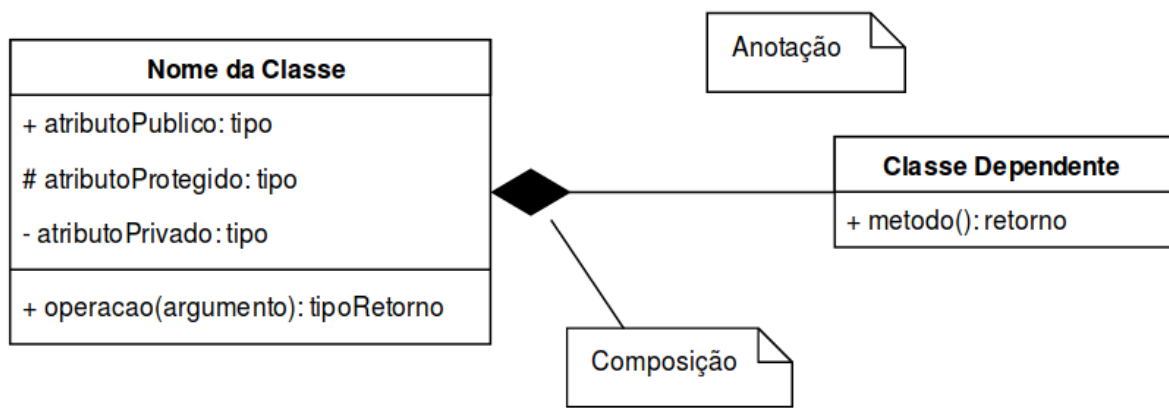


## 1. Diagrama de Classes

### Resumo:

Representa a estrutura estática do sistema, mostrando **classes**, seus **atributos**, **métodos** e os **relacionamentos** entre elas (herança, associação, agregação, composição).

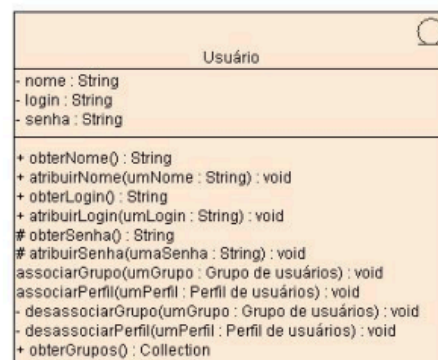
**Usado para:** Modelar a arquitetura do software e entender os objetos e suas interações.



## Diagramas da UML – Representação de classes



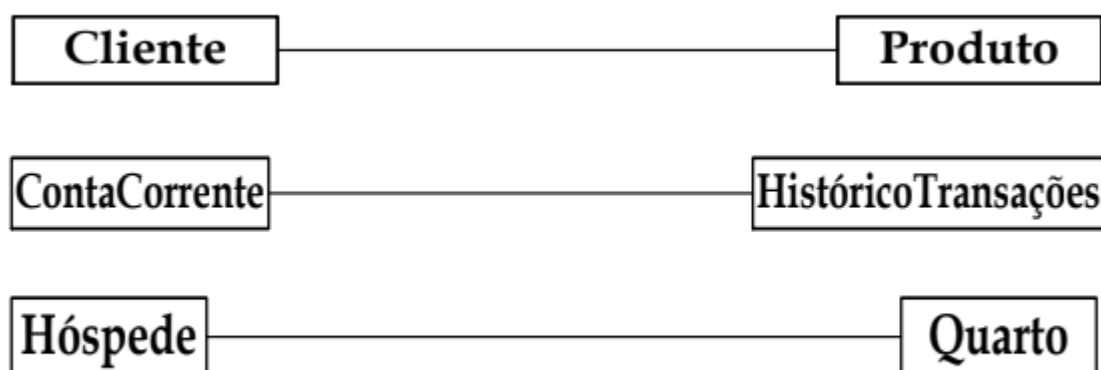
- O encapsulamento visa garantir o acesso apenas sobre operações e atributos disponibilizados pela interface da classe;
- Utiliza-se modificadores de acesso em atributos e operações para se definir a visibilidade dos mesmos:
  - + : Público (public);
  - # : Protegido (protected);
  - ~ : Pacote (Package);
  - - : Privado (Private).



## Associações



- Para representar o fato de que objetos podem **se relacionar** uns com os outros, utiliza-se a **associação**.
- Uma **associação** representa **relacionamentos** (ligações) que são formados entre objetos durante a execução do sistema.
  - embora as associações sejam representadas entre classes do diagrama, tais associações representam ligações possíveis entre *objetos* das classes envolvidas.
  - É um relacionamento estrutural que especifica objetos de um item conectados a objetos de outro item. A partir de uma associação conectando duas classes, você é capaz de navegar do objeto de uma classe até o objeto de outra classe.



