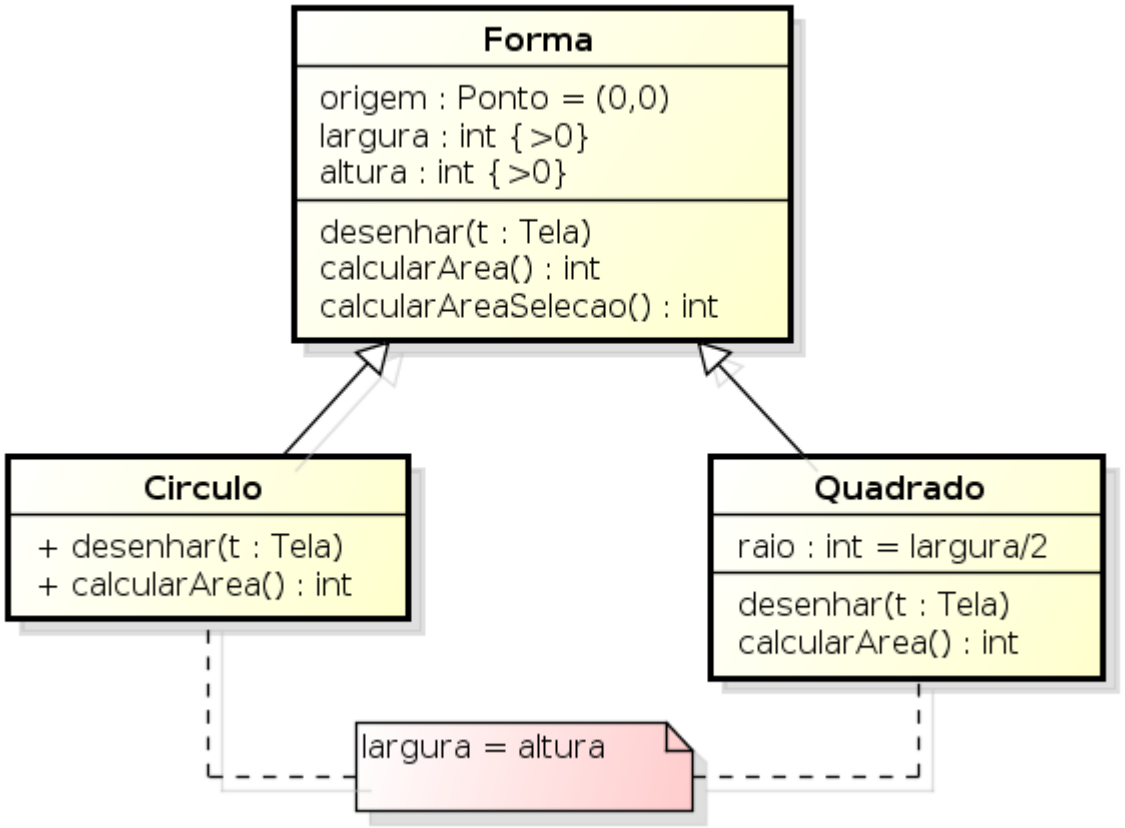


Diagrama de classes

- **Relacionamento de herança**
  - **Operações e classes abstratas**
    - Existem **situações** (como na **classe** `Forma`) em que é **impossível** saber o que **desenhar** e como **calcular** a **área**, pois **não** se **sabe** qual é a **geometria** de Forma – ela é uma **classe** tão **genérica** que parte de suas **funcionalidades** tornam-se **indefinidas**;
    - Nesses casos pode-se simplesmente deixá-las indefinidas – transformando-as em **operações abstratas**. Uma **operação abstrata** é aquela que de **propósito** está **indefinida** – apenas **possui nome**, **parâmetros** e **retorno**, mas **não código** – na classe em que ela foi atribuída;
    - Cria-se, então, um **modo** de **reutilizar** o **nome** da **operação** em **subclasses** mais "**concretas**", que **implementarão** essas **operações** (princípio da substituição e polimorfismo).

Diagrama de classes

- **Relacionamento de herança**
  - **Exemplo de classe abstrata**

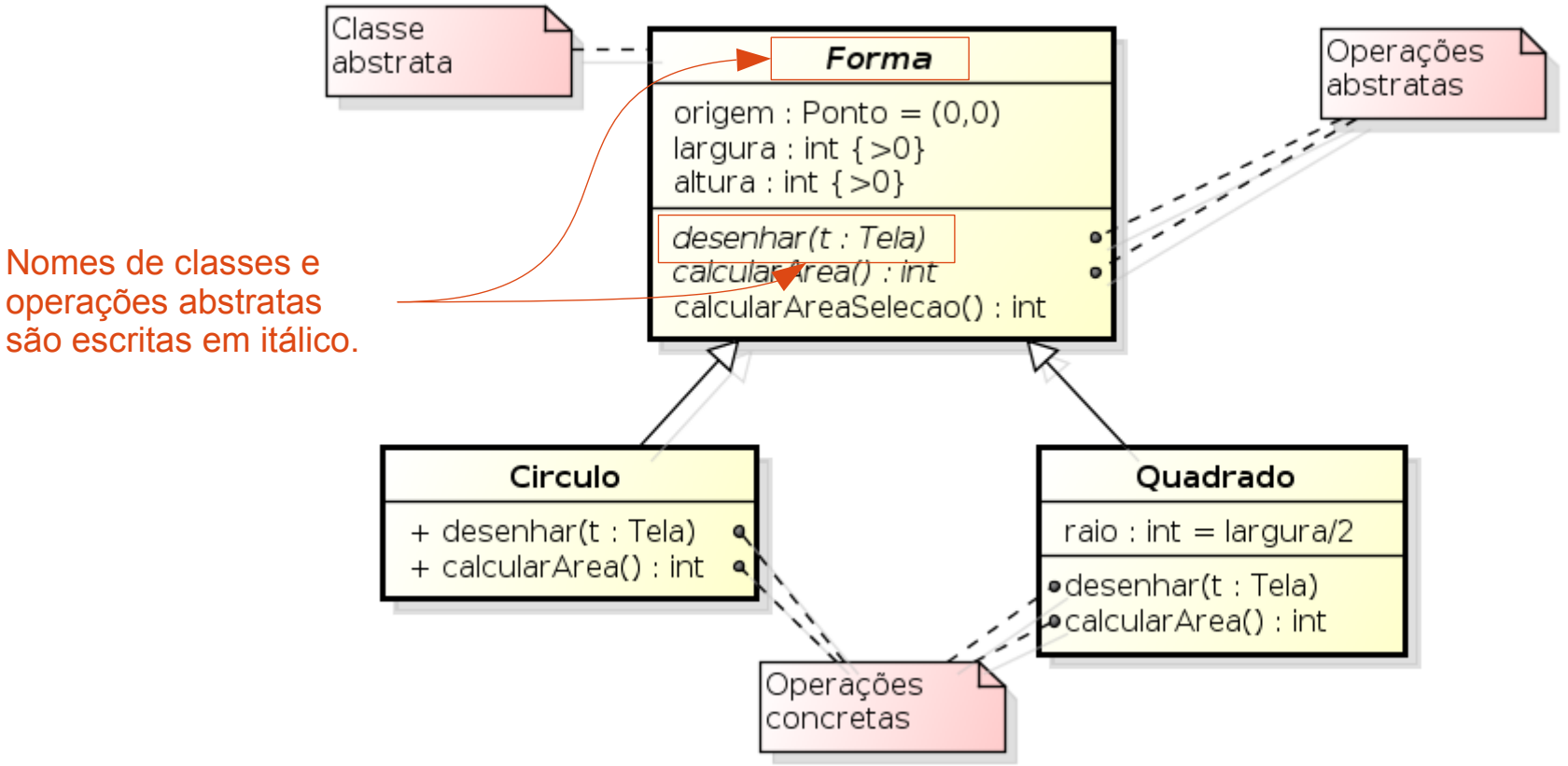
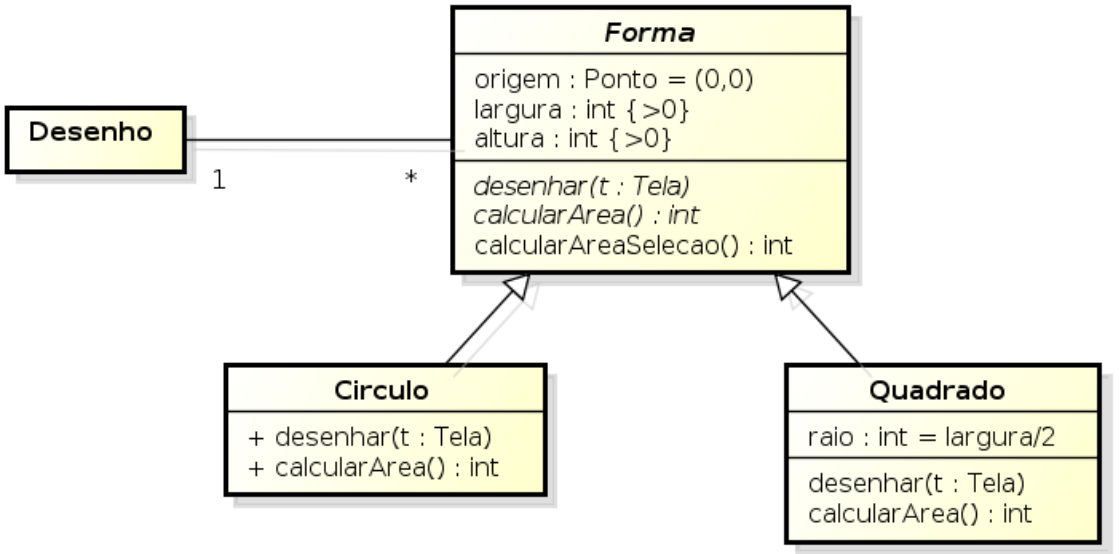


Diagrama de classes

- **Relacionamento de herança**
  - **Conceitos**
    - **Polimorfismo**
      - Uma **operação polimórfica** é aquela que tem **muitas implementações.** Por **exemplo**, as **operações abstratas** `desenhar()` e `calcularArea()` da classe `Forma` **possuem**, **cada uma**, **duas implementações diferentes**, uma na **classe** `Quadrado` e **outra** na **classe** `Circulo`;
      - O **que** faz com que o **polimorfismo** seja um aspecto **essencial** da **orientação** a **objetos** é que ele **permite enviar** a **mesma mensagem** (**executar** a **mesma operação**) em **objetos** de **diferentes classes** e então **fazer** com que eles **respondam adequadamente.**
      - Executando `desenhar()` em um **objeto** da classe **Quadrado** faz com que o **objeto** desenhe um **quadrado**; **executando** `desenhar()` em um **objeto** da classe **Circulo** faz com que o objeto **desenhe** um **círculo**, sem a necessidade de saber qual o tipo do objeto envolvido.