## Multiplicidades



Nome	Simbologia
Apenas um	1..1 (ou 1)
Zero ou mais	0..* (ou *)
Um ou mais	1..*
Zero ou um	0..1
Intervalo específico	$L_i..L_s$
Combinação (ex.: 4,5,6,7,9)	4..7,9

$L_i$  Limite inferior da multiplicidade

$L_s$  Limite superior da multiplicidade

## Agregação



- Sejam duas classes associadas, X e Y. Se uma das perguntas a seguir for respondida com um sim, provavelmente há uma agregação onde X é todo e Y é parte. (teste “tem um”, “parte de”)
  - *X tem um ou mais Y?*
  - *Y é parte de X?*



## Notação para uma agregação



- A agregação é representada por um diamante branco, sempre do lado do “Objeto-Todo”.



## Composição



- Existe um tipo de relacionamento TODO-PARTE no qual a participação do objeto da classe **todo** é obrigatória: esse relacionamento chama-se **composição**.
- Quem está iniciando com a orientação a objetos pode sentir alguma dificuldade em entender a diferença entre essas duas formas de associação.
- Esse é sempre um assunto polêmico, pois, dependendo das **regras de negócio**, o mesmo relacionamento que acontece por meio de uma **agregação**, em determinado sistema, pode, em outro, se dar por uma **composição**.



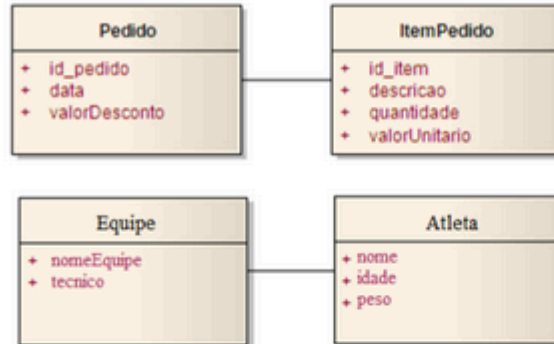
## Notação para uma agregação



- A composição é representada por um diamante negro, também desenhado do lado do “Objeto-Todo”.



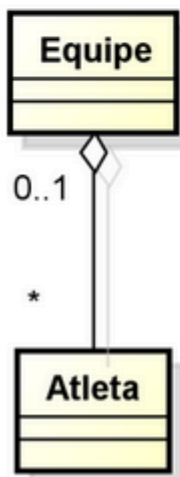
## Agregação ou Composição?



Os relacionamentos **Todo-Parte** acima são representados por:

- **Pedido** (Objeto-Todo) e **ItemPedido** (Objeto-Parte)
- **Equipe** (Objeto-Todo) e **Atleta** (Objeto-Parte)

## Equipe – Jogador - Agregação



- Uma equipe **contém** 0 ou mais atletas
- Um atleta **faz parte de** uma equipe (num dado momento), mas também pode estar fora de alguma equipe (desempregado, por exemplo).
- As partes da agregação podem fazer outras coisas em outras partes da aplicação.
- Uma agregação informa que uma classe faz parte de outra classe, mas não de forma exclusiva: não vincula o tempo de vida do todo e suas partes.
- Na agregação, se uma **Equipe** é destruída, seus **Atletas** continuam existindo, pois podem participar de outras Equipes.

## Composição



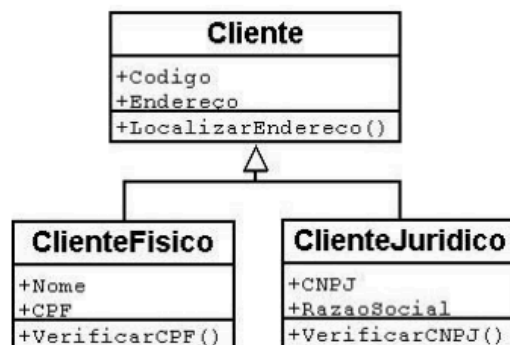
- A composição é uma forma de agregação onde há uma relação mais forte.
- Na composição os elementos que estão contidos dentro de outro objeto dependem dele para existir: as partes não podem existir sem o todo (não faria sentido).
- Informa que uma classe faz parte de outra classe de forma exclusiva
- Eles são criados e destruídos de acordo com o seu container (objeto todo).



## Generalização



- A generalização/especialização é um relacionamento que implementa um mecanismo de herança e é representada no diagrama de classes da seguinte forma:



A seta (fechada e sem preenchimento) aponta sempre da classe mais específica para a classe mais genérica.

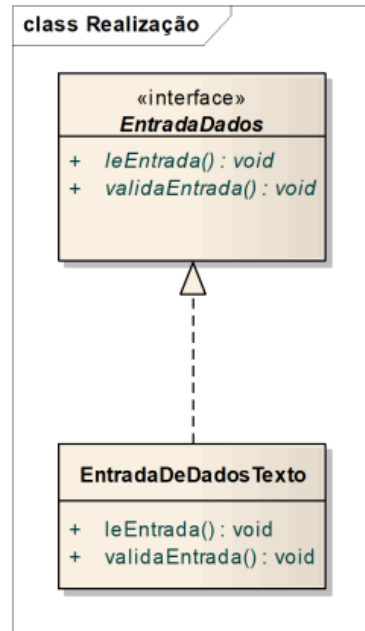




# Realização



- Uma **realização** é um relacionamento no qual um item concretiza o comportamento de outro item.

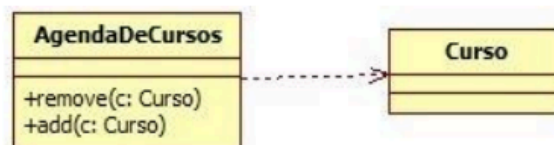




# Dependência



- **Pode-se observar dois tipos diferentes de dependência:**
  - quando uma classe só utiliza a outra como parâmetro para suas operações (mais comum).
  - quando o relacionamento é unilateral, ou seja, a classe dependente interage com a classe independente, que por sua vez, não tem conhecimento da classe dependente.
- **Na UML podem ser criadas dependências entre muitos outros itens, principalmente pacotes.**



**O que é:** modelo **estático** da aplicação: classes, atributos, operações e **relacionamentos** (associação, dependência, generalização/herança, agregação, composição).

**Quando usar:** para estruturar o domínio/arquitetura, identificar responsabilidades e **multiplicidades** (1, 0..\*, etc.).

**Notação-chave:** visibilidade (+ público, – privado), tipos, relações com setas, estereótipos (<<interface>>), **agregação** (losango vazio), **composição** (losango preenchido).

**Erros comuns:** confundir agregação × composição; esquecer multiplicidade e navegabilidade; métodos demais (detalhe de implementação) cedo demais.

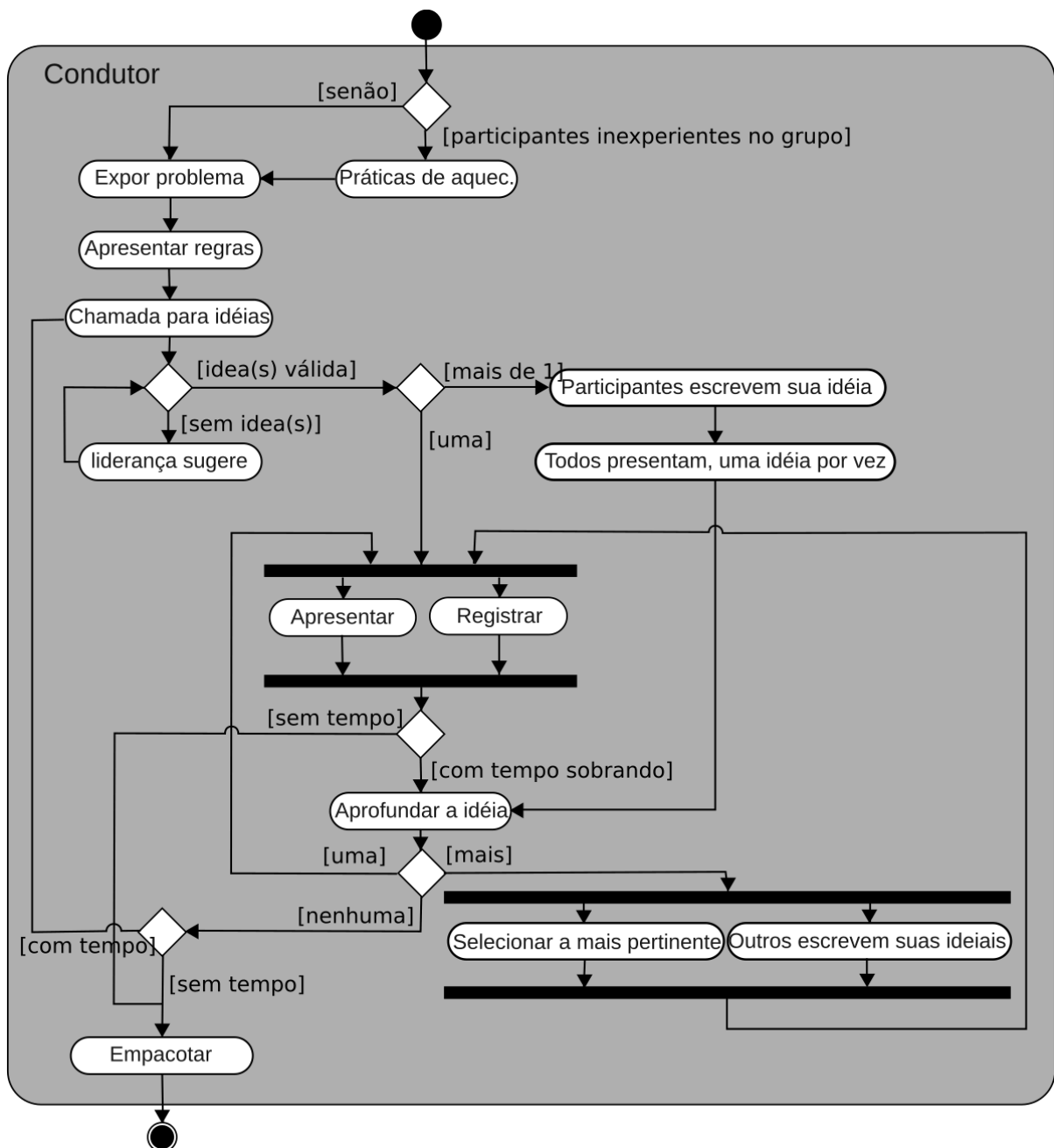
**Exemplo:** Em um sistema de e-commerce, podemos ter as classes **Produto**, **Cliente** e **Pedido**. **Pedido** possui uma lista de **Produto** (composição) e está associado a um **Cliente**. Multiplicidades indicam que um cliente pode ter vários pedidos e um pedido pode conter vários produtos.