Diagrama de classes

- **Relacionamento de herança**
  - **Exemplo** de **polimorfismo**



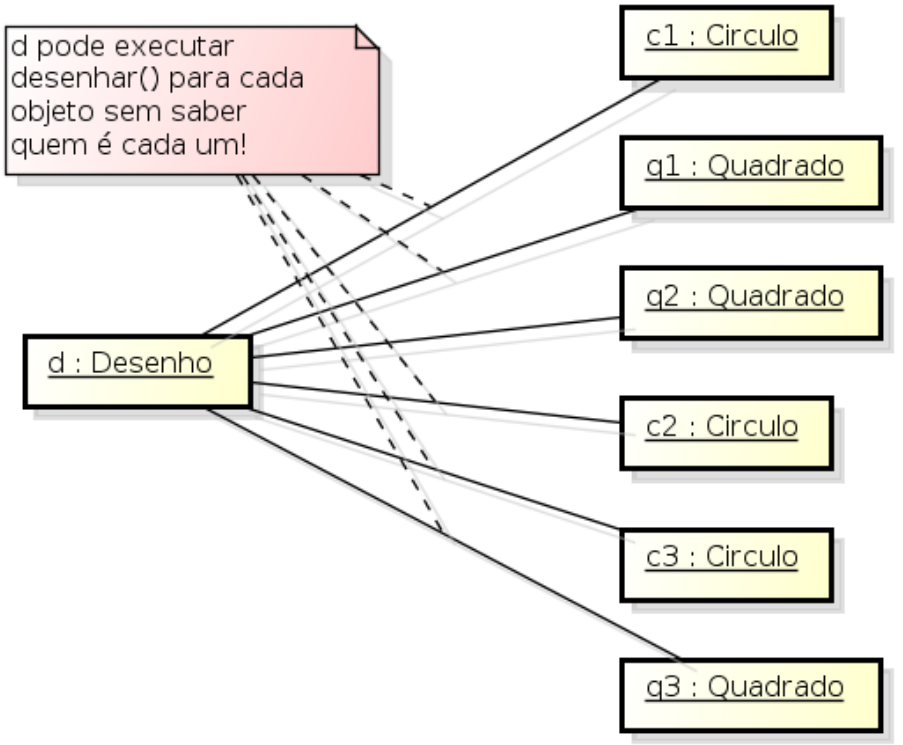
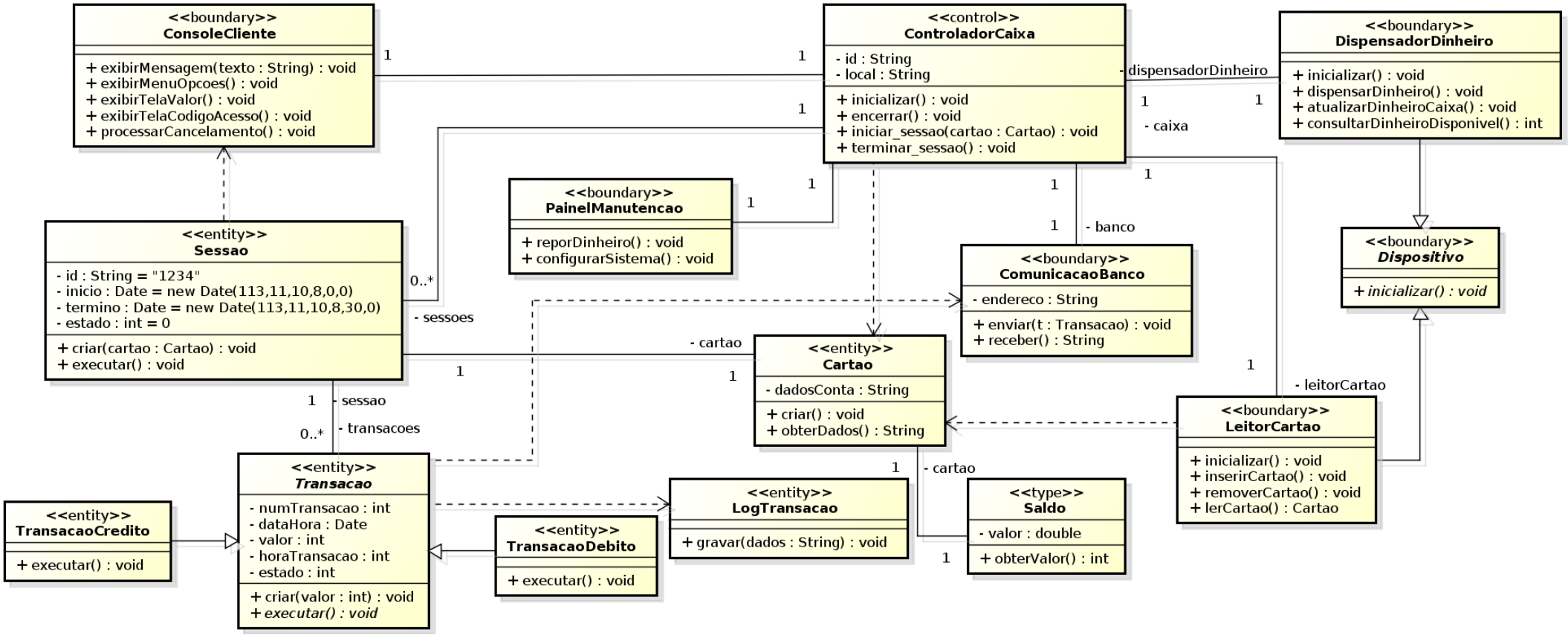


Diagrama de classes

- **Relacionamento de herança**
  - **Exemplo de polimorfismo**
    - Imaginar que um **documento** pode estar **associado** a **diversos objetos** de **desenho** (considerar apenas círculos e quadrados, mas poderiam ser muito mais;
    - Esta **associação** indica que um **objeto** do tipo `Desenho` tem **ligações** com **zero** ou **mais objetos** do tipo `Forma`;
    - **Princípio** da **substituição**: a lista de **objetos** de `Desenho` conterá apenas **quadrados** e **círculos** (objetos de classes concretas).
    - Sempre que um **desenho** precisar se "**desenhar**", basta ele **invocar** a **operação** `desenhar()` de **cada objeto** de sua **lista** de **formas**;
    - Por **polimorfismo**, **objeto** de **desenho** desenha corretamente suas formas sem saber detalhe algum delas!

Diagrama de classes

- **Exemplo**
  - Classes de um caixa eletrônico com relacionamentos



Agenda

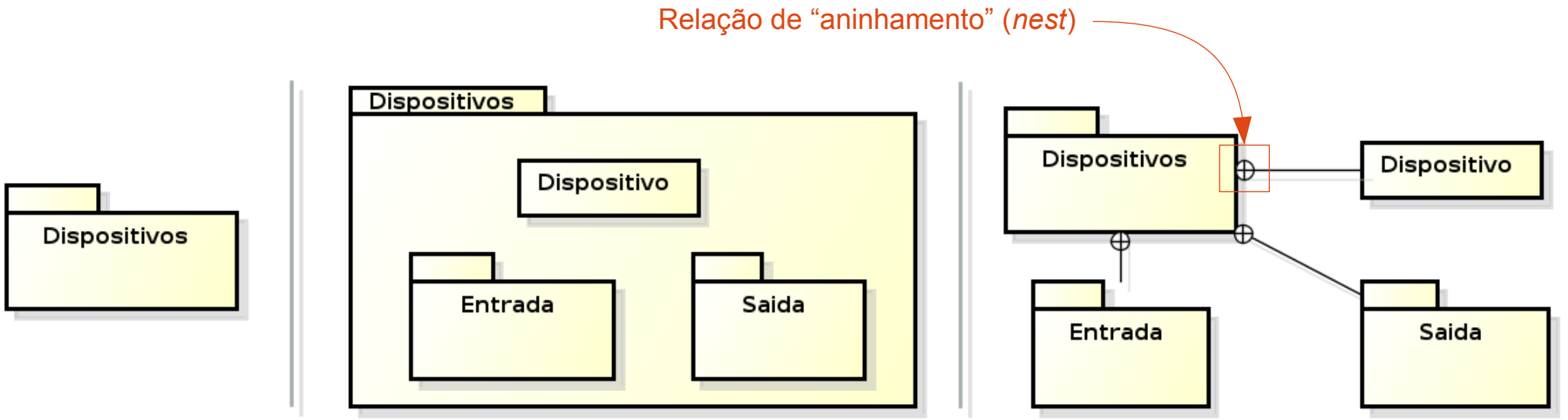
- Diagrama de classes
- Diagrama de pacotes
- Diagrama de sequência
- Diagrama de atividade

Diagrama de pacotes

- **Pacotes**
  - Um **pacote** (“package”) em UML é um **mecanismo** de **agrupamento lógico** de **elementos**. Cada **pacote** define o seu **próprio espaço** de **nomes** (“namespace”) dentro do qual **todos** os **nomes** devem ser **exclusivos**;
  - **Finalidade** de um **pacote**:
    - **Prover** um **espaço** de **nomes encapsulado** dentro do qual todos os nomes devem ser **únicos**;
    - **Agrupar elementos semanticamente relacionados**;
    - **Definir** “**limites semânticos**” no modelo;
    - **Fornecer unidades** para que o **trabalho** seja **dividido** e **executado** de modo **paralelo** e também **facilitar** o **gerenciamento** de **configuração** do software.

Diagrama de pacotes

- **Pacotes**
  - O **ícone** do **pacote** é uma **pasta**, e o **nome** do **pacote** pode ser **exibido** na guia, se o **conteúdo** da **embalagem** for **exibido**, ou no **corpo** da **pasta**, caso **contrário**:



Formas equivalentes de apresentar pacotes

Diagrama de pacotes

- **Pacotes**
  - Um **pacote** define um **espaço** de **nomes** – ao se **associar** um **elemento** a um **pacote**, este **elemento** será **qualificado** pelo nome do **pacote**.
  - Os **nomes** criam um **caminho** no qual é **possível** se **chegar** ao **elemento**. O **separador** de **espaço** de nomes é o símbolo "**: :**":

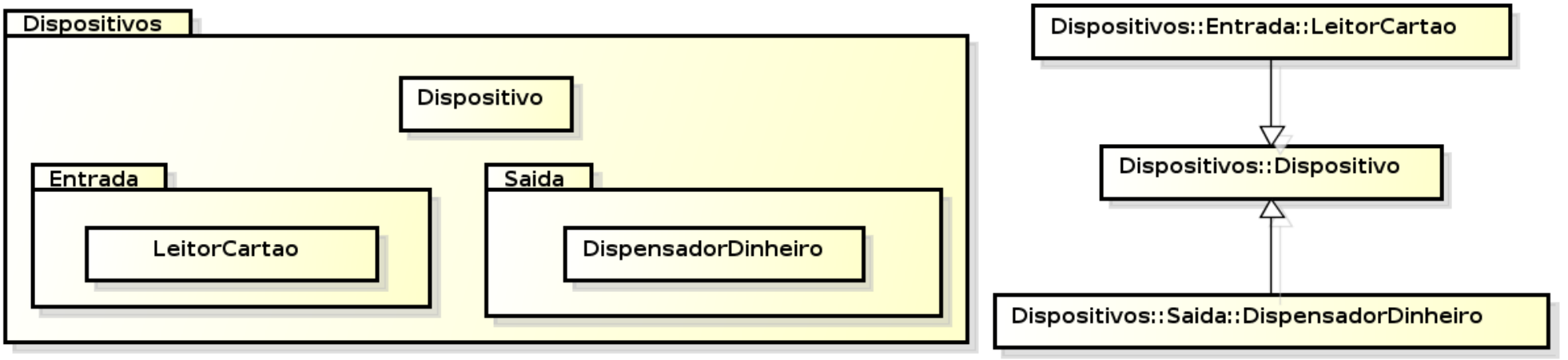


Diagrama de pcotes

- **Pacotes**
  - **Dependência entre pacotes**
    - As **dependências** entre **classificadores** aplicam-se **igualmente** a **pacotes**:

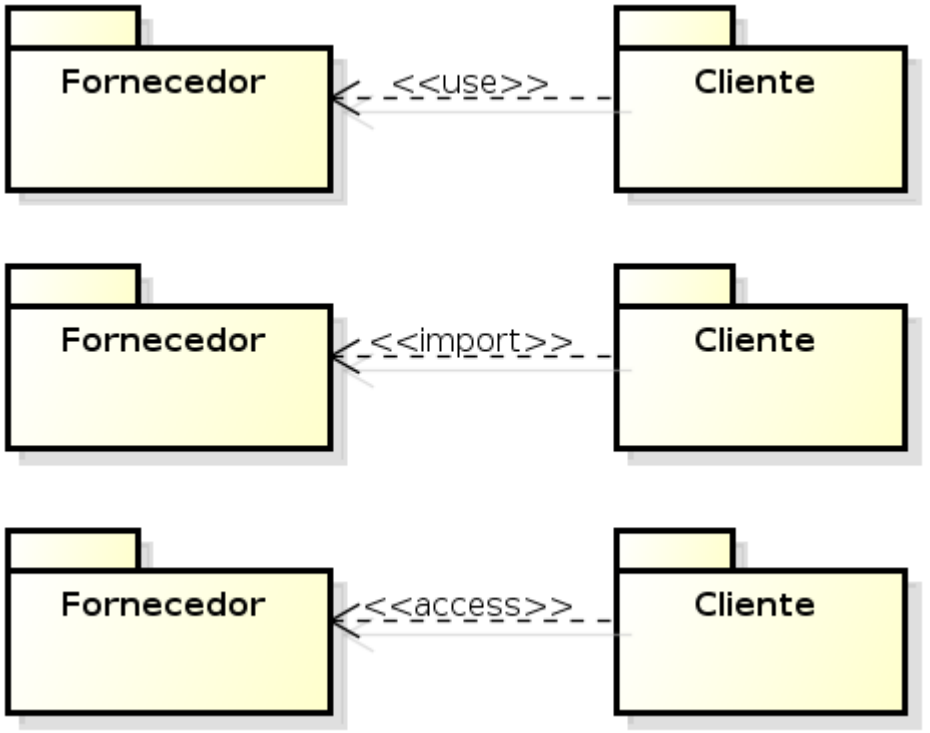


Diagrama de pacotes

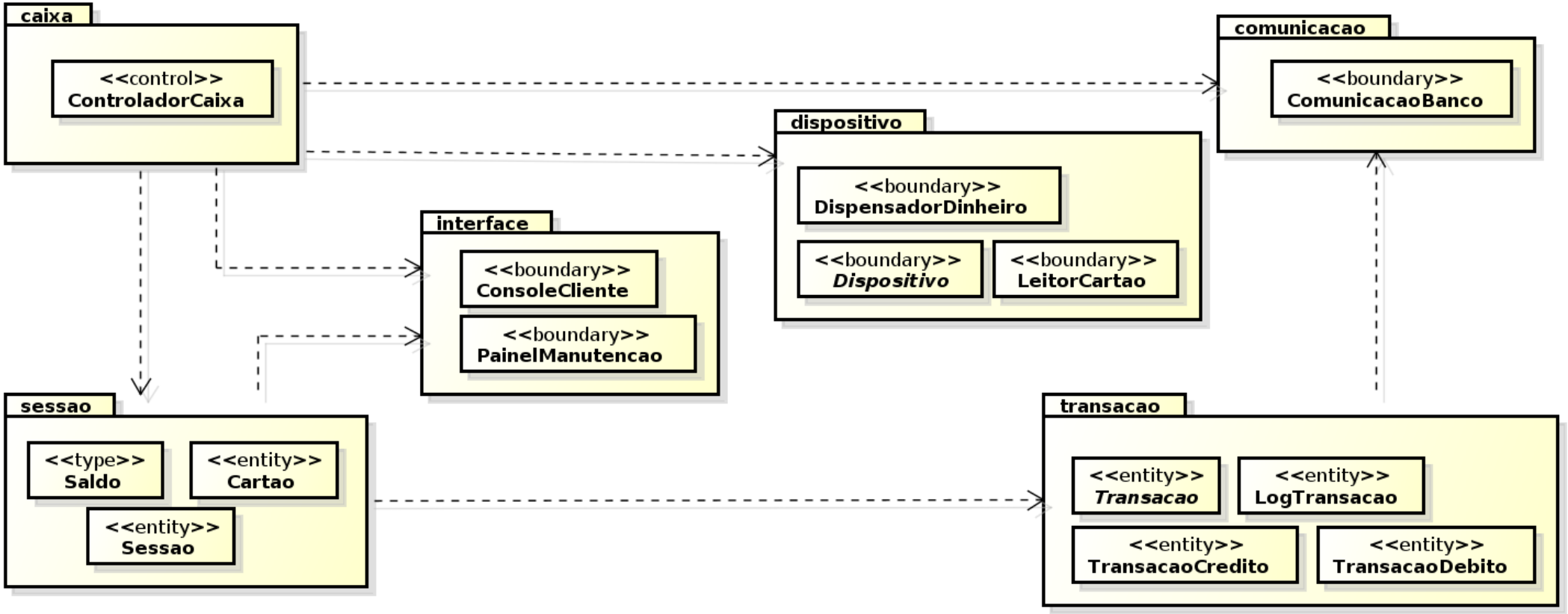
- **Pacotes**
  - **Acoplamento**
    - O **objetivo** de se **criar pacotes** de **análise** é **organizar** as **classes** em um **conjunto** de **pacotes coesos** formando **partições** e **camadas**.
    - Esta **tarefa** é **denominada** de **análise** da **arquitetura** e seu **objetivo** é tentar **minimizar** a **quantidade** de **acoplamentos** no sistema. Pode-se fazer isso de **três formas**:
      - **Minimizar** as **dependências** entre **pacotes** de **análise**.
      - **Minimizar** o **número** de **elementos públicos** em cada pacote de análise.
      - **Maximizar** o **número** de **elementos privados** em cada pacote de análise.

Diagrama de pacotes

- **Pacotes**
  - **Processo para determinar pacotes**
    - **Pacotes** de **análise** são **descobertos** ao **longo** de um **período** de **tempo** que o modelo se **desenvolve** e **amadurece**;
    - Os **pacotes** de análise devem refletir **agrupamentos reais**, **semânticos** de **elementos**;
    - Os **modelos estáticos** são os **modelos** mais **úteis** para serem **utilizados** como **fonte** de **pacotes**, principalmente quando se observa a **existência** de **grupos coesos** de **classes** nos **diagramas** de **classe** e quando existem **hierarquias** de **herança**;
    - **Modelos** de **casos** de **uso podem** ser **fonte** de **pacotes**, pela perspectiva de processos.

Diagrama de pacotes

- **Exemplo**
  - Pacotes para o problema do caixa eletrônico



Agenda

- Diagrama de classes
- Diagrama de pacotes
- Diagrama de sequência
- Diagrama de atividade

Diagrama de sequência

- **Conceitos**
  - Um **diagrama** de **sequência** é um **tipo** de **diagrama** de **interação** da **UML** cujo **propósito** é (AMBLER, 2005; PILONE; PITMAN, 2005):
    - **Validar** e **detalhar** um **cenário** existente em um **caso** de **uso** ou outro elemento tal como um **subsistema**;
    - **Conferir visualmente** a **invocação** de **operações** definidas nas **classes** e **auxiliar** na **modelagem** das **classes**;
    - Auxiliar na **identificação** de **classes** cujo **comportamento pode** ser mais **complexo** e **necessitar auxílio** de **diagramas** de **máquina** de **estado**, por exemplo;
    - **Detectar gargalos** em **projetos orientados** a **objetos** – **observando** como as **mensagens** são **trocadas** entre os objetos, é **possível detectar** a **necessidade** de **distribuir melhor** as **responsabilidades**.

Diagrama de sequência

- **Conceitos**
  - Diagrama de sequência **detalham** as **interações entre** os **objetos - enfatizam** o **tipo** e a **ordem** das **mensagens passadas** entre os **elementos durante** a **execução**;
  - **Enfatizam** a **comunicação** entre **objetos**, e **não** a **manipulação** de **dados associada** a essa comunicação;
  - É o **tipo mais comum** de **diagrama** de **interação** e é **intuitivo** para **novos usuários** de **UML**.

Diagrama de sequência

- **Interações**
  - São **unidades** de **comportamento** de um **classificador**;
  - O **classificador fornece** o **contexto** para a **interação**, que **pode usar** qualquer uma das **características** de seu **classificador** de contexto ou **qualquer** outra **que** ele **tenha acesso**;
  - Na **realização** de um **caso** de **uso**, o **classificador** de **contexto** é um **caso** de **uso – deve-se criar uma** ou **mais interações** para **demonstrar** como o **comportamento especificado** pelo **caso** de **uso** pode ser **realizado** por **instâncias** de **classificadores** que **enviam mensagens** entre si;
  - **Conforme** se **trabalha** com **diagramas** de **interação**, **começa-se** a **entender mais** e **mais** das **operações** e **atributos** das **classes** de **análise permitindo** a **revisão** e **atualização** dos **diagramas** de classes de análise.