## 2. Diagrama de Atividades

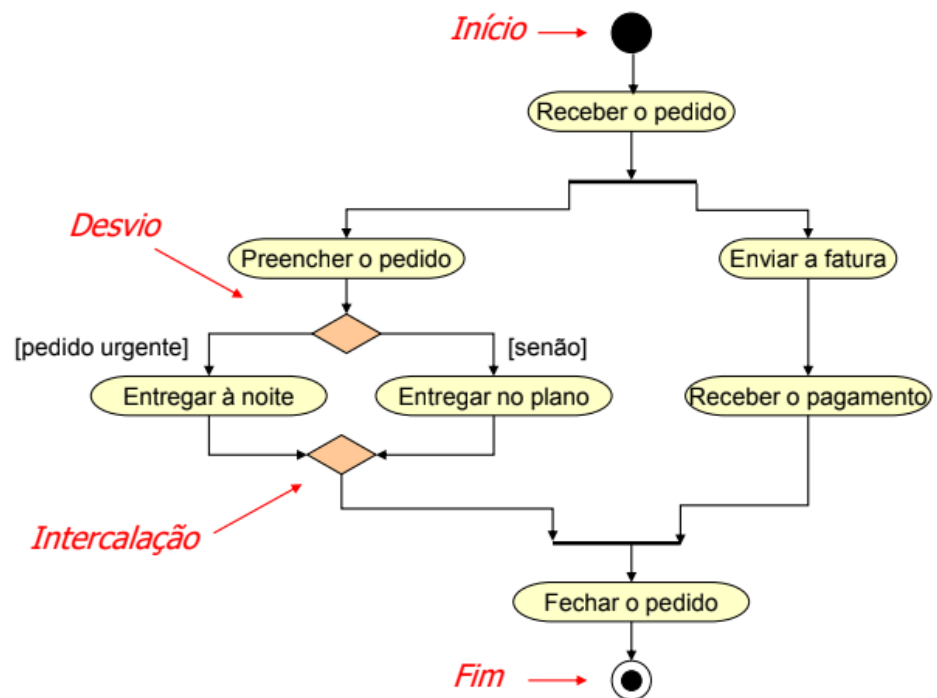
### Resumo:

Mostra o **fluxo de atividades** ou **processos** dentro de um sistema, com decisões, ramificações, paralelismos e ações.

**Usado para:** Representar fluxos de trabalho, como o processo de login ou compra online.



## Diagrama de Atividades

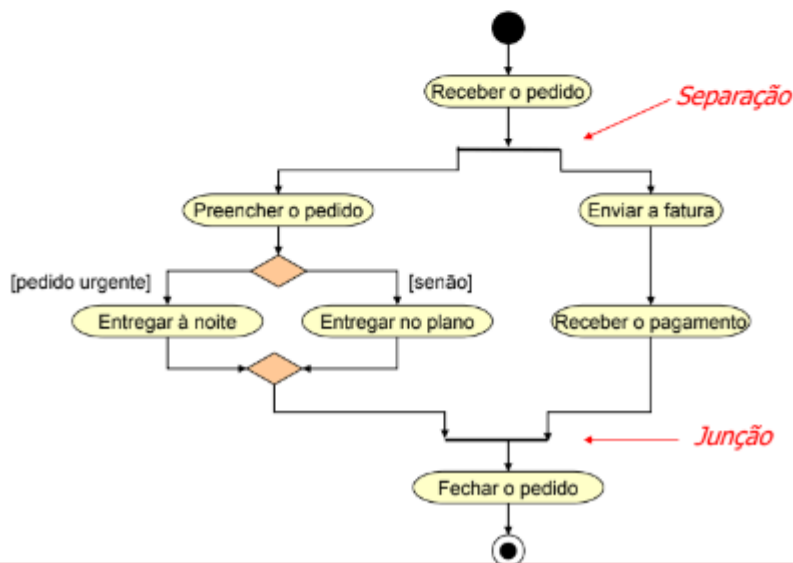


## Diagrama de Atividades

### Comportamento paralelo:

- É indicado por separações e junções (*forks* e *joins*);
- **Separação:**
  - Uma transição de entrada;
  - Várias transições de saída;
  - Uma transição de entrada dispara todas as transições de saída;
- **Junção:**
  - Múltiplas transições de entrada;
  - Sincroniza as atividades que acontecem em paralelo;
  - Separação e junção devem se completar.

## Diagrama de Atividades



**O que é:** fluxo **procedimental** de atividades/ações, com **nós iniciais/finais**, **decisão/merge**, **fork/join** (paralelismo), **swimlanes** (responsáveis) e **objetos** fluindo.

**Quando usar:** modelar **processos/rotinas** (ex.: login, checkout); descrever regras de negócio com caminhos alternativos.

**Erros comuns:** misturar “o quê” (processo) com “quem/código”; esquecer guardas nas setas de decisão; paralelismo sem join.

**Exemplo:** No processo de compra online: iniciar → selecionar produtos → adicionar ao carrinho → [decisão: cliente cadastrado?] → se sim, faz login; se não, realiza cadastro → escolher forma de pagamento → confirmar pedido → finalizar.

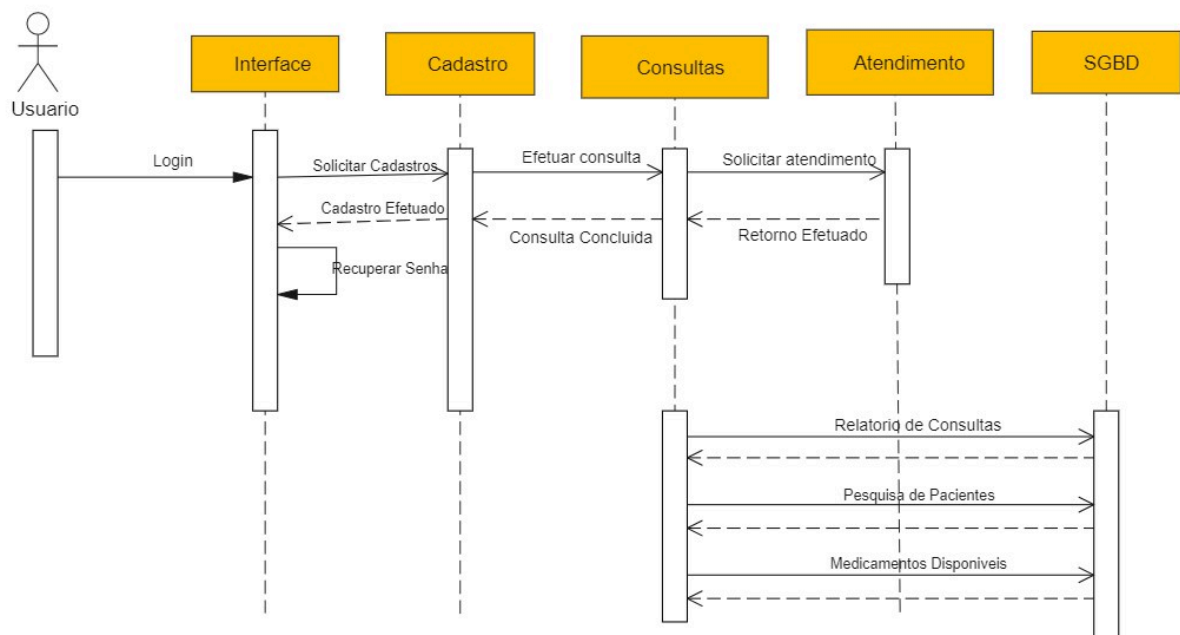
---

### 3. Diagrama de Sequência

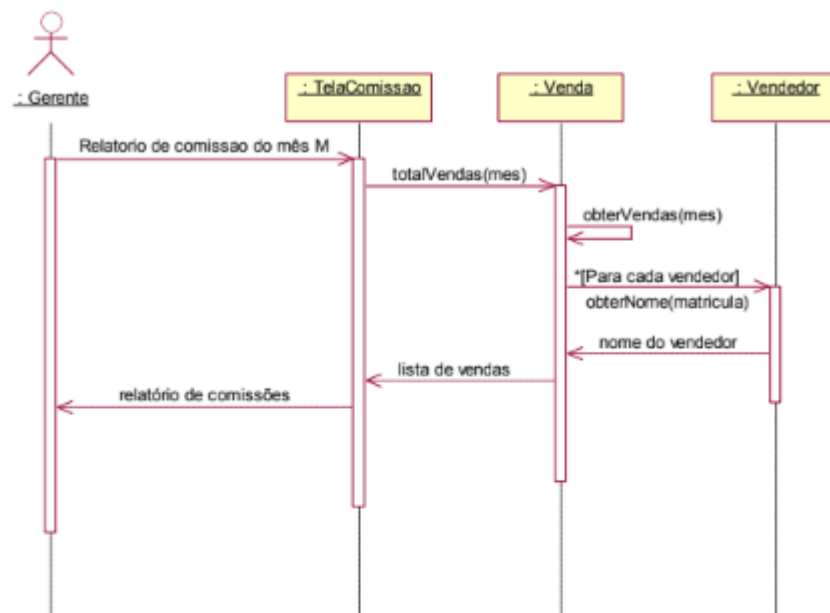
#### Resumo:

Descreve como os **objetos interagem no tempo**, focando na troca de mensagens entre eles para realizar uma funcionalidade.