Diagrama de sequência

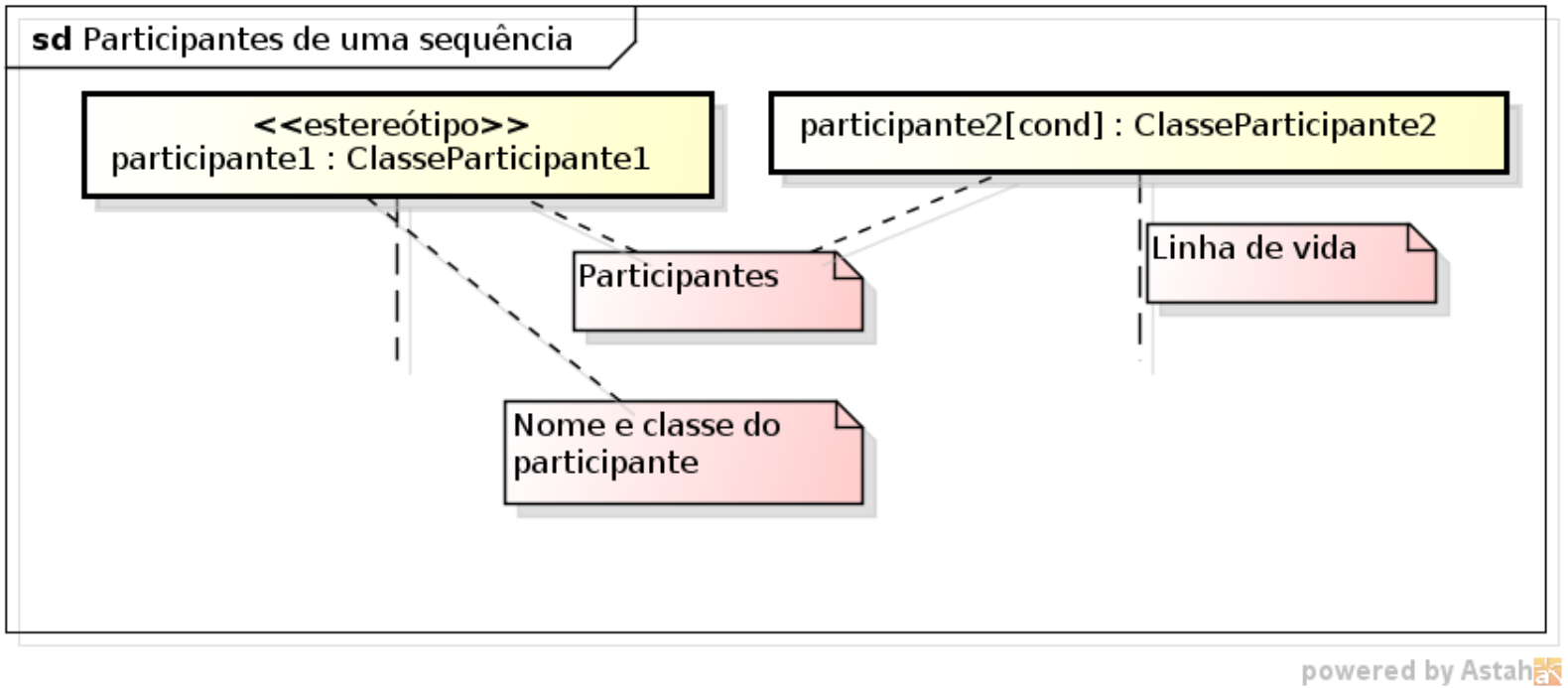
- **Cenários**
  - É uma **sequência específica** de **entradas** e **saídas** que **representa** um **caminho** de um **caso** de **uso**;
  - **Casos** de **uso** podem ter **diversos cenários**;
  - **Cenários** são **úteis**, pois **explicam** como o **sistema** irá **interagir** com **outros elementos** no seu **ambiente** (atores);
  - **Ajudam especialistas** do **domínio** em **entender** o **problema** e **descartar cenários irreais**;
  - Formas de **descrever cenários**: **diagramas** de **sequência** (mais usado), **diagramas** de **comunicação**, **diagramas** de **temporização**;
  - Primeiramente, **especificar cenários** das **operações normais** – depois os **cenários** contendo **exceções**.

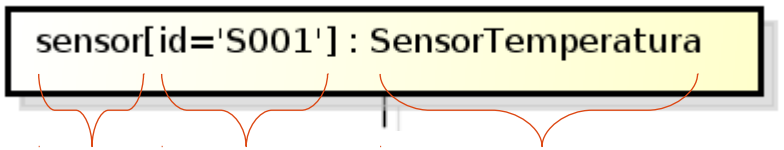
Diagrama de sequência

- **Linha de vida**
  - **Representa** um **participante** em uma **interação**;
  - **Detalha** como uma **instância** de um **classificador específico** (objeto de uma classe, por exemplo) **participa** da **interação**;
  - A **sintaxe** de uma linha de vida **engloba** um **nome** (**opcional**), um **tipo**, e um **seletor** (**opcional**):
    - **Nome**: é utilizado para **referir** a **linha** de **vida** dentro de uma **interação**;
    - **Tipo**: é o **nome** do **classificador** instanciado pela linha de vida.
    - **Seletor**: é uma condição lógica que pode ser usada para **selecionar** uma **única instância** que satisfaça a **condição**.

Diagrama de sequência

- **Linha de vida**
  - **Representação**

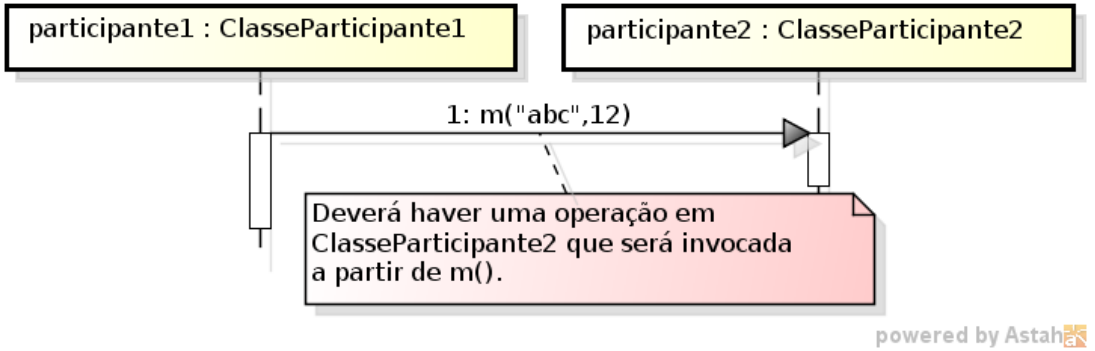




Nome Seletor Classe

Diagrama de sequência

- **Mensagem**
  - **Representa** um **tipo específico** de **comunicação entre duas linhas de vida** em uma **interação**:
    - A **execução** de **operação**: **mensagem** de **chamada** – é um **pedido** para que uma **operação** da **linha** de **vida** que **tem** a **mesma assinatura** da **mensagem seja executada**:



    - A **criação** ou **destruição** de uma **instância**: **mensagem** de **criação** ou **destruição** (construtor/destrutor em POO);
    - O **envio** de um **sinal** (mensagem assíncrona que causa transição de estado).

Diagrama de sequência

- **Mensagem**
  - **Quando uma linha** de **vida** está **executando** uma **mensagem**, diz-se que ela **possui** o **foco de controle ou** de **ativação**;
  - Conforme a **interação progride** ao **longo** do **tempo**, a **ativação** se **move** entre **linhas** da **vida** e esse **mecanismo** é **denominado** de **fluxo** de **controle**.

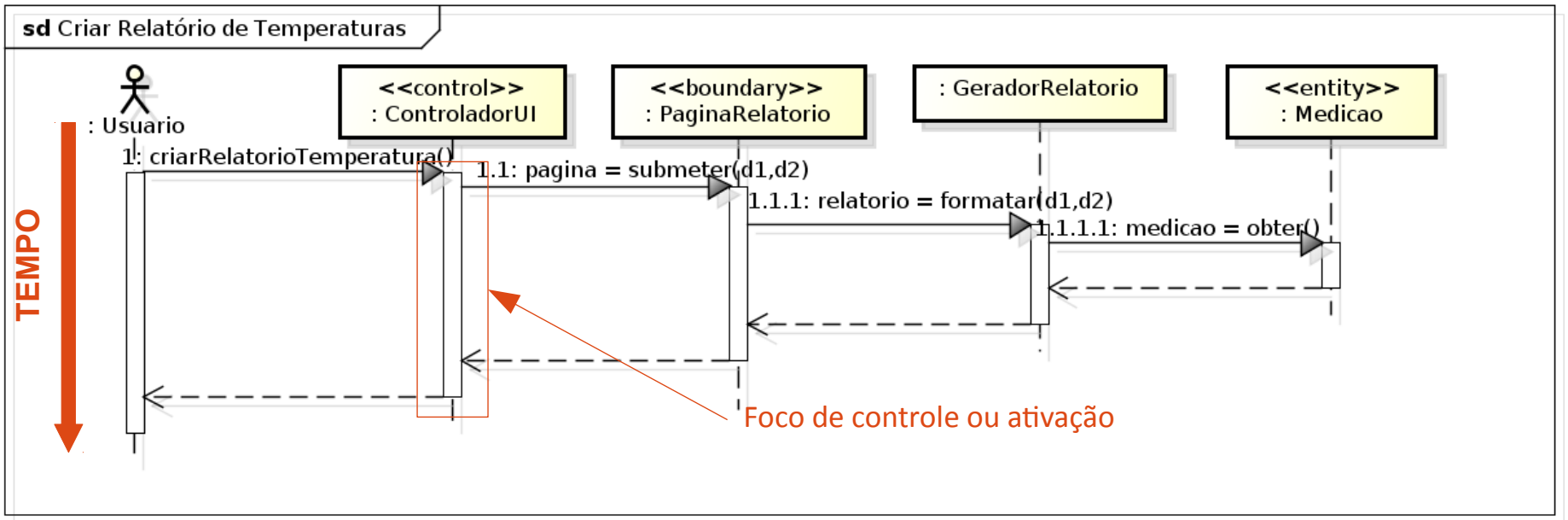


Diagrama de sequência

- **Mensagem**
  - **Forma geral**

```
atributo = nome_mensagem_ou_sinal(argumentos):tipo_retorno
```

  - **Tipos**
    - **Síncrona**: o **emissor aguarda** o **receptor** para **concluir** a **execução** da **operação** requisitada;
    - **Assíncrona**: o **emissor não aguarda** o **receptor** e **continua** para o **próximo** passo;
    - **Retorno**: o **receptor** de uma **mensagem anteriormente enviada retorna** o **foco** de **controle** ao **emissor** da mensagem;
    - **Criação**: o **emissor cria** uma **instância** do **classificador receptor**;
    - **Destruição**: o **emissor destrói** o **receptor**;
    - **Mensagem encontrada**: o **emissor** está **fora** do **escopo** da **interação**;
    - **Mensagem perdida**: nunca chega ao destino (erro).