**Usado para:** Visualizar o comportamento de um caso de uso específico.

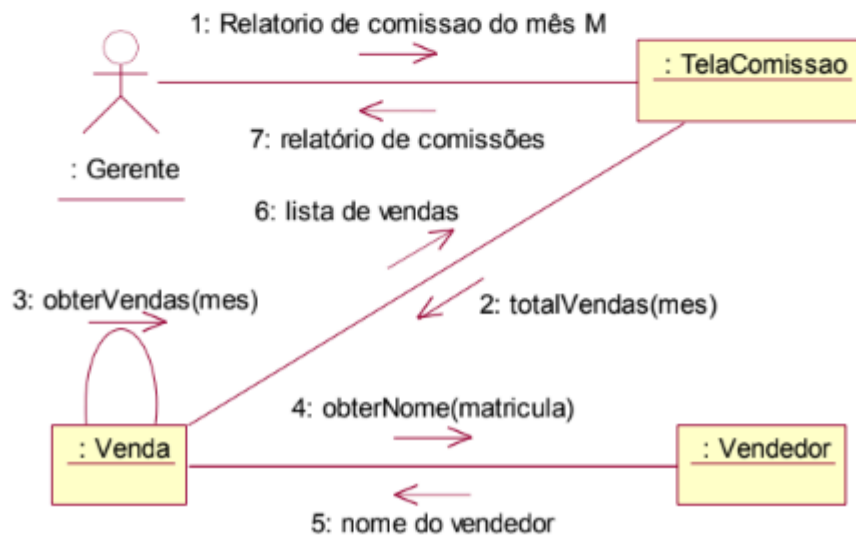


## Diagrama de Sequência



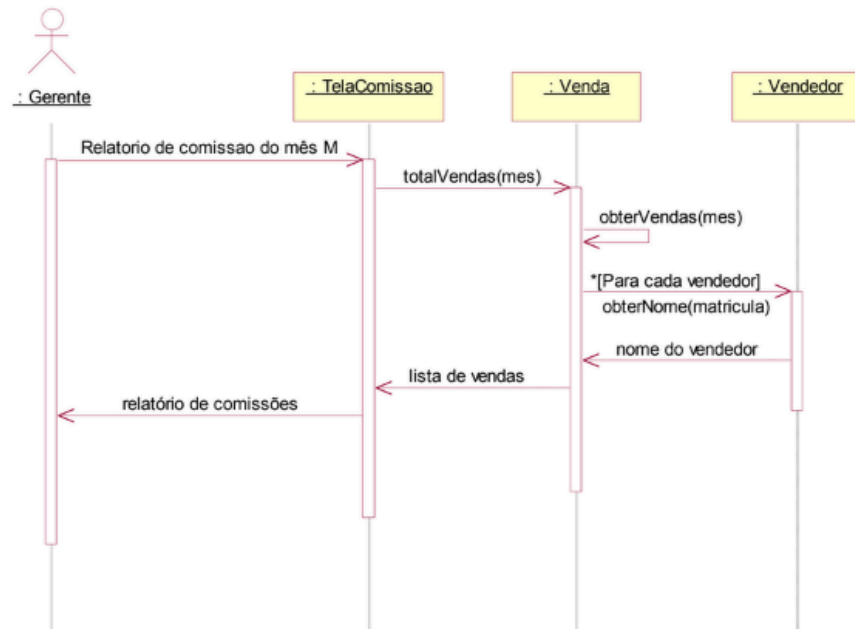
Dúvidas sobre esse resumo ou sobre o tópico em geral? Comente abaixo.

## Diagrama de Comunicação



## Diagrama de Sequência (Interação)

---



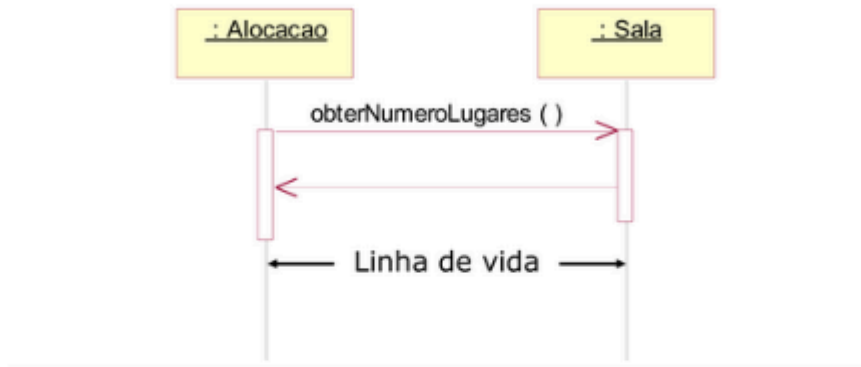


## Diagrama de Sequência (Interação)

---

### Linha de vida:

- Representa a vida do objeto dentro de um determinado período de tempo.

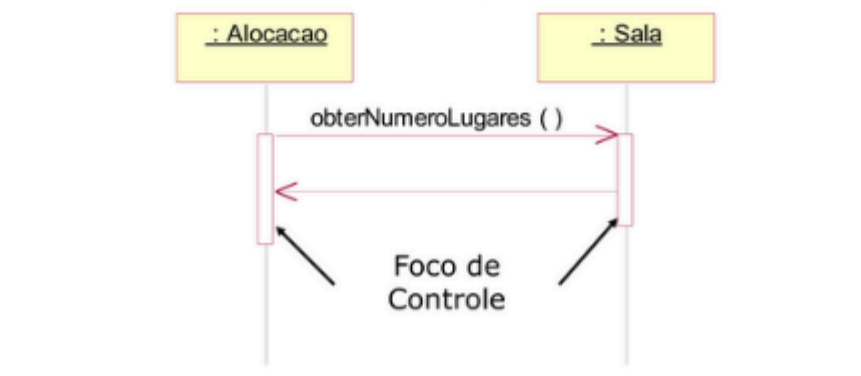


## Diagrama de Sequência (Interação)

---

### Foco de controle:

- Representa o tempo durante o qual um objeto fica com o controle do fluxo de execução.



**O que é:** interação **no tempo** entre objetos/partes (lifelines, barras de ativação), **mensagens** síncronas/assíncronas, retornos e **fragmentos** (alt/opt/loop/par).

**Quando usar:** detalhar um **caso de uso** ou cenário crítico (incluindo exceções) e validar a colaboração entre componentes.

**Erros comuns:** mensagens sem ordem clara; não indicar quem inicia; esquecer condições/loops; “pular” camadas (UI → DB direto).

**Exemplo:** No caso de “realizar login”, o diagrama pode mostrar: **Usuário** envia **dados de login** para **TelaLogin** → esta envia para **ControladorLogin** → o controlador consulta **BancoDeDados** → retorna confirmação → a tela exibe mensagem de sucesso.

---

## 4. Diagrama de Estados

### Resumo:

Mostra os **estados possíveis de um objeto** e as **transições** causadas por eventos.

**Usado para:** Modelar comportamentos dinâmicos, como o ciclo de vida de um pedido (Criado → Pago → Enviado → Entregue).



**O que é:** ciclo de vida de um objeto, com **estados**, **transições** rotuladas como **evento** [guarda] / **ação**, **entry/exit/do**, estado inicial/final e **subestados**.

**Quando usar:** objetos com comportamento **dirigido a eventos** (pedido, sessão, tarefa, dispositivo).

**Erros comuns:** usar para processo global (que é atividade); transições sem evento; não modelar condições de guarda.

**Exemplo:** Um pedido no e-commerce pode estar nos estados: **Criado** → (pagamento confirmado) → **Pago** → (produto enviado) → **Enviado** → (produto recebido) → **Entregue**. Caso o pagamento seja negado, há transição para **Cancelado**.

---

## ✓ 5. Casos de Teste

### Resumo:

São descrições detalhadas que testam se uma funcionalidade do sistema está funcionando como esperado. Incluem **entrada, ação esperada e resultado esperado**.

**Usado para:** Verificar se os requisitos do sistema foram corretamente implementados.

Identificação	UFC-ETF-LO-CT-AU1	
Itens a testar	Verificar se a tentativa de efetuar o login utilizando uma senha não cadastrada exibe mensagem apropriada	
Entradas	Campo	Valor
	Utilizador	Ismaylesantos
	Senha	Great123
Saídas Esperadas	Campo	Valor
	Nome	Ismaylesantos
	Senha	
	Mensagem	Erro ao entrar
Ambiente	Banco de dados de teste	
Procedimentos	Login – UFC-ETF-LO-PT-AU	
Dependências	Não aplicável	