Diagrama de sequência

- **Mensagem**
  - **Simbologia**

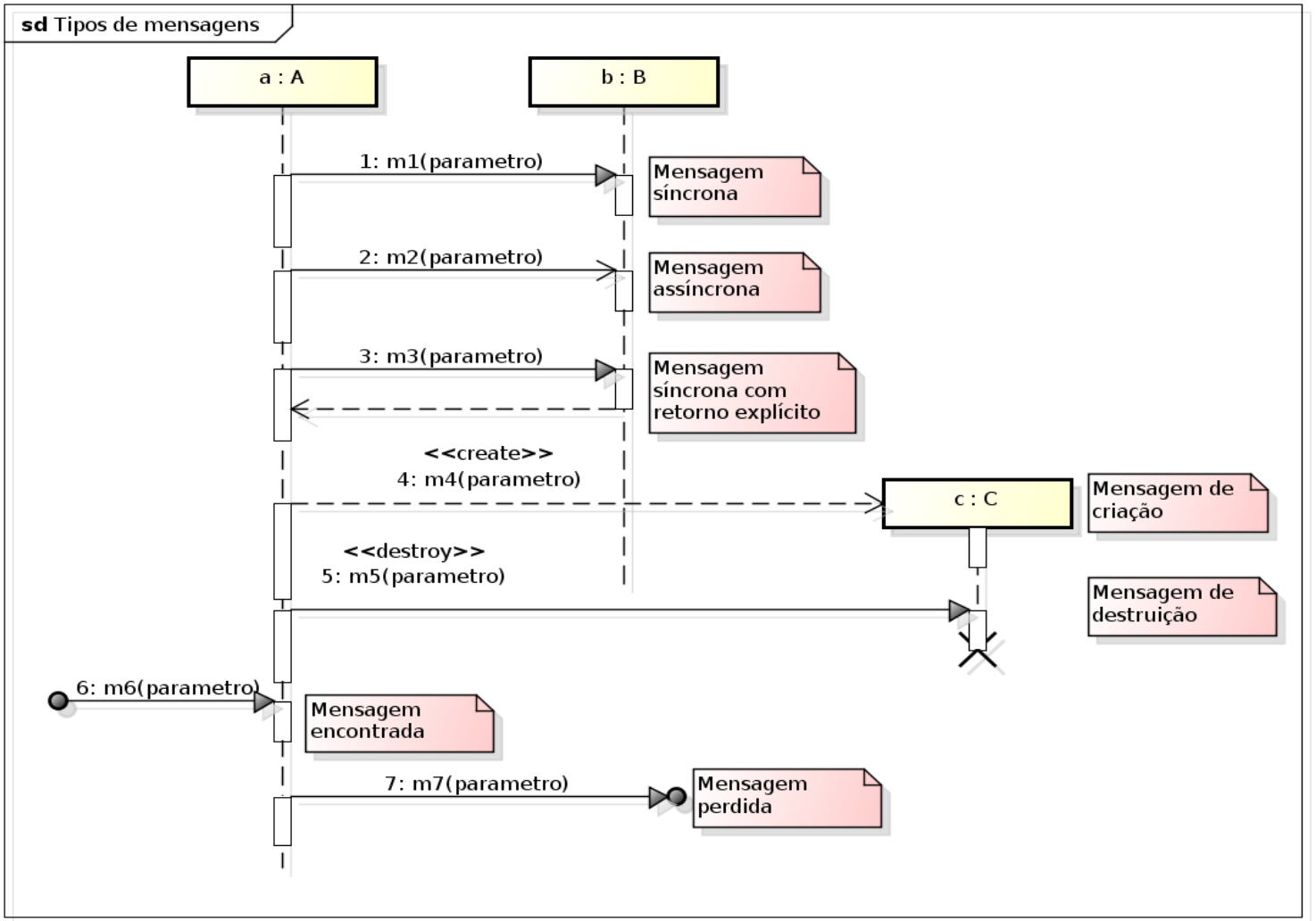


Diagrama de sequência

- **Fragmentos combinados**
  - **Conceitos**
    - Pode-se ser **dividir** um **diagrama** de **sequência** em **áreas denominadas** de **fragmentos combinados**;
    - Cada **fragmento combinado** tem um **operador**, **um ou mais operandos** e **zero** ou **mais condições** de **guarda**:
      - O **operador determina** como seus **operandos** são **executados**;
      - As **condições** de **guarda determinam** se seus **operandos podem executar**.
    - As **condições** de **guarda** são **expressões lógicas** e o **operando executa** se, e **somente se** a **expressão** é **avaliada** como **verdadeira**;
    - **Uma única condição** de **guarda** pode ser **aplicada** a **todos** os **operandos** ou **cada operando pode** ter sua **própria condição** de **guarda exclusiva**.

Diagrama de sequência

- **Fragmentos combinados**
  - **Conceitos**
    - **Fragmentos combinados mais comuns** (total de 13!):

Operador	Significado
`alt`	Fragmento alternativo para condições lógicas mutualmente exclusivas, expressadas nas condições de guarda
`loop`	Fragmento de repetição enquanto a condição de guarda é verdadeira. Pode-se também ser escrito como loop(n) para representar n repetições e alguns autores usam até como um laço para-até-faça: `loop(i,1,10)`
`opt`	Fragmento opcional que executa se a guarda é verdadeira.
`par`	Fragmentos que são executados em paralelo.
`critical`	Região crítica dentro da qual somente uma thread pode executá-la por vez.

Diagrama de sequência

- **Fragmentos combinados**
  - **Exemplo do fragmento** `alt`

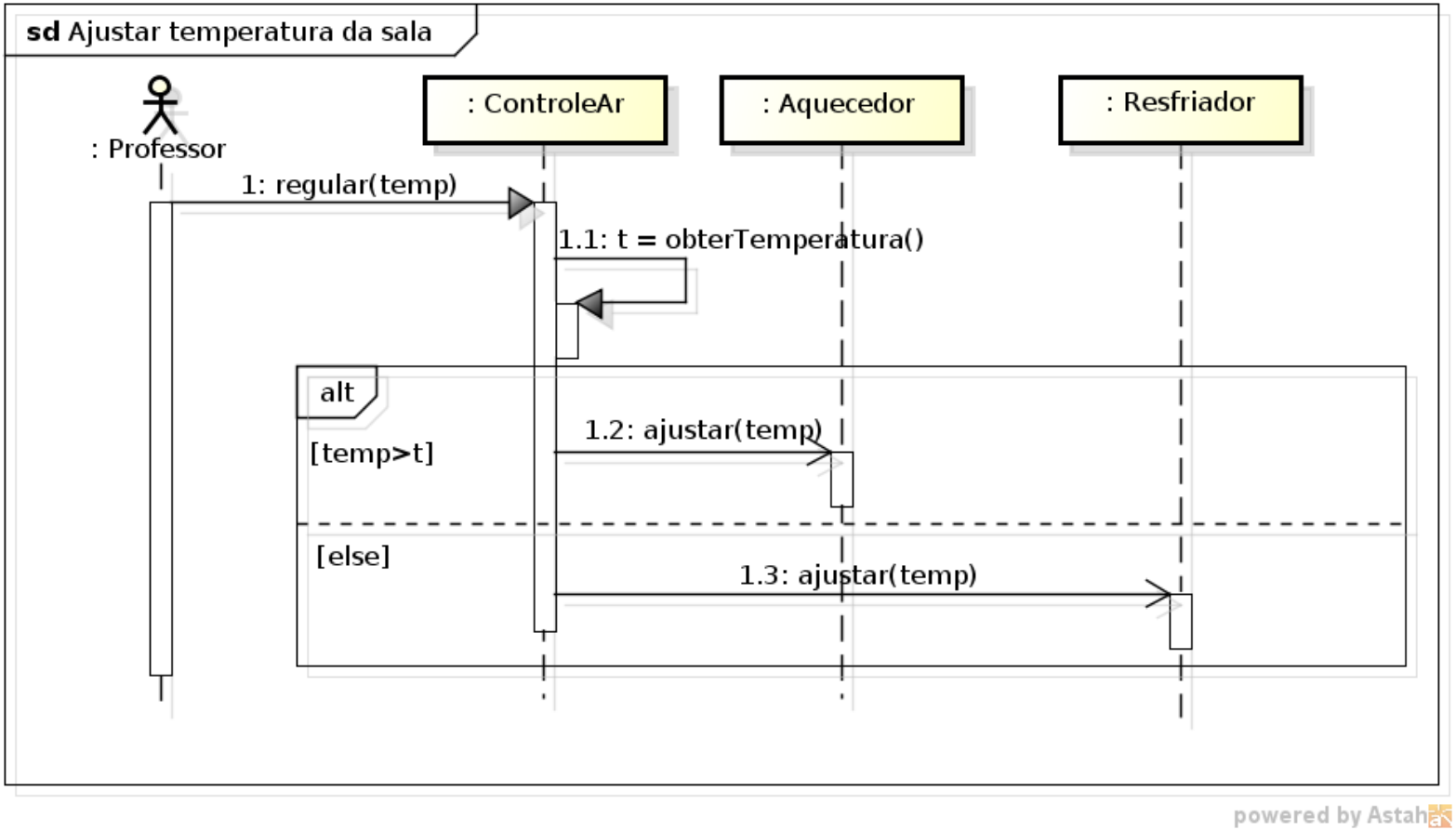


Diagrama de sequência

- **Fragmentos combinados**
  - **Exemplo do fragmento `loop`**

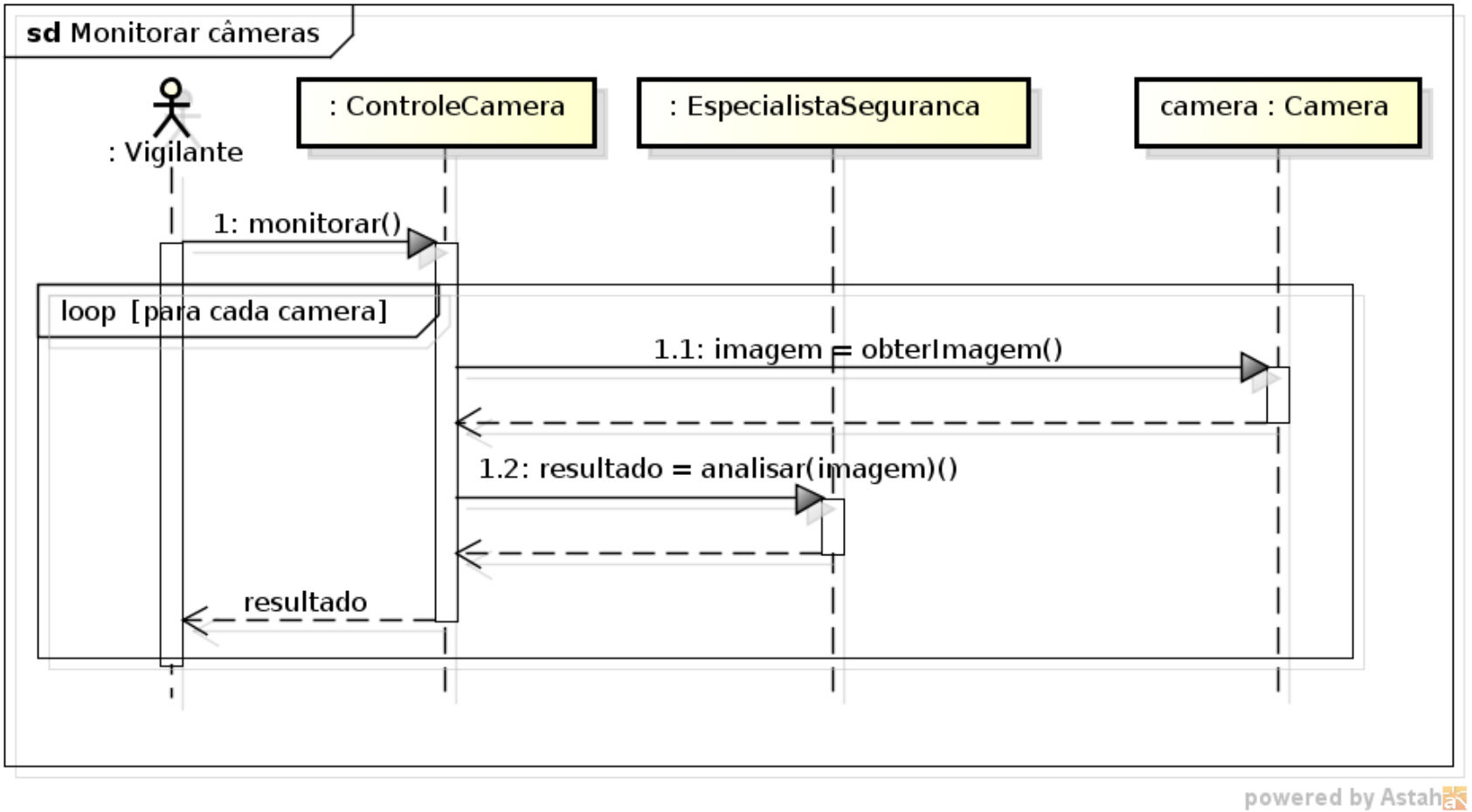


Diagrama de sequência

- **Fragmentos combinados**
  - **Exemplo do fragmento** `opt`

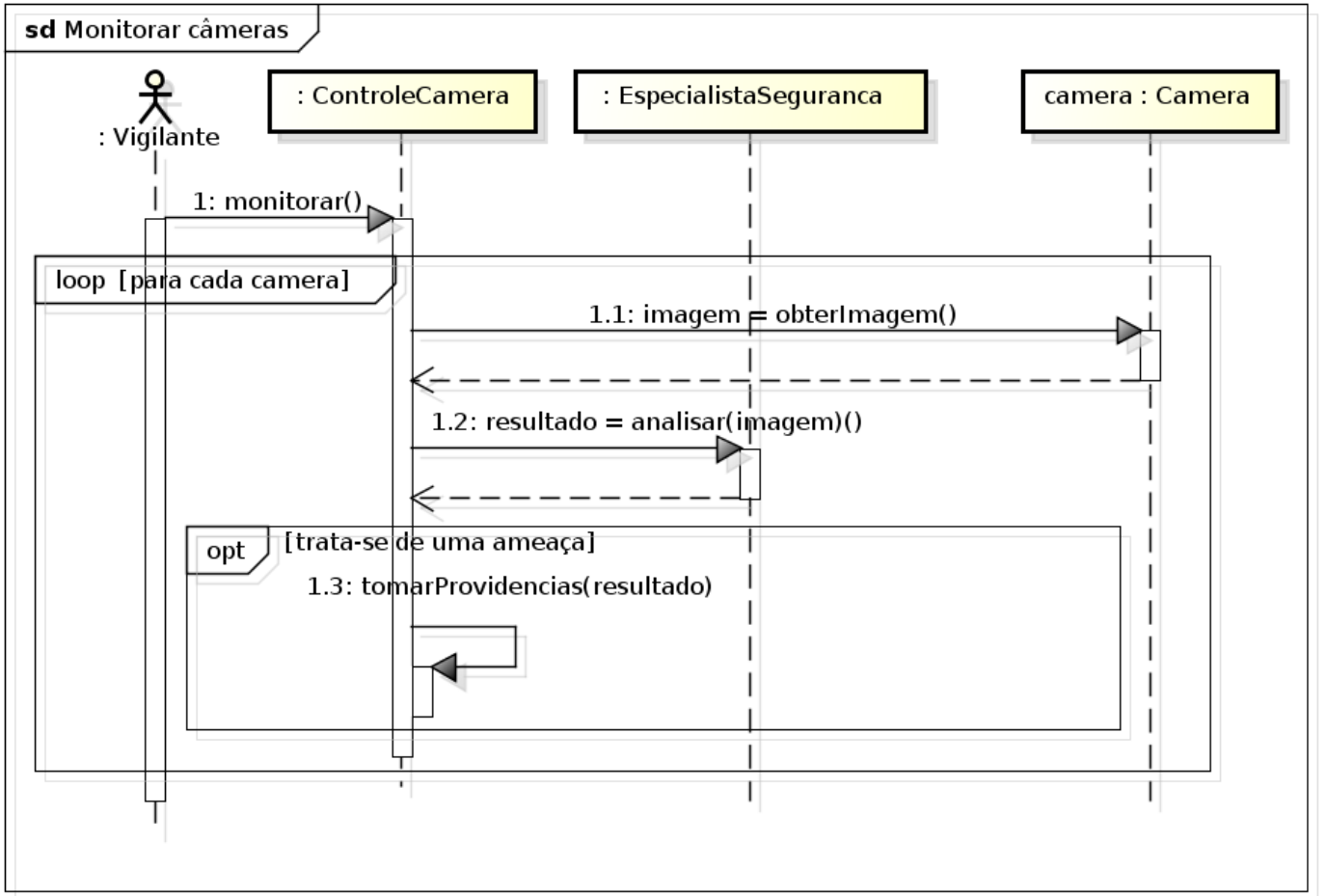


Diagrama de sequência

- **Fragmentos combinados**
  - **Exemplo dos fragmentos `par` e `critical`**

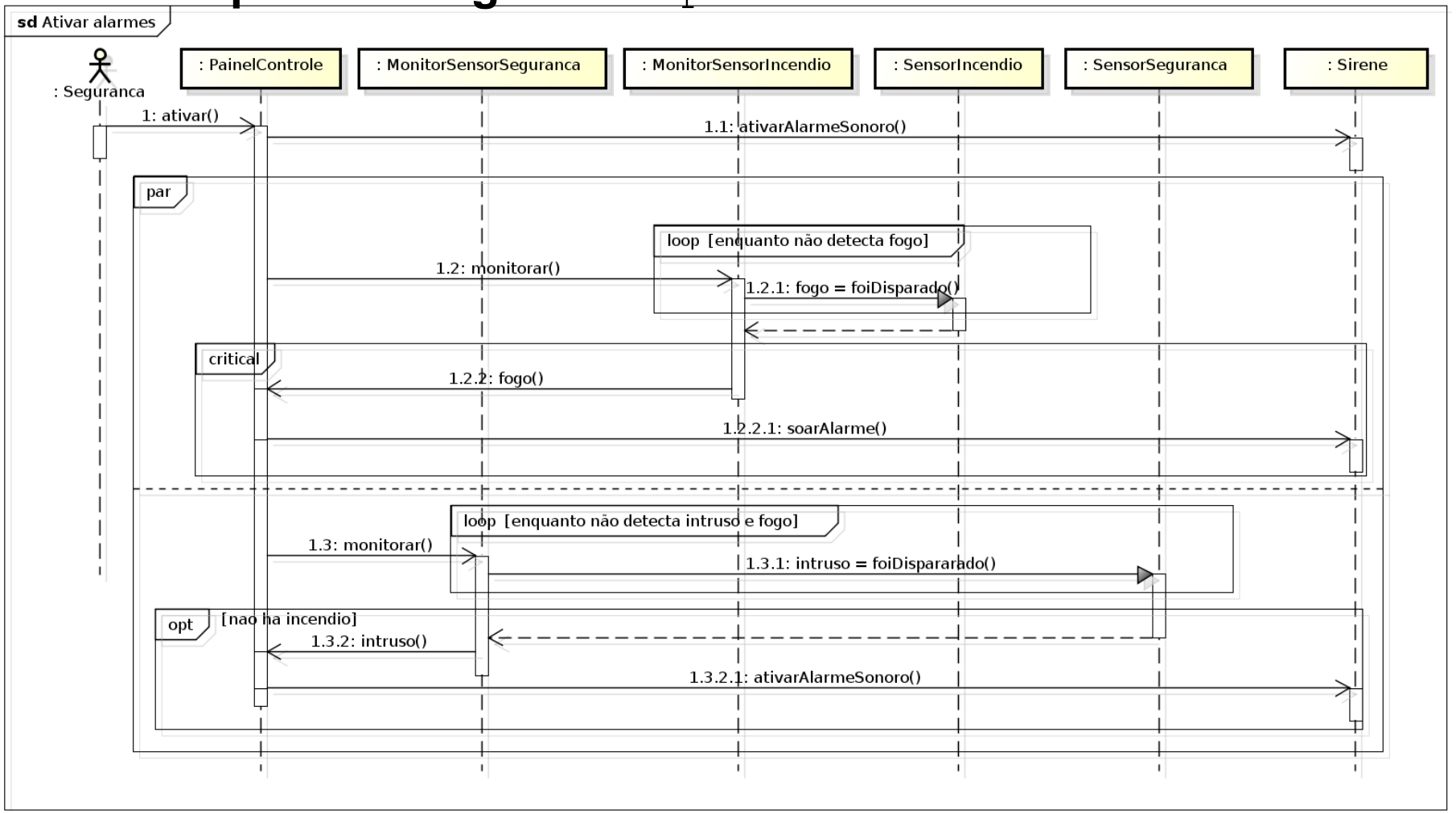
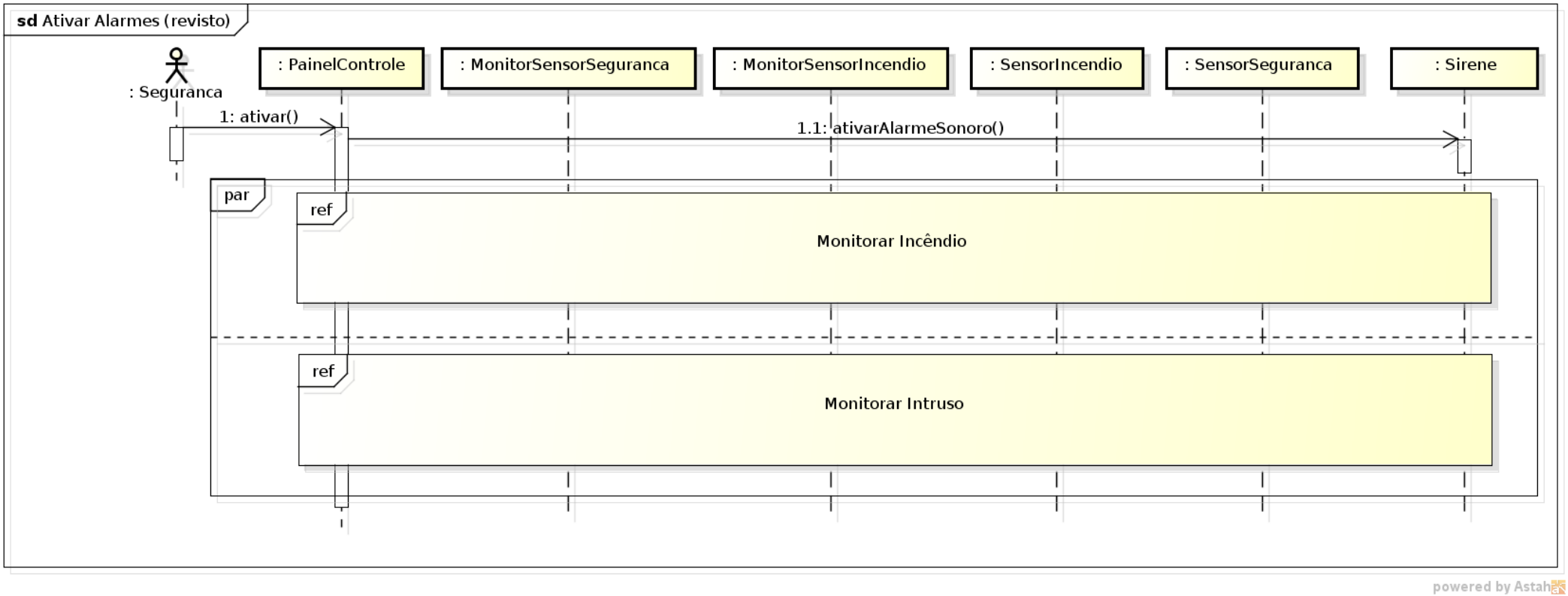


Diagrama de sequência

- **Referenciando diagramas de sequência**
  - **Conceitos**
    - Quando um **diagrama** de se **sequência torna-se muito complexo** é **interessante dividi-lo** em diagramas menores;
    - Cada **diagrama menor possui** um **nome** e um **quadro determinado** pelo **nome** `sd` (de *sequence diagram*);
    - No **diagrama original**, esses **diagramas menores** são **referenciados** por **fragmentos** `ref` (de *reference*);
    - Pode-se, inclusive, utilizar parâmetros, se necessário.