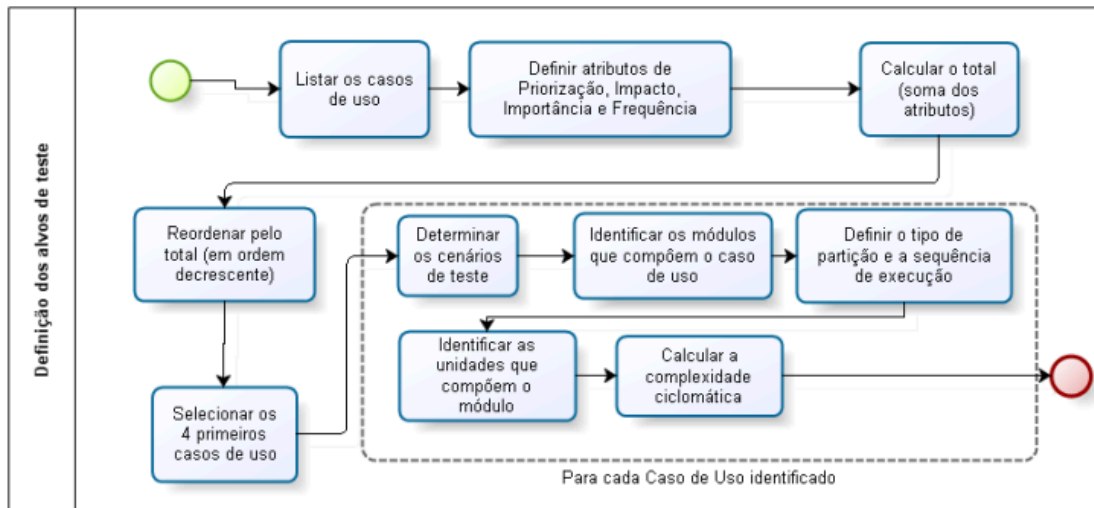
**O que são:** especificações verificáveis: **ID, título, objetivo, pré-condições, dados/steps, resultado esperado, pós-condições**, rastreio ao requisito.

**Técnicas úteis:** **particionamento por equivalência, análise de valores-limite, tabela de decisão, estado-transição**; níveis (unidade, integração, sistema, aceitação) e tipos (caixa-preta/caixa-branca).

**Erros comuns:** casos sem oráculo (resultado esperado vago), falta de dados, ignorar cenários negativos e limites.



## Fluxo geral do trabalho de testes



## Como determinar os alvos dos teste?



- Os alvos devem ser determinados usando **critérios de Priorização dos casos de uso para teste**;
  - **PRIORIZAÇÃO DOS CASOS DE USO:**
    - 1 – Baixa prioridade
    - 2 – Média prioridade
    - 3 - Alta prioridade
  - **IMPACTO** - Impacto de falha no requisito sobre o sistema:
    - 1 – Baixo impacto
    - 2 – Médio impacto
    - 3 – Alto impacto
  - **IMPORTÂNCIA** - Comparação de importância entre os requisitos. Este valor tem um grau de subjetividade:
    - 1 – Baixa importância
    - 2 – Média importância
    - 3 – Alta importância
  - **FREQUÊNCIA** - relacionado à quantidade de vezes que o requisito é executado durante um período de utilização do sistema;

### Exemplo:

- **ID:** CT-001
- **Objetivo:** Validar login com credenciais corretas.
- **Pré-condições:** Usuário já cadastrado.
- **Passos:** 1) Acessar página de login; 2) Inserir e-mail e senha válidos; 3) Clicar em “Entrar”.
- **Resultado esperado:** Sistema redireciona para página inicial do usuário.



Bruno Vilela Oliveira

11 de ago.



Pessoal, boa noite.

Um dos colegas da turma me pediu para elaborar um exemplo de caso de teste extraído a partir de um caso de uso diferente do exemplo da aula.

fizemos um rapidamente aqui na aula de dúvida que foi baseado no caso de uso apresentado nos slides (slide 25) da aula sobre casos de uso (caso de uso cadastrar usuário).

Segue a resolução do problema (derivar casos de teste a partir de casos de uso):

Parte 1 - identificação dos cenários para o caso de uso

Cenário 1 - fluxo principal

Cenário 2 - Fluxo principal - fluxo alternativo 1

Parte 2 - Derivação dos casos de teste

v - valor informado

i - valor informado inválido

CTX - Caso de teste X (sendo x um número sequencial)

ID	nome, sobrenome, e-mail, telefone	resultado esperado
CT1 - v	v v v	cadastro realizado com sucesso
CT2 - v	v i v	O sistema exibe a mensagem de erro: "E-mail já cadastrado - informe um outro e-mail." e retorna ao preenchimento dos dados

O cenário 1 ocorre quando o caso de uso é realizado passando apenas pelo fluxo principal.

O cenário 2 ocorre quando é informado um e-mail já cadastrado (o que foi classificado como inválido para este dado na operação de cadastro, já que não se pode realizar mais de um cadastro com o mesmo e-mail. Por isso a nomenclatura (valor informado inválido) para o caso de teste.



Adicionar comentário para a turma...



Os **requisitos funcionais** descrevem **o que o sistema deve fazer**, ou seja, as **funcionalidades** e comportamentos que ele precisa oferecer para atender às necessidades do usuário ou do negócio.


- **Exemplos:**

- "O sistema deve permitir que o usuário realize login com e-mail e senha."
- "O sistema deve calcular o valor do frete com base no CEP informado."

Já os **requisitos não funcionais** definem **como o sistema deve se comportar** ou **as restrições/qualidades** que ele precisa atender, independentemente da funcionalidade.

- **Exemplos:**

- "O sistema deve responder a qualquer solicitação em no máximo 2 segundos."
- "O sistema deve estar disponível 99,9% do tempo."

 Resumindo:

- **Funcionais** → O **que** o sistema faz.
- **Não funcionais** → **Como** o sistema faz.

---

Acho que é isso!