Diagrama de sequência

- **Referenciando diagramas de sequência**
  - **Exemplo**



Agenda

- Diagrama de classes
- Diagrama de pacotes
- Diagrama de sequência
- Diagrama de atividade

Diagrama de atividade

- **Conceitos**
  - São **úteis** para **descrever comportamentos**:
    - **Processos** de **negócios**
    - Casos de Uso
    - ***Workflows***
  - **Elementos do diagrama**
    - **Atividade**: representa um **comportamento**, que é **realizado** por conjunto de **ações**;
    - **Ação**: representa um **passo elementar** em uma **atividade**. São **exemplos** de **ações**:
      - Funções matemáticas;
      - Chamadas de outros comportamentos (atividades);
      - Cálculo de comissão sobre vendas;
      - Gerar uma lista de itens para serem repostos no estoque.

Diagrama de atividade

- **Simbologia básica**

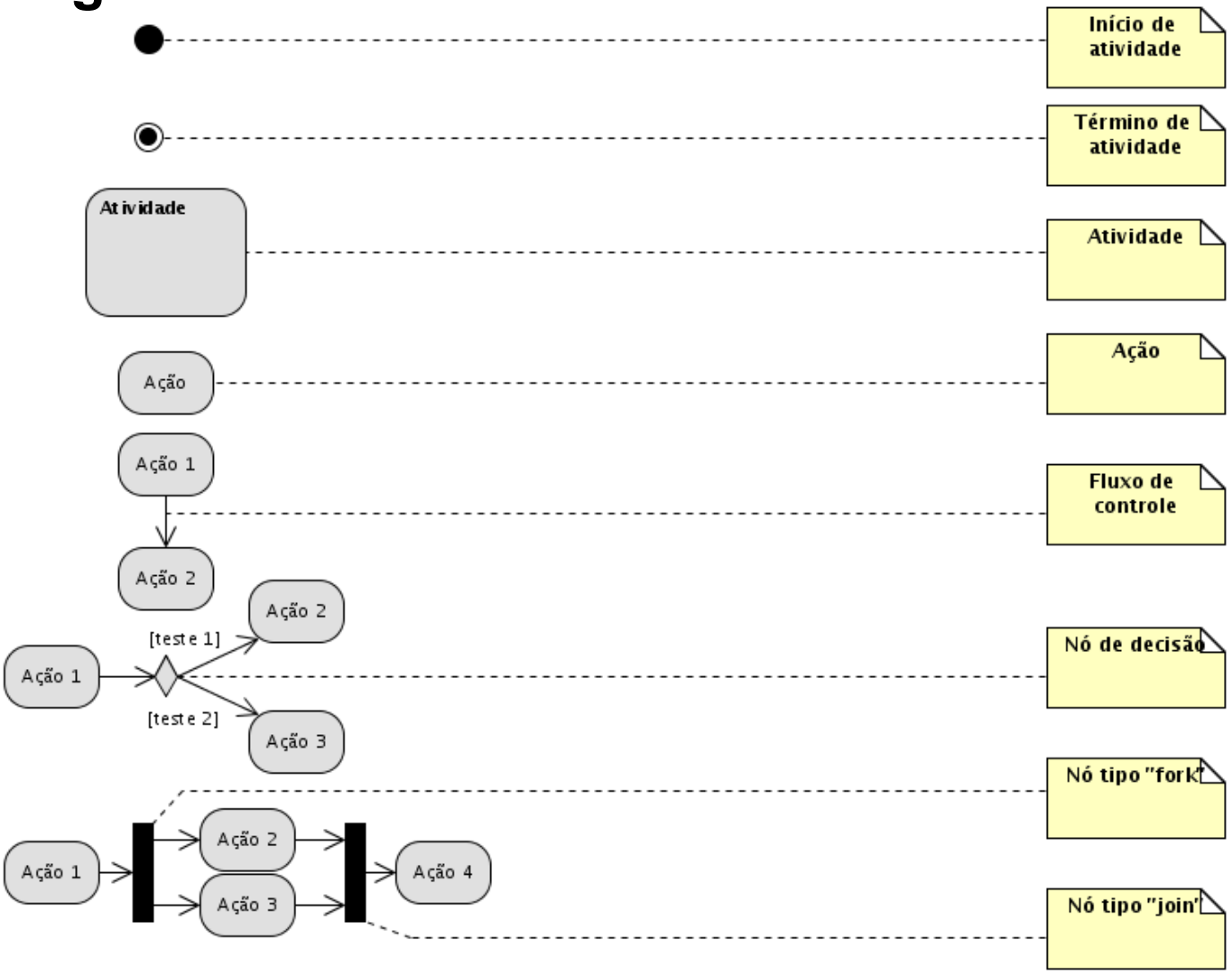


Diagrama de atividade

- **Aplicação em processos de negócios**
  - **Exemplo de processo**: *Processar pedidos de vendas* (versão simplificada)

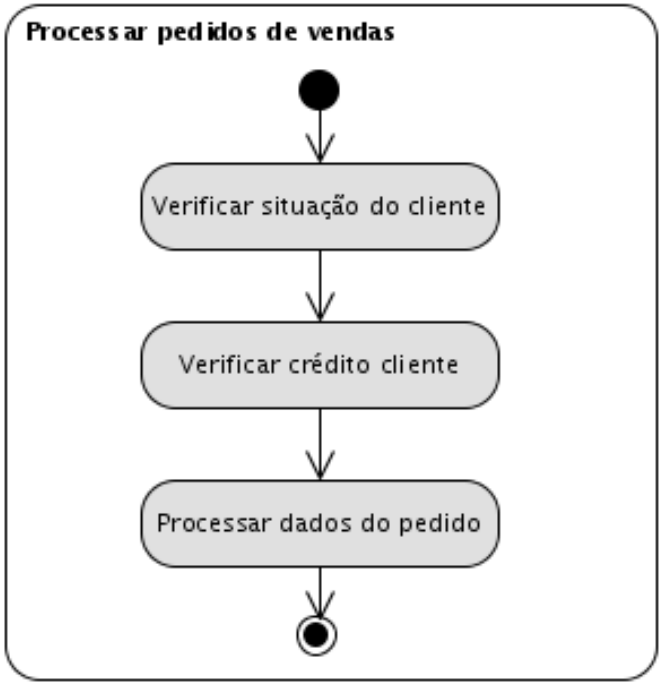


Diagrama de atividade

- **Aplicação em processos de negócios**
  - **Exemplo de processo**: *Processar pedidos de vendas* (versão transfuncional)

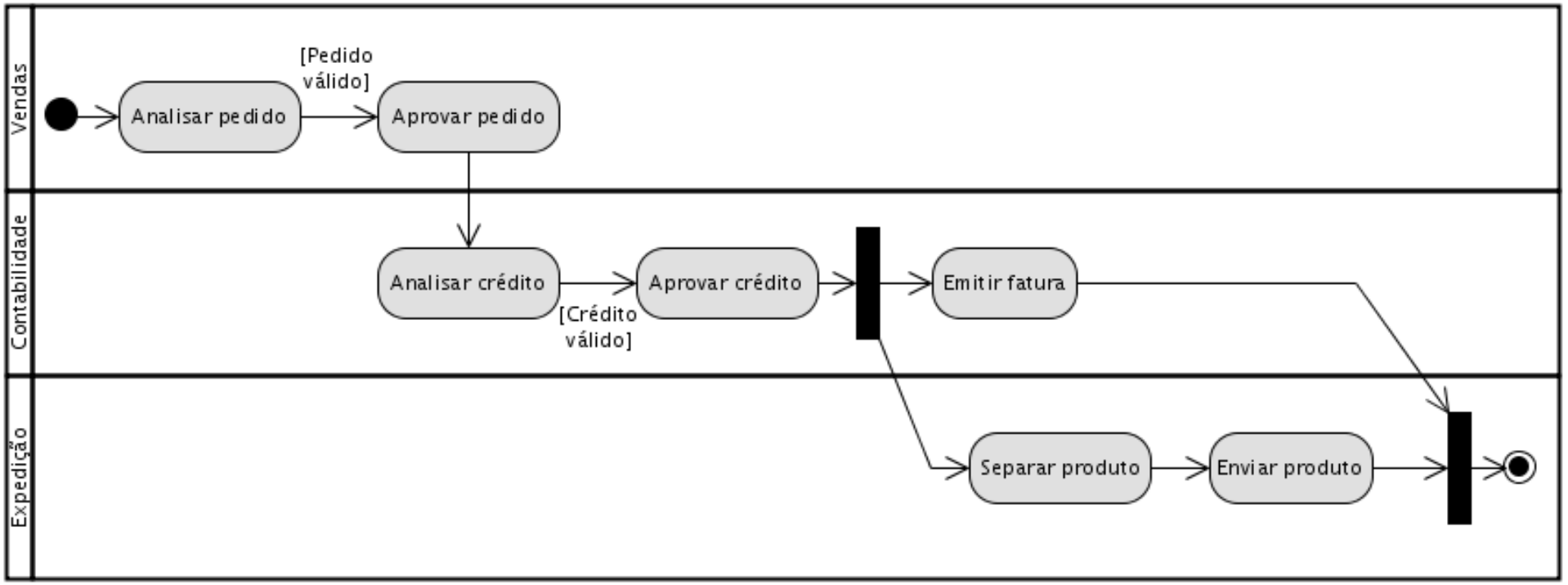


Diagrama de atividade

- **Aplicação em casos de uso**
  - **Exemplo de caso de uso**: *Retirar dinheiro*

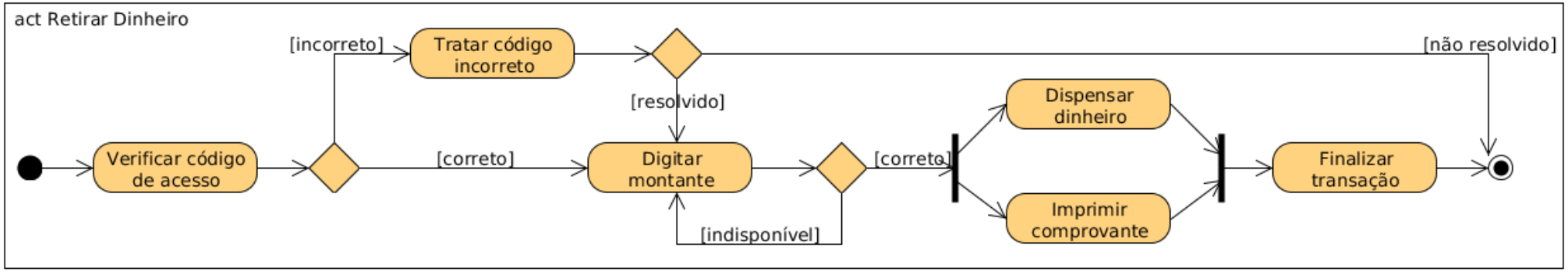


Diagrama de atividade

- **Nó de objeto**
  - Serve para indicar que **instâncias** de um **classificador** particular estão **disponíveis** em um **ponto** específico da **atividade**;
  - Os **ramos** que **entram** e **saem** de um **nó** de **objeto** são denominados **fluxos** de **objeto** e representam o movimento de **objetos** no **fluxo**.

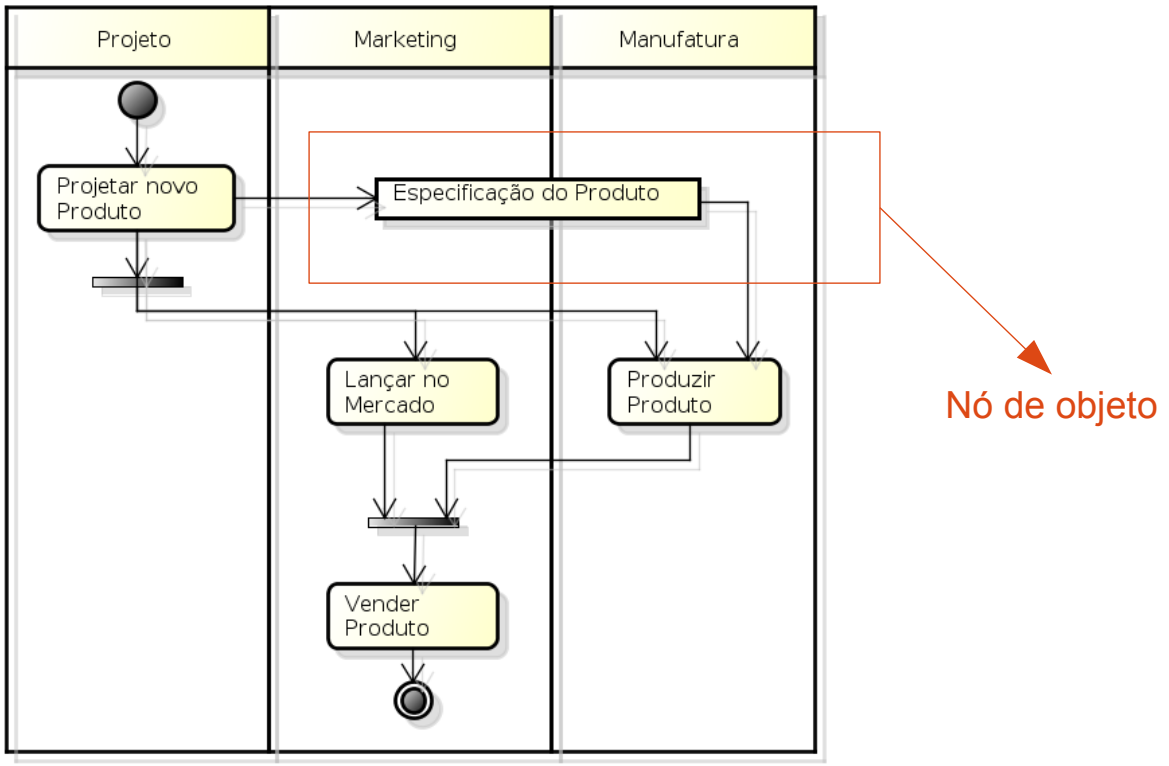


Diagrama de atividade

- **Parâmetros de atividade**
  - **Nós** de **objetos** podem ser utilizados para **prover entradas** e **saídas** de **atividades** e podem ser **anotados** com o **estado**.
  - **Exemplo**:

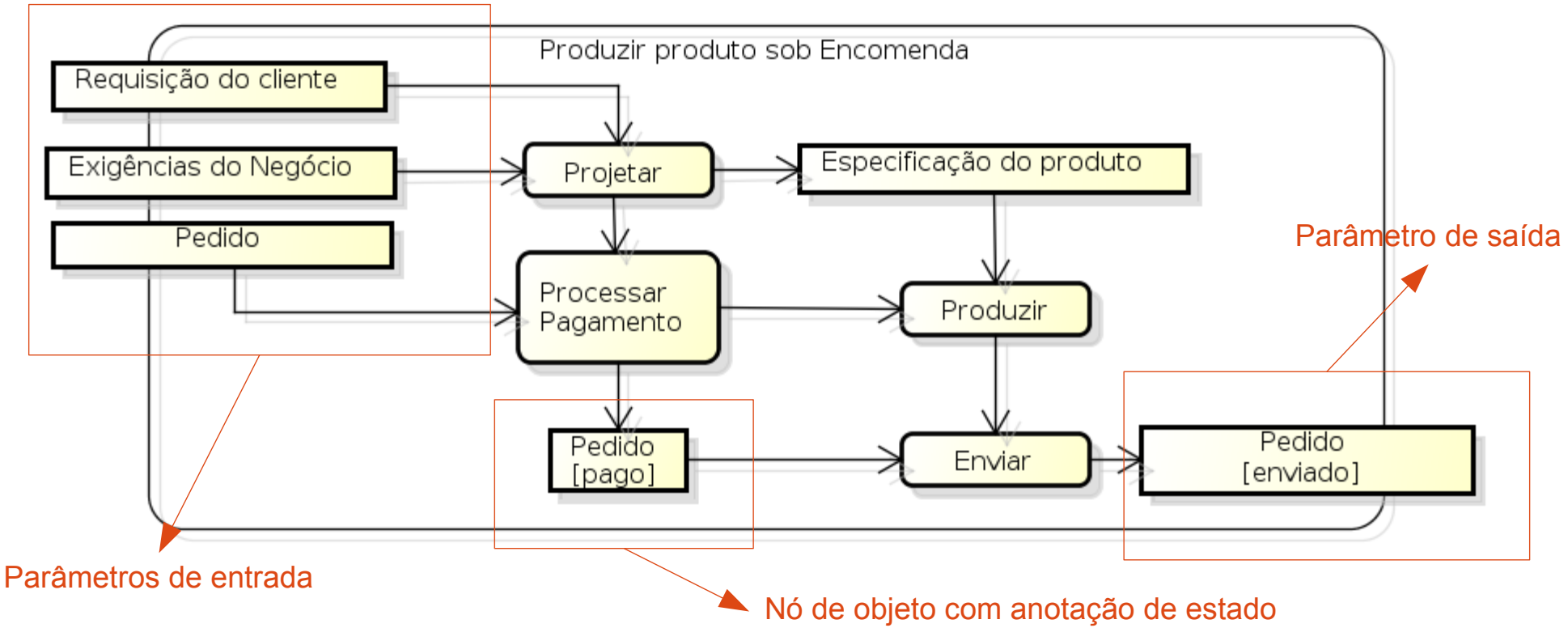
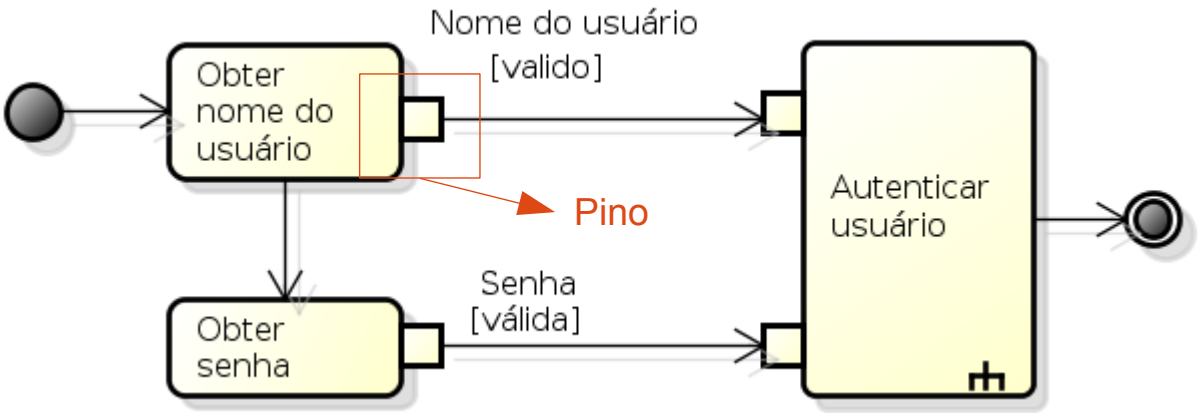


Diagrama de atividade

- **Pinos**
  - O conceito de pino na UML serve para simplificar um diagrama que apresenta muitos nós e fluxos de objetos:



  - Nesta atividade, os pinos representam respectivamente o nome e a senha válidos de um usuário que, depois de serem coletados, são passados a uma atividade que o autentica.
  - O símbolo da atividade de autenticação representa que ela é uma referência a uma atividade que de fato está definida em outro diagrama.

Referências bibliográficas

- ARLOW, J.; NEUSTADT, I. **UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design**. Upper Saddle River, NJ: Addison-Wesley, 2005.
- BOOCH, G. et al. **Object-Oriented Analysis and Design with Applications**. Upper Saddle River, N.J.: Addison-Wesley, 2007.
- BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **The Unified Modeling Language User Guide**. Upper Saddle River, N.J.: Addison-Wesley, 2005.
- JACOBSON, I. et al. **Object-Oriented Software Engineering: A Use Case Driven Approach**. Wokingham, England; Reading, Mass.: ACM Press ; Addison-Wesley Pub., 1992.
- MILES, R.; HAMILTON, K. **Learning UML 2.0**. Sebastopol, CA: O'Reilly Media, 2006.
- PILONE, D.; PITMAN, N. **UML 2.0 in a Nutshell**. Sebastopol, CA: O'Reilly, 2005.
- RUBIN, K. S.; GOLDBERG, A. **Object behavior analysis**. Communications of the ACM, v. 35, n. 9, p. 48–62, 1992.
- WIRFS-BROCK, R.; MCKEAN, A. **Object design : roles, responsibilities, and collaborations**. Boston [Mass.]; London: Addison-Wesley, 2003.